Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

SISTEMA DE COMUNICACIÓN XBEE PARA UNA RED DE SENSORES

DOCUMENTO 2- *DOCUMENTACIÓN*

Alumno/Alumna: Estévez, Santé, José Javier

Director/Directora: Oleagordía, Aguirre, Iñigo Javier

Curso: 2017-2018

Fecha: 23/07/2018

Índice- Documentación Anexa

1 XBee-Quick-Reference-Guide 3

2 XBee/XBee-PRO® RF Modules 4

3 XBee SIP ADAPTER 71

4 ATZB-24-A2,B0_ZigBit 81

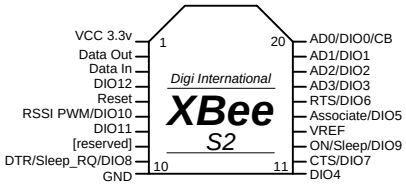
5 ATmega168_summary 109

6 ST3232_Convertidor TTL-RS232 143

7 AVR20XX_SerialNet_User_Guide 156

8 Arduino_user_manual_es 233

9 Manual+Programacion+Arduino 281

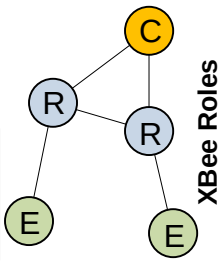


XBee S2 Quick Reference Guide

IEEE 802.15.4 = Zigbee Protocol. XBee is a microcontroller made by digi which uses the Zigbee protocol.

The XBee uses 3.3V and has a smaller pin spacing than most breadboards/proto boards. Because of this, it is often useful to purchase a kit to interface the XBee with a breadboard.

Sept/2012 http://tunnelsup.com



XBee Roles

Coordinator – 1 required in every network
In charge of setting up the network
Can never sleep

Router – multiple may exist
Can relay signals from other routers/EPs
Can never sleep

End Point – multiple may exist
Cannot relay signals
Can sleep to save power

Specs

Operating Voltage: 2.1 – 3.6V	Digital I/O pins: 11	RF Data Rate: 250kbps
Operating Current: 40mA@3.3V	Analog input pins: 4	Throughput speed: 35kbps
Indoor range: 40 Meters	Mesh routable	Frequency: ISM 2.4GHz
Line of sight range: 120 Meters	Self Healing network	OK Temp: -40 to 85C
Max Analog Pin Reading: 1.2V	Firmware: ZB ZigBee	

XBee Modes

Transparent – Communication through the XBee. If data is not generated from the XBee itself then both XBees should be set to AT.
Command – Communication to the XBee. If one XBee is sensing data, that XBee should be in AT mode while the receiving one should be in API mode.

XBee Setup

Connect the XBee to a TTL Serial FTDI adapter – OR – Arduino hack: Connect RX to RX, TX to TX, RESET to ground to bypass the Arduino entirely and get serial to XBee.
Use the free X-CTU software to configure the XBee.
Baud: 9600 – FC: Hardware – Data Bits: 8 – Parity: None – Stop Bits: 1

Basic Settings

PAN ID – The network to communicate over. If 0, the XBee will join any.
DH/DL – Destination Serial number. Used to send to a specific XBee's Serial. Set to 0 to send to just the Coorindator. Set to 0x0000000000FFFF to broadcast.
JV – Router/EP should be set to 1 so it rejoins the network on startup

Pin Settings

For pin settings to work, receiver XBee must be in API mode
D0 – Set pin 0 to start sensing
IR – Collect data on sensing pins every XX millisecs

Arduino Connectivity:
Arduino TX connects to XBee RX (Data in)
Arduino RX connects to XBee TX (Data out)

Arduino Integration:
Data sent to Serial.print() will go out TX port of Arduino which is then connected to the RX port of XBee. If XBee is in AT mode it will transmit it wirelessly. Data received from XBee will be sent to the Serial.

Arduino Example: Read an analog value using API

```
// Remote XBee: AT, Base XBee: API
if (Serial.available() >= 21) {   // Make sure the frame is all there
 if (Serial.read() == 0x7E) {     // 7E is the start byte
  for (int i = 1; i<19; i++) {    // Skip ahead to the analog data
    byte discardByte = Serial.read();
  }
  int analogMSB = Serial.read(); // Read the first analog byte data
  int analogLSB = Serial.read();  // Read the second byte
  int analogReading = analogLSB + (analogMSB * 256);
 }
}
```

API format for Remote AT Command Request

Byte	Example	Description
0	0x7e	Start byte – Indicates beginning of data frame
1	0x00	Length – Number of bytes (ChecksumByte# – 1 – 2)
2	0x10	
3	0x17	Frame type - 0x17 means this is a AT command Request
4	0x52	Frame ID – Command sequence number
5	0x00	64-bit Destination Address (Serial Number)
6	0x13	MSB is byte 5, LSB is byte 12
7	0xA2	
8	0x00	0x0000000000000000 = Coordinator
9	0x40	0x000000000000FFFF = Broadcast
10	0x77	
11	0x9C	
12	0x49	
13	0xFF	Destination Network Address
14	0xFE	(Set to 0xFFFE to send a broadcast)
15	0x02	Remote command options (set to 0x02 to apply changes)
16	0x44 (D)	AT Command Name (Two ASCII characters)
17	0x02 (2)	
18	0x04	Command Parameter (queries if not present)
19	0XF5	Checksum

Arduino Example: Change the pin setting on a remote Xbee

```
// Remote XBee: AT, Base XBee: API
Serial.write(0x7E);  // Sync up the start byte
Serial.write((byte)0x0);    // Length MSB (always 0)
Serial.write(0x10);  // Length LSB
Serial.write(0x17);  // 0x17 is the frame ID for sending an AT command
Serial.write((byte)0x0);    // Frame ID (no reply needed)
Serial.write((byte)00);     // Send the 64 bit destination address
Serial.write((byte)00);     // (Sending 0x000000000000FFFF (broadcast))
Serial.write((byte)00);
Serial.write((byte)00);
Serial.write((byte)00);
Serial.write((byte)00);
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF); // Destination Network
Serial.write(0xFE); // (Set to 0xFFFE if unknown)
Serial.write(0x02); // Set to 0x02 to apply these changes
Serial.write('D');    // AT Command: D1
Serial.write('1');
Serial.write(0x05); // Set D1 to be 5 (Digital Out HIGH)
long chexsum = 0x17 + 0xFF + 0xFF +0xFF + 0xFE + 0x02 + 'D' + '1' + 0x05;
Serial.write( 0xFF – (chexsum & 0xFF)); // Checksum
```

API format for I/O Data Sample RX Indicaton

Byte	Example	Description
0	0x7e	Start byte – Indicates beginning of data frame
1	0x00	Length – Number of bytes (ChecksumByte# – 1 – 2)
2	0x14	
3	0x92	Frame type - 0x92 indicates this will be a data sample
4	0x00	64-bit Source Address (Serial Number)
5	0x13	MSB is byte 4, LSB is byte 11
6	0xA2	
7	0x00	
8	0x40	
9	0x77	
10	0x9C	
11	0x49	
12	0x36	Source Network Address – 16 Bit
13	0x6A	
14	0x01	Receive Opts. 01=Packet Acknowledged. 02=Broadcast packet
15	0x01	Number of sample sets. Always set to 1 due to XBEE limitations
16	0x00	Digital Channel Mask – Indicates which pins are set to DIO
17	0x20	
18	0x01	Analog Channel Mask – Indicates which pins are set to ADC
19	0x00	Digital Sample Data (if any) – Reads the same as Digital Mask
20	0x14	
21	0x04	Analog Sample data (if any)
22	0x25	There will be two bytes here for every pin set for ADC
23	0xF5	Checksum(0xFF - the 8 bit sum of the bytes from byte 3 to this byte)

Sleep Mode

Endpoints can sleep to save power. An endpoint that only wakes up every 5 minutes to send data may only be awake for 6 seconds a day.
SM – 4 = Cyclic sleep
SP – Sleep time (up to 28 secs)
SN – Number of sleep cycles
ST – Time awake

Pin I/O Options

0 – Disabled
1 – N/A
2 – ADC
3 – Digital IN
4 – Digital OUT, LOW
5 – Digital OUT, HIGH

Digital Ch Mask

First Byte
n/a n/a n/a D12 D11 D10 n/a n/a
Second Byte
D7 D6 D5 D4 D3 D2 D1 D0
Example:
0x00 0x0D = 0000 0000 0000 1101
Pins D3, D2 and D0

Analog Ch Mask

(volt) n/a n/a n/a A3 A2 A1 A0
Example:
0x05 = 0000 0101 = Pin A2 and A0

Notes

XBee®/XBee-PRO® RF Modules

XBee®/XBee-PRO® RF Modules

RF Module Operation

RF Module Configuration

Appendices

Product Manual v1.xEx - 802.15.4 Protocol
For RF Module Part Numbers: XB24-A...-001, XBP24-A...-001

IEEE® 802.15.4 RF Modules by Digi International

Digi International Inc.
11001 Bren Road East
Minnetonka, MN 55343
877 912-3444 or 952 912-3444
http://www.digi.com

90000982_B
2009.09.23

Technical Support:	Phone:	(866) 765-9885 toll-free U.S.A. & Canada (801) 765-9885 Worldwide
		8:00 am - 5:00 pm [U.S. Mountain Time]
	Live Chat:	www.digi.com
	Online Support:	http://www.digi.com/support/eservice/login.jsp
	Email:	rf-experts@digi.com

Contents

1. XBee®/XBee-PRO® RF Modules 4

Key Features 4
Worldwide Acceptance 4
Specifications 5
Mechanical Drawings 5
Mounting Considerations 6
Pin Signals 7
Electrical Characteristics 8

2. RF Module Operation 10

Serial Communications 10
UART Data Flow 10
Transparent Operation 11
API Operation 11
Flow Control 12
ADC and Digital I/O Line Support 13
I/O Data Format 13
API Support 14
Sleep Support 14
DIO Pin Change Detect 14
Sample Rate (Interval) 14
I/O Line Passing 15
Configuration Example 15
XBee®/XBee-PRO® Networks 16
Peer-to-Peer 16
NonBeacon (w/ Coordinator) 16
Association 17
XBee®/XBee-PRO® Addressing 20
Unicast Mode 20
Broadcast Mode 20
Modes of Operation 21
Idle Mode 21
Transmit/Receive Modes 21
Sleep Mode 23
Command Mode 25

3. RF Module Configuration 26

Programming the RF Module 26
Programming Examples 26
Remote Configuration Commands 27
Sending a Remote Command 27
Applying Changes on Remote 27
Remote Command Responses 27
Command Reference Tables 27
Command Descriptions 36
API Operation 57
API Frame Specifications 57
API Types 58

Appendix A: Agency Certifications 64

United States (FCC) 64
OEM Labeling Requirements 64
FCC Notices 64
FCC-Approved Antennas (2.4 GHz) 65
Approved Antennas 67
Canada (IC) 68
Labeling Requirements 68
Japan 68
Labeling Requirements 68

Appendix B: Additional Information 69

1-Year Warranty 69

1. XBee®/XBee-PRO® RF Modules

The XBee and XBee-PRO RF Modules were engineered to meet IEEE 802.15.4 standards and support the unique needs of low-cost, low-power wireless sensor networks. The modules require minimal power and provide reliable delivery of data between devices.

The modules operate within the ISM 2.4 GHz frequency band and are pin-for-pin compatible with each other.

Key Features

Long Range Data Integrity

XBee

- Indoor/Urban: up to 100’ (30 m)
- Outdoor line-of-sight: up to 300’ (90 m)
- Transmit Power: 1 mW (0 dBm)
- Receiver Sensitivity: -92 dBm

XBee-PRO

- Indoor/Urban: up to 300’ (90 m), 200' (60 m) for International variant
- Outdoor line-of-sight: up to 1 mile (1600 m), 2500' (750 m) for International variant
- Transmit Power: 63mW (18dBm), 10mW (10dBm) for International variant
- Receiver Sensitivity: -100 dBm

RF Data Rate: 250,000 bps

Advanced Networking & Security

Retries and Acknowledgements

DSSS (Direct Sequence Spread Spectrum)

Each direct sequence channels has over 65,000 unique network addresses available

Source/Destination Addressing

Unicast & Broadcast Communications

Point-to-point, point-to-multipoint and peer-to-peer topologies supported

Low Power

XBee

- TX Peak Current: 45 mA (@3.3 V)
- RX Current: 50 mA (@3.3 V)
- Power-down Current: < 10 µA

XBee-PRO

- TX Peak Current: 250mA (150mA for international variant)
- TX Peak Current (RPSMA module only): 340mA (180mA for international variant
- RX Current: 55 mA (@3.3 V)
- Power-down Current: < 10 µA

ADC and I/O line support

Analog-to-digital conversion, Digital I/O

I/O Line Passing

Easy-to-Use

No configuration necessary for out-of box RF communications

Free X-CTU Software (Testing and configuration software)

AT and API Command Modes for configuring module parameters

Extensive command set

Small form factor

Worldwide Acceptance

FCC Approval (USA) Refer to Appendix A [p64] for FCC Requirements. Systems that contain XBee®/XBee-PRO® RF Modules inherit Digi Certifications.

ISM (Industrial, Scientific & Medical) **2.4 GHz frequency band**

Manufactured under **ISO 9001:2000** registered standards

XBee®/XBee-PRO® RF Modules are optimized for use in the United States, Canada, Australia, Japan, and Europe. Contact Digi for complete list of government agency approvals.

Specifications

Table 1-01. Specifications of the XBee®/XBee-PRO® RF Modules

Specification	XBee	XBee-PRO
Performance		
Indoor/Urban Range	Up to 100 ft (30 m)	Up to 300 ft. (90 m), up to 200 ft (60 m) International variant
Outdoor RF line-of-sight Range	Up to 300 ft (90 m)	Up to 1 mile (1600 m), up to 2500 ft (750 m) international variant
Transmit Power Output (software selectable)	1mW (0 dBm)	63mW (18dBm)* 10mW (10 dBm) for International variant
RF Data Rate	250,000 bps	250,000 bps
Serial Interface Data Rate (software selectable)	1200 bps - 250 kbps (non-standard baud rates also supported)	1200 bps - 250 kbps (non-standard baud rates also supported)
Receiver Sensitivity	-92 dBm (1% packet error rate)	-100 dBm (1% packet error rate)
Power Requirements		
Supply Voltage	2.8 – 3.4 V	2.8 – 3.4 V
Transmit Current (typical)	45mA (@ 3.3 V)	250mA (@3.3 V) (150mA for international variant) RPSMA module only: 340mA (@3.3 V) (180mA for international variant)
Idle / Receive Current (typical)	50mA (@ 3.3 V)	55mA (@ 3.3 V)
Power-down Current	< 10 µA	< 10 µA
General		
Operating Frequency	ISM 2.4 GHz	ISM 2.4 GHz
Dimensions	0.960” x 1.087” (2.438cm x 2.761cm)	0.960” x 1.297” (2.438cm x 3.294cm)
Operating Temperature	-40 to 85º C (industrial)	-40 to 85º C (industrial)
Antenna Options	Integrated Whip, Chip or U.FL Connector, RPSMA Connector	Integrated Whip, Chip or U.FL Connector, RPSMA Connector
Networking & Security		
Supported Network Topologies	Point-to-point, Point-to-multipoint & Peer-to-peer	
Number of Channels (software selectable)	16 Direct Sequence Channels	12 Direct Sequence Channels
Addressing Options	PAN ID, Channel and Addresses	PAN ID, Channel and Addresses
Agency Approvals		
United States (FCC Part 15.247)	OUR-XBEE	OUR-XBEEPRO
Industry Canada (IC)	4214A XBEE	4214A XBEEPRO
Europe (CE)	ETSI	ETSI (Max. 10 dBm transmit power output)*
Japan	R201WW07215214	R201WW08215111 (Max. 10 dBm transmit power output)*
Austraila	C-Tick	C-Tick

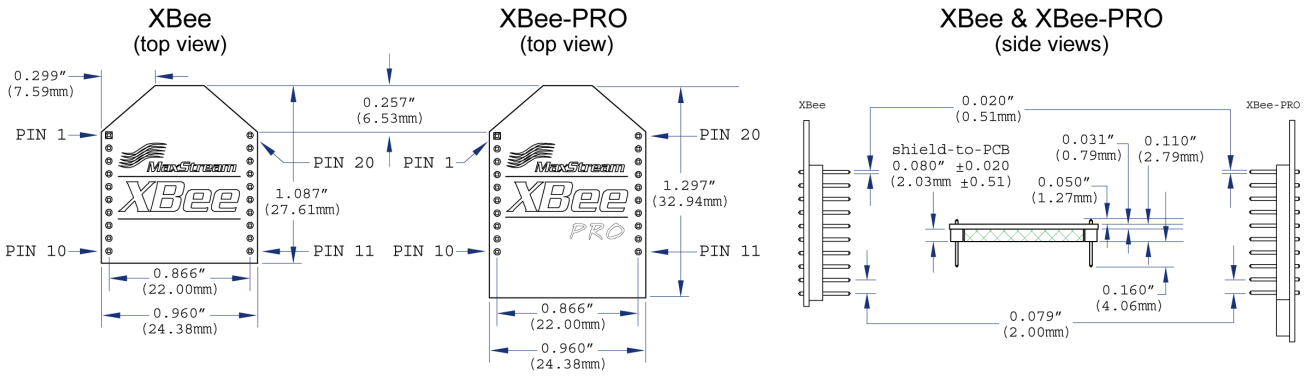
* See Appendix A for region-specific certification requirements.

Antenna Options: The ranges specified are typical when using the integrated Whip (1.5 dBi) and Dipole (2.1 dBi) antennas. The Chip antenna option provides advantages in its form factor; however, it typically yields shorter range than the Whip and Dipole antenna options when transmitting outdoors.For more information, refer to the "XBee Antennas" Knowledgebase Article located on Digi's Support Web site

Mechanical Drawings

Figure 1-01. Mechanical drawings of the XBee®/XBee-PRO® RF Modules (antenna options not shown)

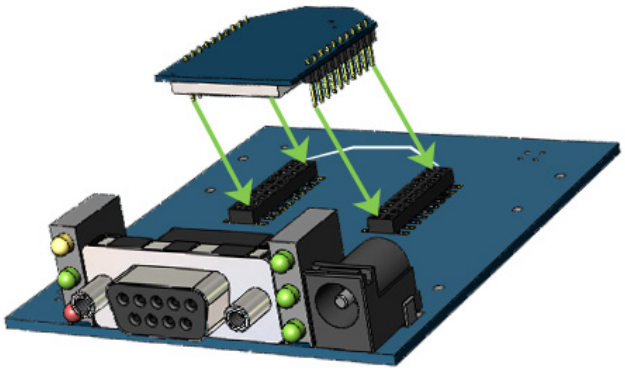
The XBee and XBee-PRO RF Modules are pin-for-pin compatible.



Mounting Considerations

The XBee®/XBee-PRO® RF Module was designed to mount into a receptacle (socket) and therefore does not require any soldering when mounting it to a board. The XBee Development Kits contain RS-232 and USB interface boards which use two 20-pin receptacles to receive modules.

Figure 1-02. XBee Module Mounting to an RS-232 Interface Board.



The receptacles used on Digi development boards are manufactured by Century Interconnect. Several other manufacturers provide comparable mounting solutions; however, Digi currently uses the following receptacles:

- Through-hole single-row receptacles -
  Samtec P/N: MMS-110-01-L-SV (or equivalent)
- Surface-mount double-row receptacles -
  Century Interconnect P/N: CPRMSL20-D-0-1 (or equivalent)
- Surface-mount single-row receptacles -
  Samtec P/N: SMM-110-02-SM-S

Digi also recommends printing an outline of the module on the board to indicate the orientation the module should be mounted.

Pin Signals

Figure 1-03. XBee®/XBee-PRO® RF Module Pin Numbers

(top sides shown - shields on bottom)

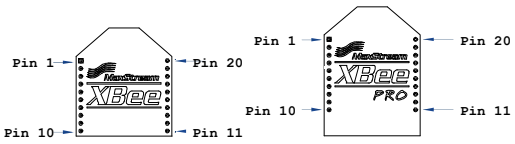


Table 1-02. Pin Assignments for the XBee and XBee-PRO Modules
(Low-asserted signals are distinguished with a horizontal line above signal name.)

Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / **CONFIG**	Input	UART Data In
4	DO8*	Output	Digital Output 8
5	**RESET**	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI	Output	PWM Output 0 / RX Signal Strength Indicator
7	PWM1	Output	PWM Output 1
8	[reserved]	-	Do not connect
9	DTR / SLEEP_RQ / DI8	Input	Pin Sleep Control Line or Digital Input 8
10	GND	-	Ground
11	AD4 / DIO4	Either	Analog Input 4 or Digital I/O 4
12	**CTS** / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	ON / **SLEEP**	Output	Module Status Indicator
14	VREF	Input	Voltage Reference for A/D Inputs
15	Associate / AD5 / DIO5	Either	Associated Indicator, Analog Input 5 or Digital I/O 5
16	**RTS** / AD6 / DIO6	Either	Request-to-Send Flow Control, Analog Input 6 or Digital I/O 6
17	AD3 / DIO3	Either	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Either	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Either	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0	Either	Analog Input 0 or Digital I/O 0

* Function is not supported at the time of this release

Design Notes:

- Minimum connections: VCC, GND, DOUT & DIN
- Minimum connections for updating firmware: VCC, GND, DIN, DOUT, RTS & DTR
- Signal Direction is specified with respect to the module
- Module includes a 50k Ω pull-up resistor attached to **RESET**
- Several of the input pull-ups can be configured using the PR command
- Unused pins should be left disconnected

Electrical Characteristics

Table 1-03. DC Characteristics (VCC = 2.8 - 3.4 VDC)

Symbol	Characteristic	Condition	Min	Typical		Max	Unit
V_{IL}	Input Low Voltage	All Digital Inputs	-	-		0.35 * VCC	V
V_{IH}	Input High Voltage	All Digital Inputs	0.7 * VCC	-		-	V
V_{OL}	Output Low Voltage	I_{OL} = 2 mA, VCC >= 2.7 V	-	-		0.5	V
V_{OH}	Output High Voltage	I_{OH} = -2 mA, VCC >= 2.7 V	VCC - 0.5	-		-	V
II_{IN}	Input Leakage Current	V_{IN} = VCC or GND, all inputs, per pin	-	0.025		1	µA
II_{OZ}	High Impedance Leakage Current	V_{IN} = VCC or GND, all I/O High-Z, per pin	-	0.025		1	µA
TX	Transmit Current	VCC = 3.3 V	-	45 (XBee)	215, 140 (PRO, Int)	-	mA
RX	Receive Current	VCC = 3.3 V	-	50 (XBee)	55 (PRO)	-	mA
PWR-DWN	Power-down Current	SM parameter = 1	-	< 10		-	µA

Table 1-04. ADC Characteristics (Operating)

Symbol	Characteristic	Condition	Min	Typical	Max	Unit
V_{REFH}	VREF - Analog-to-Digital converter reference range		2.08	-	V_{DDAD*}	V
I_{REF}	VREF - Reference Supply Current	Enabled	-	200	-	µA
		Disabled or Sleep Mode	-	< 0.01	0.02	µA
V_{INDC}	Analog Input Voltage[1]		V_{SSAD} - 0.3	-	V_{DDAD} + 0.3	V

1. Maximum electrical operating range, not valid conversion range.

* V_{DDAD} is connected to VCC.

Table 1-05. ADC Timing/Performance Characteristics[1]

Symbol	Characteristic	Condition	Min	Typical	Max	Unit
R_{AS}	Source Impedance at Input[2]		-	-	10	kΩ
V_{AIN}	Analog Input Voltage[3]		V_{REFL}		V_{REFH}	V
RES	Ideal Resolution (1 LSB)[4]	2.08V ≤ V_{DDAD} ≤ 3.6V	2.031	-	3.516	mV
DNL	Differential Non-linearity[5]		-	±0.5	±1.0	LSB
INL	Integral Non-linearity[6]		-	±0.5	±1.0	LSB
E_{ZS}	Zero-scale Error[7]		-	±0.4	±1.0	LSB
F_{FS}	Full-scale Error[8]		-	±0.4	±1.0	LSB
E_{IL}	Input Leakage Error[9]		-	±0.05	±5.0	LSB
E_{TU}	Total Unadjusted Error[10]		-	±1.1	±2.5	LSB

1. All ACCURACY numbers are based on processor and system being in WAIT state (very little activity and no IO switching) and that adequate low-pass filtering is present on analog input pins (filter with 0.01 µF to 0.1 µF capacitor between analog input and VREFL). Failure to observe these guidelines may result in system or microcontroller noise causing accuracy errors which will vary based on board layout and the type and magnitude of the activity.
Data transmission and reception during data conversion may cause some degradation of these specifications, depending on the number and timing of packets. It is advisable to test the ADCs in your installation if best accuracy is required.
2. R_{AS} is the real portion of the impedance of the network driving the analog input pin. Values greater than this amount may not fully charge the input circuitry of the ATD resulting in accuracy error.
3. Analog input must be between V_{REFL} and V_{REFH} for valid conversion. Values greater than V_{REFH} will convert to $3FF.
4. The resolution is the ideal step size or 1LSB = $(V_{REFH}–V_{REFL})/1024$
5. Differential non-linearity is the difference between the current code width and the ideal code width (1LSB). The current code width is the difference in the transition voltages to and from the current code.
6. Integral non-linearity is the difference between the transition voltage to the current code and the adjusted ideal transition voltage for the current code. The adjusted ideal transition voltage is $(Current\ Code–1/2)*(1/((V_{REFH}+E_{FS})–(V_{REFL}+E_{ZS})))$.
7. Zero-scale error is the difference between the transition to the first valid code and the ideal transition to that code. The Ideal transition voltage to a given code is $(Code–1/2)*(1/(V_{REFH}–V_{REFL}))$.
8. Full-scale error is the difference between the transition to the last valid code and the ideal transition to that code. The ideal transition voltage to a given code is $(Code–1/2)*(1/(V_{REFH}–V_{REFL}))$.
9. Input leakage error is error due to input leakage across the real portion of the impedance of the network driving the analog pin. Reducing the impedance of the network reduces this error.

10. Total unadjusted error is the difference between the transition voltage to the current code and the ideal straight-line transfer function. This measure of error includes inherent quantization error (1/2LSB) and circuit error (differential, integral, zero-scale, and full-scale) error. The specified value of E_{TU} assumes zero E_{IL} (no leakage or zero real source impedance).

2. RF Module Operation

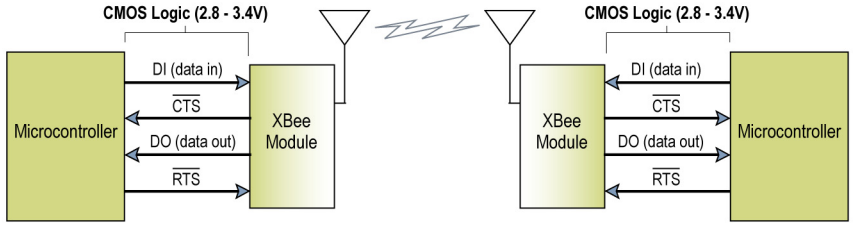
Serial Communications

The XBee®/XBee-PRO® RF Modules interface to a host device through a logic-level asynchronous serial port. Through its serial port, the module can communicate with any logic and voltage compatible UART; or through a level translator to any serial device (For example: Through a Digi proprietary RS-232 or USB interface board).

UART Data Flow

Devices that have a UART interface can connect directly to the pins of the RF module as shown in the figure below.

Figure 2-01. System Data Flow Diagram in a UART-interfaced environment
(Low-asserted signals distinguished with horizontal line over signal name.)

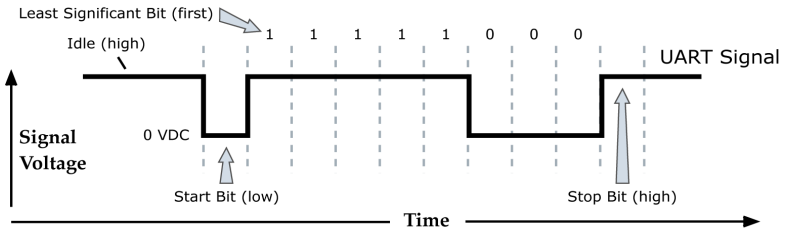


Serial Data

Data enters the module UART through the DI pin (pin 3) as an asynchronous serial signal. The signal should idle high when no data is being transmitted.

Each data byte consists of a start bit (low), 8 data bits (least significant bit first) and a stop bit (high). The following figure illustrates the serial bit pattern of data passing through the module.

Figure 2-02. UART data packet 0x1F (decimal number "31") as transmitted through the RF module
Example Data Format is 8-N-1 (bits - parity - # of stop bits)



Serial communications depend on the two UARTs (the microcontroller's and the RF module's) to be configured with compatible settings (baud rate, parity, start bits, stop bits, data bits).

The UART baud rate and parity settings on the XBee module can be configured with the BD and SB commands, respectively. See the command table in Chapter 3 for details.

Transparent Operation

By default, XBee®/XBee-PRO® RF Modules operate in Transparent Mode. When operating in this mode, the modules act as a serial line replacement - all UART data received through the DI pin is queued up for RF transmission. When RF data is received, the data is sent out the DO pin.

Serial-to-RF Packetization

Data is buffered in the DI buffer until one of the following causes the data to be packetized and transmitted:

1. No serial characters are received for the amount of time determined by the RO (Packetization Timeout) parameter. If RO = 0, packetization begins when a character is received.
2. The maximum number of characters that will fit in an RF packet (100) is received.
3. The Command Mode Sequence (GT + CC + GT) is received. Any character buffered in the DI buffer before the sequence is transmitted.

If the module cannot immediately transmit (for instance, if it is already receiving RF data), the serial data is stored in the DI Buffer. The data is packetized and sent at any RO timeout or when 100 bytes (maximum packet size) are received.

If the DI buffer becomes full, hardware or software flow control must be implemented in order to prevent overflow (loss of data between the host and module).

API Operation

API (Application Programming Interface) Operation is an alternative to the default Transparent Operation. The frame-based API extends the level to which a host application can interact with the networking capabilities of the module.

When in API mode, all data entering and leaving the module is contained in frames that define operations or events within the module.

Transmit Data Frames (received through the DI pin (pin 3)) include:

- RF Transmit Data Frame
- Command Frame (equivalent to AT commands)

Receive Data Frames (sent out the DO pin (pin 2)) include:

- RF-received data frame
- Command response
- Event notifications such as reset, associate, disassociate, etc.

The API provides alternative means of configuring modules and routing data at the host application layer. A host application can send data frames to the module that contain address and payload information instead of using command mode to modify addresses. The module will send data frames to the application containing status packets; as well as source, RSSI and payload information from received data packets.

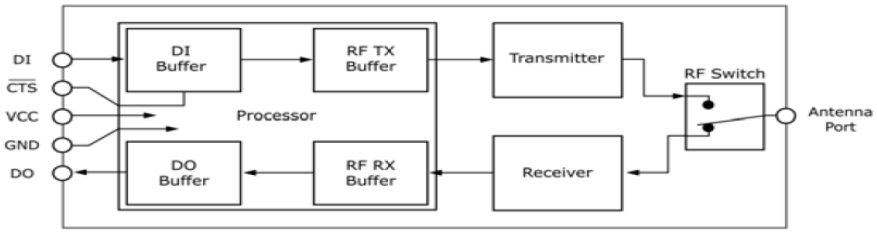
The API operation option facilitates many operations such as the examples cited below:

-> Transmitting data to multiple destinations without entering Command Mode
-> Receive success/failure status of each transmitted RF packet
-> Identify the source address of each received packet

To implement API operations, refer to API sections [p57].

Flow Control

Figure 2-03. Internal Data Flow Diagram



DI (Data In) Buffer

When serial data enters the RF module through the DI pin (pin 3), the data is stored in the DI Buffer until it can be processed.

Hardware Flow Control ($\overline{\text{CTS}}$). When the DI buffer is 17 bytes away from being full; by default, the module de-asserts $\overline{\text{CTS}}$ (high) to signal to the host device to stop sending data [refer to D7 (DIO7 Configuration) parameter]. $\overline{\text{CTS}}$ is re-asserted after the DI Buffer has 34 bytes of memory available.

How to eliminate the need for flow control:

1. Send messages that are smaller than the DI buffer size (202 bytes).
2. Interface at a lower baud rate [BD (Interface Data Rate) parameter] than the throughput data rate.

Case in which the DI Buffer may become full and possibly overflow:

If the module is receiving a continuous stream of RF data, any serial data that arrives on the DI pin is placed in the DI Buffer. The data in the DI buffer will be transmitted over-the-air when the module is no longer receiving RF data in the network.

Refer to the RO (Packetization Timeout), BD (Interface Data Rate) and D7 (DIO7 Configuration) command descriptions for more information.

DO (Data Out) Buffer

When RF data is received, the data enters the DO buffer and is sent out the serial port to a host device. Once the DO Buffer reaches capacity, any additional incoming RF data is lost.

Hardware Flow Control ($\overline{\text{RTS}}$). If $\overline{\text{RTS}}$ is enabled for flow control (D6 (DIO6 Configuration) Parameter = 1), data will not be sent out the DO Buffer as long as $\overline{\text{RTS}}$ (pin 16) is de-asserted.

Two cases in which the DO Buffer may become full and possibly overflow:

1. If the RF data rate is set higher than the interface data rate of the module, the module will receive data from the transmitting module faster than it can send the data to the host.
2. If the host does not allow the module to transmit data out from the DO buffer because of being held off by hardware or software flow control.

Refer to the D6 (DIO6 Configuration) command description for more information.

ADC and Digital I/O Line Support

The XBee®/XBee-PRO® RF Modules support ADC (Analog-to-digital conversion) and digital I/O line passing. The following pins support multiple functions:

Table 2-01. Pin functions and their associated pin numbers and commands
AD = Analog-to-Digital Converter, DIO = Digital Input/Output
Pin functions not applicable to this section are denoted within (parenthesis).

Pin Function	Pin#	AT Command
AD0 / DIO0	20	D0
AD1 / DIO1	19	D1
AD2 / DIO2	18	D2
AD3 / DIO3 / (COORD_SEL)	17	D3
AD4 / DIO4	11	D4
AD5 / DIO5 / (ASSOCIATE)	15	D5
DIO6 / (RTS)	16	D6
DIO7 / (CTS)	12	D7
DI8 / (DTR) / (Sleep_RQ)	9	D8

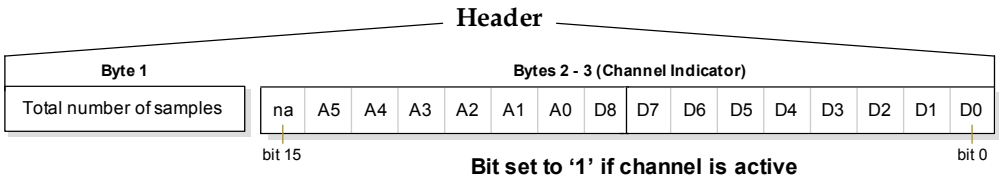
To enable ADC and DIO pin functions:

For ADC Support:	Set ATDn = 2
For Digital Input support:	Set ATDn = 3
For Digital Output Low support:	Set ATDn = 4
For Digital Output High support:	Set ATDn = 5

I/O Data Format

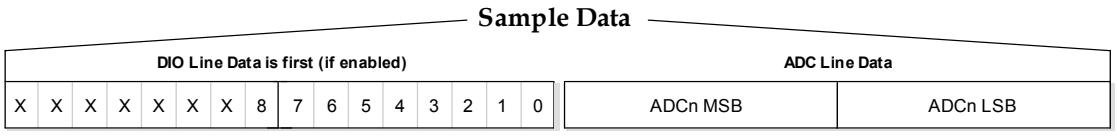
I/O data begins with a header. The first byte of the header defines the number of samples forth-coming. The last 2 bytes of the header (Channel Indicator) define which inputs are active. Each bit represents either a DIO line or ADC channel.

Figure 2-04. Header



Sample data follows the header and the channel indicator frame is used to determine how to read the sample data. If any of the DIO lines are enabled, the first 2 bytes are the DIO sample. The ADC data follows. ADC channel data is represented as an unsigned 10-bit value right-justified on a 16- bit boundary.

Figure 2-05. Sample Data



API Support

I/O data is sent out the UART using an API frame. All other data can be sent and received using Transparent Operation [refer to p11] or API framing if API mode is enabled (AP > 0).

API Operations support two RX (Receive) frame identifiers for I/O data (set 16-bit address to 0xFFFE and the module will do 64-bit addressing):

- 0x82 for RX (Receive) Packet: 64-bit address I/O
- 0x83 for RX (Receive) Packet: 16-bit address I/O

The API command header is the same as shown in the "RX (Receive) Packet: 64-bit Address" and "RX (Receive) Packet: 64-bit Address" API types [refer to p63]. RX data follows the format described in the I/O Data Format section [p13].

Applicable Commands: AP (API Enable)

Sleep Support

Automatic wakeup sampling can be suppressed by setting SO bit 1.When an RF module wakes, it will always do a sample based on any active ADC or DIO lines. This allows sampling based on the sleep cycle whether it be Cyclic Sleep (SM parameter = 4 or 5) or Pin Sleep (SM = 1 or 2). To gather more samples when awake, set the IR (Sample Rate) parameter.

For Cyclic Sleep modes: If the IR parameter is set, the module will stay awake until the IT (Samples before TX) parameter is met. The module will stay awake for ST (Time before Sleep) time.

Applicable Commands: IR (Sample Rate), IT (Samples before TX), SM (Sleep Mode), IC (DIO Change Detect), SO (Sleep Options)

DIO Pin Change Detect

When "DIO Change Detect" is enabled (using the IC command), DIO lines 0-7 are monitored. When a change is detected on a DIO line, the following will occur:

1. An RF packet is sent with the updated DIO pin levels. This packet will not contain any ADC samples.
2. Any queued samples are transmitted before the change detect data. This may result in receiving a packet with less than IT (Samples before TX) samples.

Note: Change detect will not affect Pin Sleep wake-up. The D8 pin (DTR/Sleep_RQ/DI8) is the only line that will wake a module from Pin Sleep. If not all samples are collected, the module will still enter Sleep Mode after a change detect packet is sent.

Applicable Commands: IC (DIO Change Detect), IT (Samples before TX)

NOTE: Change detect is only supported when the Dx (DIOx Configuration) parameter equals 3,4 or 5.

Sample Rate (Interval)

The Sample Rate (Interval) feature allows enabled ADC and DIO pins to be read periodically on modules that are not configured to operate in Sleep Mode. When one of the Sleep Modes is enabled and the IR (Sample Rate) parameter is set, the module will stay awake until IT (Samples before TX) samples have been collected.

Once a particular pin is enabled, the appropriate sample rate must be chosen. The maximum sample rate that can be achieved while using one A/D line is 1 sample/ms or 1 KHz (Note that the modem will not be able to keep up with transmission when IR & IT are equal to "1" and that configuring the modem to sample at rates greater than once every 20ms is not recommended).

Applicable Commands: IR (Sample Rate), IT (Samples before TX), SM (Sleep Mode)

I/O Line Passing

Virtual wires can be set up between XBee®/XBee-PRO® Modules. When an RF data packet is received that contains I/O data, the receiving module can be setup to update any enabled outputs (PWM and DIO) based on the data it receives.

Note that I/O lines are mapped in pairs. For example: AD0 can only update PWM0 and DI5 can only update DO5. The default setup is for outputs not to be updated, which results in the I/O data being sent out the UART (refer to the IU (Enable I/O Output) command). To enable the outputs to be updated, the IA (I/O Input Address) parameter must be setup with the address of the module that has the appropriate inputs enabled. This effectively binds the outputs to a particular module's input. This does not affect the ability of the module to receive I/O line data from other modules - only its ability to update enabled outputs. The IA parameter can also be setup to accept I/O data for output changes from any module by setting the IA parameter to 0xFFFF.

When outputs are changed from their non-active state, the module can be setup to return the output level to it non-active state. The timers are set using the Tn (Dn Output Timer) and PT (PWM Output Timeout) commands. The timers are reset every time a valid I/O packet (passed IA check) is received. The IC (Change Detect) and IR (Sample Rate) parameters can be setup to keep the output set to their active output if the system needs more time than the timers can handle.

Note: DI8 cannot be used for I/O line passing.

Applicable Commands: IA (I/O Input Address), Tn (Dn Output Timeout), P0 (PWM0 Configuration), P1 (PWM1 Configuration), M0 (PWM0 Output Level), M1 (PWM1 Output Level), PT (PWM Output Timeout), RP (RSSSI PWM Timer)

Configuration Example

As an example for a simple A/D link, a pair of RF modules could be set as follows:

Remote Configuration	Base Configuration
DL = 0x1234	DL = 0x5678
MY = 0x5678	MY = 0x1234
D0 = 2	P0 = 2
D1 = 2	P1 = 2
IR = 0x14	IU = 1
IT = 5	IA = 0x5678 (or 0xFFFF)

These settings configure the remote module to sample AD0 and AD1 once each every 20 ms. It then buffers 5 samples each before sending them back to the base module. The base should then receive a 32-Byte transmission (20 Bytes data and 12 Bytes framing) every 100 ms.

XBee®/XBee-PRO® Networks

The following terms will be used to explicate the network operations:

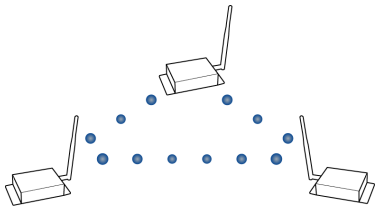
Table 2-02. Terms and definitions

Term	Definition
PAN	Personal Area Network - A data communication network that includes one or more End Devices and optionally a Coordinator.
Coordinator	A Full-function device (FFD) that provides network synchronization by polling nodes [NonBeacon (w/ Coordinator) networks only]
End Device	*When in the same network as a Coordinator* - RF modules that rely on a Coordinator for synchronization and can be put into states of sleep for low-power applications.
Association	The establishment of membership between End Devices and a Coordinator. Association is only applicable in NonBeacon (w/Coordinator) networks.

Peer-to-Peer

By default, XBee®/XBee-PRO RF Modules are configured to operate within a Peer-to-Peer network topology and therefore are not dependent upon Master/Slave relationships. NonBeacon systems operate within a Peer-to-Peer network topology and therefore are not dependent upon Master/Slave relationships. This means that modules remain synchronized without use of master/server configurations and each module in the network shares both roles of master and slave. Digi's peer-to-peer architecture features fast synchronization times and fast cold start times. This default configuration accommodates a wide range of RF data applications.

Figure 2-06. Peer-to-Peer Architecture



A peer-to-peer network can be established by configuring each module to operate as an End Device (CE = 0), disabling End Device Association on all modules (A1 = 0) and setting ID and CH parameters to be identical across the network.

NonBeacon (w/ Coordinator)

A device is configured as a Coordinator by setting the CE (Coordinator Enable) parameter to "1". Coordinator power-up is governed by the A2 (Coordinator Association) parameter.

In a Coordinator system, the Coordinator can be configured to use direct or indirect transmissions. If the SP (Cyclic Sleep Period) parameter is set to "0", the Coordinator will send data immediately. Otherwise, the SP parameter determines the length of time the Coordinator will retain the data before discarding it. Generally, SP (Cyclic Sleep Period) and ST (Time before Sleep) parameters should be set to match the SP and ST settings of the End Devices.

Association

Association is the establishment of membership between End Devices and a Coordinator. The establishment of membership is useful in scenarios that require a central unit (Coordinator) to relay messages to or gather data from several remote units (End Devices), assign channels or assign PAN IDs.

An RF data network that consists of one Coordinator and one or more End Devices forms a PAN (Personal Area Network). Each device in a PAN has a PAN Identifier [ID (PAN ID) parameter]. PAN IDs must be unique to prevent miscommunication between PANs. The Coordinator PAN ID is set using the ID (PAN ID) and A2 (Coordinator Association) commands.

An End Device can associate to a Coordinator without knowing the address, PAN ID or channel of the Coordinator. The A1 (End Device Association) parameter bit fields determine the flexibility of an End Device during association. The A1 parameter can be used for an End Device to dynamically set its destination address, PAN ID and/or channel.

For example: If the PAN ID of a Coordinator is known, but the operating channel is not; the A1 command on the End Device should be set to enable the 'Auto_Associate' and 'Reassign_Channel' bits. Additionally, the ID parameter should be set to match the PAN ID of the associated Coordinator.

Coordinator / End Device Setup and Operation

To configure a module to operate as a Coordinator, set the CE (Coordinator Enable) parameter to '1'. Set the CE parameter of End Devices to '0' (default). Coordinator and End Devices should contain matching firmware versions.

NonBeacon (w/ Coordinator) Systems

The Coordinator can be configured to use direct or indirect transmissions. If the SP (Cyclic Sleep Period) parameter is set to '0', the Coordinator will send data immediately. Otherwise, the SP parameter determines the length of time the Coordinator will retain the data before discarding it. Generally, SP (Cyclic Sleep Period) and ST (Time before Sleep) parameters should be set to match the SP and ST settings of the End Devices.

Coordinator Start-up

Coordinator power-up is governed by the A2 (Coordinator Association) command. On power-up, the Coordinator undergoes the following sequence of events:

1. **Check A2 parameter- Reassign_PANID Flag**

   Set (bit 0 = 1) - The Coordinator issues an Active Scan. The Active Scan selects one channel and transmits a request to the broadcast address (0xFFFF) and broadcast PAN ID (0xFFFF). It then listens on that channel for beacons from any Coordinator operating on that channel. The listen time on each channel is determined by the SD (Scan Duration) parameter value.

   Once the time expires on that channel, the Active Scan selects another channel and again transmits the BeaconRequest as before. This process continues until all channels have been scanned, or until 5 PANs have been discovered. When the Active Scan is complete, the results include a list of PAN IDs and Channels that are being used by other PANs. This list is used to assign an unique PAN ID to the new Coordinator. The ID parameter will be retained if it is not found in the Active Scan results. Otherwise, the ID (PAN ID) parameter setting will be updated to a PAN ID that was not detected.

   Not Set (bit 0 = 0) - The Coordinator retains its ID setting. No Active Scan is performed.

2. Check A2 parameter - Reassign_Channel Flag (bit 1)

Set (bit 1 = 1) - The Coordinator issues an Energy Scan. The Energy Scan selects one channel and scans for energy on that channel. The duration of the scan is specified by the SD (Scan Duration) parameter. Once the scan is completed on a channel, the Energy Scan selects the next channel and begins a new scan on that channel. This process continues until all channels have been scanned.

When the Energy Scan is complete, the results include the maximal energy values detected on each channel. This list is used to determine a channel where the least energy was detected. If an Active Scan was performed (Reassign_PANID Flag set), the channels used by the detected PANs are eliminated as possible channels. Thus, the results of the Energy Scan and the Active Scan (if performed) are used to find the best channel (channel with the least energy that is not used by any detected PAN). Once the best channel has been selected, the CH (Channel) parameter value is updated to that channel.

Not Set (bit 1 = 0) - The Coordinator retains its CH setting. An Energy Scan is not performed.

3. Start Coordinator

The Coordinator starts on the specified channel (CH parameter) and PAN ID (ID parameter). Note, these may be selected in steps 1 and/or 2 above. The Coordinator will only allow End Devices to associate to it if the A2 parameter "AllowAssociation" flag is set. Once the Coordinator has successfully started, the Associate LED will blink 1 time per second. (The LED is solid if the Coordinator has not started.)

4. Coordinator Modifications

Once a Coordinator has started:
Modifying the A2 (Reassign_Channel or Reassign_PANID bits), ID, CH or MY parameters will cause the Coordinator's MAC to reset (The Coordinator RF module (including volatile RAM) is not reset). Changing the A2 AllowAssociation bit will not reset the Coordinator's MAC. In a non-beaconing system, End Devices that associated to the Coordinator prior to a MAC reset will have knowledge of the new settings on the Coordinator. Thus, if the Coordinator were to change its ID, CH or MY settings, the End Devices would no longer be able to communicate with the non-beacon Coordinator. Once a Coordinator has started, the ID, CH, MY or A2 (Reassign_Channel or Reassign_PANID bits) should not be changed.

End Device Start-up

End Device power-up is governed by the A1 (End Device Association) command. On power-up, the End Device undergoes the following sequence of events:

1. Check A1 parameter - AutoAssociate Bit

Set (bit 2 = 1) - End Device will attempt to associate to a Coordinator. (refer to steps 2-3).

Not Set (bit 2 = 0) - End Device will not attempt to associate to a Coordinator. The End Device will operate as specified by its ID, CH and MY parameters. Association is considered complete and the Associate LED will blink quickly (5 times per second). When the AutoAssociate bit is not set, the remaining steps (2-3) do not apply.

2. Discover Coordinator (if Auto-Associate Bit Set)

The End Device issues an Active Scan. The Active Scan selects one channel and transmits a BeaconRequest command to the broadcast address (0xFFFF) and broadcast PAN ID (0xFFFF). It then listens on that channel for beacons from any Coordinator operating on that channel. The listen time on each channel is determined by the SD parameter.

Once the time expires on that channel, the Active Scan selects another channel and again transmits the BeaconRequest command as before. This process continues until all channels have been scanned, or until 5 PANs have been discovered. When the Active Scan is complete, the results include a list of PAN IDs and Channels that are being used by detected PANs.

The End Device selects a Coordinator to associate with according to the A1 parameter "Reassign_PANID" and "Reassign_Channel" flags:

Reassign_PANID Bit Set (bit 0 = 1)- End Device can associate with a PAN with any ID value.

Reassign_PANID Bit Not Set (bit 0 = 0) - End Device will only associate with a PAN whose ID setting matches the ID setting of the End Device.

Reassign_Channel Bit Set (bit 1 = 1) - End Device can associate with a PAN with any CH value.

Reassign_Channel Bit Not Set (bit 1 = 0)- End Device will only associate with a PAN whose CH setting matches the CH setting of the End Device.

After applying these filters to the discovered Coordinators, if multiple candidate PANs exist, the End Device will select the PAN whose transmission link quality is the strongest. If no valid Coordinator is found, the End Device will either go to sleep (as dictated by its SM (Sleep Mode) parameter) or retry Association.

Note - An End Device will also disqualify Coordinators if they are not allowing association (A2 - AllowAssociation bit); or, if the Coordinator is not using the same NonBeacon scheme as the End Device. (They must both be programmed with NonBeacon code.)

3. Associate to Valid Coordinator

Once a valid Coordinator is found (step 2), the End Device sends an AssociationRequest message to the Coordinator. It then waits for an AssociationConfirmation to be sent from the Coordinator. Once the Confirmation is received, the End Device is Associated and the Associate LED will blink rapidly (2 times per second). The LED is solid if the End Device has not associated.

4. End Device Changes once an End Device has associated

Changing A1, ID or CH parameters will cause the End Device to disassociate and restart the Association procedure.

If the End Device fails to associate, the AI command can give some indication of the failure.

XBee®/XBee-PRO® Addressing

Every RF data packet sent over-the-air contains a Source Address and Destination Address field in its header. The RF module conforms to the 802.15.4 specification and supports both short 16-bit addresses and long 64-bit addresses. A unique 64-bit IEEE source address is assigned at the factory and can be read with the SL (Serial Number Low) and SH (Serial Number High) commands. Short addressing must be configured manually. A module will use its unique 64-bit address as its Source Address if its MY (16-bit Source Address) value is "0xFFFF" or "0xFFFE".

To send a packet to a specific module using 64-bit addressing: Set the Destination Address (DL + DH) of the sender to match the Source Address (SL + SH) of the intended destination module.

To send a packet to a specific module using 16-bit addressing: Set DL (Destination Address Low) parameter to equal the MY parameter of the intended destination module and set the DH (Destination Address High) parameter to '0'.

Unicast Mode

By default, the RF module operates in Unicast Mode. Unicast Mode is the only mode that supports retries. While in this mode, receiving modules send an ACK (acknowledgement) of RF packet reception to the transmitter. If the transmitting module does not receive the ACK, it will re-send the packet up to three times or until the ACK is received.

Short 16-bit addresses. The module can be configured to use short 16-bit addresses as the Source Address by setting (MY < 0xFFFE). Setting the DH parameter (DH = 0) will configure the Destination Address to be a short 16-bit address (if DL < 0xFFFE). For two modules to communicate using short addressing, the Destination Address of the transmitter module must match the MY parameter of the receiver.

The following table shows a sample network configuration that would enable Unicast Mode communications using short 16-bit addresses.

Table 2-03. Sample Unicast Network Configuration (using 16-bit addressing)

Parameter	RF Module 1	RF Module 2
MY (Source Address)	0x01	0x02
DH (Destination Address High)	0	0
DL (Destination Address Low)	0x02	0x01

Long 64-bit addresses. The RF module's serial number (SL parameter concatenated to the SH parameter) can be used as a 64-bit source address when the MY (16-bit Source Address) parameter is disabled. When the MY parameter is disabled (MY = 0xFFFF or 0xFFFE), the module's source address is set to the 64-bit IEEE address stored in the SH and SL parameters.

When an End Device associates to a Coordinator, its MY parameter is set to 0xFFFE to enable 64-bit addressing. The 64-bit address of the module is stored as SH and SL parameters. To send a packet to a specific module, the Destination Address (DL + DH) on the sender must match the Source Address (SL + SH) of the desired receiver.

Broadcast Mode

Any RF module within range will accept a packet that contains a broadcast address. When configured to operate in Broadcast Mode, receiving modules do not send ACKs (Acknowledgements) and transmitting modules do not automatically re-send packets as is the case in Unicast Mode.

To send a broadcast packet to all modules regardless of 16-bit or 64-bit addressing, set the destination addresses of all the modules as shown below.

Sample Network Configuration (All modules in the network):

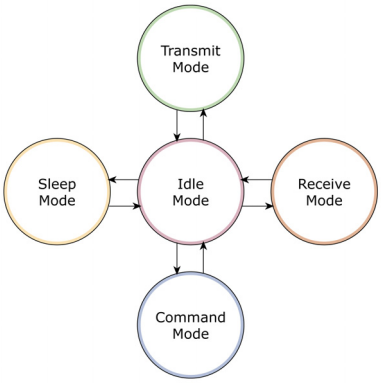
- DL (Destination Low Address) = 0x0000FFFF
- DH (Destination High Address) = 0x00000000 (default value)

NOTE: When programming the module, parameters are entered in hexadecimal notation (without the "0x" prefix). Leading zeros may be omitted.

Modes of Operation

XBee®/XBee-PRO® RF Modules operate in five modes.

Figure 2-07. Modes of Operation



Idle Mode

When not receiving or transmitting data, the RF module is in Idle Mode. The module shifts into the other modes of operation under the following conditions:

- Transmit Mode (Serial data is received in the DI Buffer)
- Receive Mode (Valid RF data is received through the antenna)
- Sleep Mode (Sleep Mode condition is met)
- Command Mode (Command Mode Sequence is issued)

Transmit/Receive Modes

RF Data Packets

Each transmitted data packet contains a Source Address and Destination Address field. The Source Address matches the address of the transmitting module as specified by the MY (Source Address) parameter (if MY >= 0xFFFE), the SH (Serial Number High) parameter or the SL (Serial Number Low) parameter. The <Destination Address> field is created from the DH (Destination Address High) and DL (Destination Address Low) parameter values. The Source Address and/or Destination Address fields will either contain a 16-bit short or long 64-bit long address.

The RF data packet structure follows the 802.15.4 specification.

[Refer to the XBee/XBee-PRO Addressing section for more information]

Direct and Indirect Transmission

There are two methods to transmit data:

- Direct Transmission - data is transmitted immediately to the Destination Address
- Indirect Transmission - A packet is retained for a period of time and is only transmitted after the destination module (Source Address = Destination Address) requests the data.

Indirect Transmissions can only occur on a Coordinator. Thus, if all nodes in a network are End Devices, only Direct Transmissions will occur. Indirect Transmissions are useful to ensure packet delivery to a sleeping node. The Coordinator currently is able to retain up to 2 indirect messages.

Direct Transmission

A Coordinator can be configured to use only Direct Transmission by setting the SP (Cyclic Sleep Period) parameter to "0". Also, a Coordinator using indirect transmissions will revert to direct transmission if it knows the destination module is awake.

To enable this behavior, the ST (Time before Sleep) value of the Coordinator must be set to match the ST value of the End Device. Once the End Device either transmits data to the Coordinator or polls the Coordinator for data, the Coordinator will use direct transmission for all subsequent data transmissions to that module address until ST time occurs with no activity (at which point it will revert to using indirect transmissions for that module address). "No activity" means no transmission or reception of messages with a specific address. Global messages will not reset the ST timer.

Indirect Transmission

To configure Indirect Transmissions in a PAN (Personal Area Network), the SP (Cyclic Sleep Period) parameter value on the Coordinator must be set to match the longest sleep value of any End Device. The sleep period value on the Coordinator determines how long (time or number of beacons) the Coordinator will retain an indirect message before discarding it.

An End Device must poll the Coordinator once it wakes from Sleep to determine if the Coordinator has an indirect message for it. For Cyclic Sleep Modes, this is done automatically every time the module wakes (after SP time). For Pin Sleep Modes, the A1 (End Device Association) parameter value must be set to enable Coordinator polling on pin wake-up. Alternatively, an End Device can use the FP (Force Poll) command to poll the Coordinator as needed.

CCA (Clear Channel Assessment)

Prior to transmitting a packet, a CCA (Clear Channel Assessment) is performed on the channel to determine if the channel is available for transmission. The detected energy on the channel is compared with the CA (Clear Channel Assessment) parameter value. If the detected energy exceeds the CA parameter value, the packet is not transmitted.

Also, a delay is inserted before a transmission takes place. This delay is settable using the RN (Backoff Exponent) parameter. If RN is set to "0", then there is no delay before the first CCA is performed. The RN parameter value is the equivalent of the "minBE" parameter in the 802.15.4 specification. The transmit sequence follows the 802.15.4 specification.

By default, the MM (MAC Mode) parameter = 0. On a CCA failure, the module will attempt to resend the packet up to two additional times.

When in Unicast packets with RR (Retries) = 0, the module will execute two CCA retries. Broadcast packets always get two CCA retries.

Acknowledgement

If the transmission is not a broadcast message, the module will expect to receive an acknowledgement from the destination node. If an acknowledgement is not received, the packet will be resent up to 3 more times. If the acknowledgement is not received after all transmissions, an ACK failure is recorded.

Sleep Mode

Sleep Modes enable the RF module to enter states of low-power consumption when not in use. In order to enter Sleep Mode, one of the following conditions must be met (in addition to the module having a non-zero SM parameter value):

- Sleep_RQ (pin 9) is asserted and the module is in a pin sleep mode (SM = 1, 2, or 5)
- The module is idle (no data transmission or reception) for the amount of time defined by the ST (Time before Sleep) parameter. [NOTE: ST is only active when SM = 4-5.]

Table 2-04. Sleep Mode Configurations

Sleep Mode Setting	Transition into Sleep Mode	Transition out of Sleep Mode (wake)	Characteristics	Related Commands	Power Consumption
Pin Hibernate (SM = 1)	Assert (high) Sleep_RQ (pin 9)	De-assert (low) Sleep_RQ	Pin/Host-controlled / NonBeacon systems only / Lowest Power	(SM)	< 10 µA (@3.0 VCC)
Pin Doze (SM = 2)	Assert (high) Sleep_RQ (pin 9)	De-assert (low) Sleep_RQ	Pin/Host-controlled / NonBeacon systems only / Fastest wake-up	(SM)	< 50 µA
Cyclic Sleep (SM = 4)	Automatic transition to Sleep Mode as defined by the SM (Sleep Mode) and ST (Time before Sleep) parameters.	Transition occurs after the cyclic sleep time interval elapses. The time interval is defined by the SP (Cyclic Sleep Period) parameter.	RF module wakes in pre-determined time intervals to detect if RF data is present / When SM = 5	(SM), SP, ST	< 50 µA when sleeping
Cyclic Sleep (SM = 5)	Automatic transition to Sleep Mode as defined by the SM (Sleep Mode) and ST (Time before Sleep) parameters or or on a falling edge transition of the SLEEP_RQ pin.	Transition occurs after the cyclic sleep time interval elapses. The time interval is defined by the SP (Cyclic Sleep Period) parameter.	RF module wakes in pre-determined time intervals to detect if RF data is present. Module also wakes on a falling edge of SLEEP_RQ	(SM), SP, ST	< 50 µA when sleeping

The SM command is central to setting Sleep Mode configurations. By default, Sleep Modes are disabled (SM = 0) and the module remains in Idle/Receive Mode. When in this state, the module is constantly ready to respond to serial or RF activity.

Pin/Host-controlled Sleep Modes

The transient current when waking from pin sleep (SM = 1 or 2) does not exceed the idle current of the module. The current ramps up exponentially to its idle current.

Pin Hibernate (SM = 1)

- Pin/Host-controlled
- Typical power-down current: < 10 µA (@3.0 VCC)
- Wake-up time: 13.2 msec

Pin Hibernate Mode minimizes quiescent power (power consumed when in a state of rest or inactivity). This mode is voltage level-activated; when Sleep_RQ (pin 9) is asserted, the module will finish any transmit, receive or association activities, enter Idle Mode, and then enter a state of sleep. The module will not respond to either serial or RF activity while in pin sleep.

To wake a sleeping module operating in Pin Hibernate Mode, de-assert Sleep_RQ (pin 9). The module will wake when Sleep_RQ is de-asserted and is ready to transmit or receive when the CTS line is low. When waking the module, the pin must be de-asserted at least two 'byte times' after CTS goes low. This assures that there is time for the data to enter the DI buffer.

Pin Doze (SM = 2)

- Pin/Host-controlled
- Typical power-down current: < 50 µA
- Wake-up time: 2 msec

Pin Doze Mode functions as does Pin Hibernate Mode; however, Pin Doze features faster wake-up time and higher power consumption.

To wake a sleeping module operating in Pin Doze Mode, de-assert Sleep_RQ (pin 9). The module will wake when Sleep_RQ is de-asserted and is ready to transmit or receive when the CTS line is

low. When waking the module, the pin must be de-asserted at least two 'byte times' after CTS goes low. This assures that there is time for the data to enter the DI buffer.

Cyclic Sleep Modes

Cyclic Sleep Remote (SM = 4)

- Typical Power-down Current: < 50 µA (when asleep)
- Wake-up time: 2 msec

The Cyclic Sleep Modes allow modules to periodically check for RF data. When the SM parameter is set to '4', the module is configured to sleep, then wakes once a cycle to check for data from a module configured as a Cyclic Sleep Coordinator (SM = 0, CE = 1). The Cyclic Sleep Remote sends a poll request to the coordinator at a specific interval set by the SP (Cyclic Sleep Period) parameter. The coordinator will transmit any queued data addressed to that specific remote upon receiving the poll request.

If no data is queued for the remote, the coordinator will not transmit and the remote will return to sleep for another cycle. If queued data is transmitted back to the remote, it will stay awake to allow for back and forth communication until the ST (Time before Sleep) timer expires.

Also note that $\overline{\text{CTS}}$ will go low each time the remote wakes, allowing for communication initiated by the remote host if desired.

Cyclic Sleep Remote with Pin Wake-up (SM = 5)

Use this mode to wake a sleeping remote module through either the RF interface or by the de-assertion of Sleep_RQ for event-driven communications. The cyclic sleep mode works as described above (Cyclic Sleep Remote) with the addition of a pin-controlled wake-up at the remote module. The Sleep_RQ pin is edge-triggered, not level-triggered. The module will wake when a low is detected then set $\overline{\text{CTS}}$ low as soon as it is ready to transmit or receive.

Any activity will reset the ST (Time before Sleep) timer so the module will go back to sleep only after there is no activity for the duration of the timer. Once the module wakes (pin-controlled), further pin activity is ignored. The module transitions back into sleep according to the ST time regardless of the state of the pin.

[Cyclic Sleep Coordinator (SM = 6)]

- Typical current = Receive current
- Always awake

NOTE: The SM=6 parameter value exists solely for backwards compatibility with firmware version 1.x60. If backwards compatibility with the older firmware version is not required, always use the CE (Coordinator Enable) command to configure a module as a Coordinator.

This mode configures a module to wake cyclic sleeping remotes through RF interfacing. The Coordinator will accept a message addressed to a specific remote 16 or 64-bit address and hold it in a buffer until the remote wakes and sends a poll request. Messages not sent directly (buffered and requested) are called "Indirect messages". The Coordinator only queues one indirect message at a time. The Coordinator will hold the indirect message for a period 2.5 times the sleeping period indicated by the SP (Cyclic Sleep Period) parameter. The Coordinator's SP parameter should be set to match the value used by the remotes.

Command Mode

To modify or read RF Module parameters, the module must first enter into Command Mode - a state in which incoming characters are interpreted as commands. Two Command Mode options are supported: AT Command Mode [refer to section below] and API Command Mode [p57].

AT Command Mode

To Enter AT Command Mode:

Send the 3-character command sequence "+++" and observe guard times before and after the command characters. [Refer to the "Default AT Command Mode Sequence" below.]

Default AT Command Mode Sequence (for transition to Command Mode):

- No characters sent for one second [GT (Guard Times) parameter = 0x3E8]
- Input three plus characters ("+++") within one second [CC (Command Sequence Character) parameter = 0x2B.]
- No characters sent for one second [GT (Guard Times) parameter = 0x3E8]

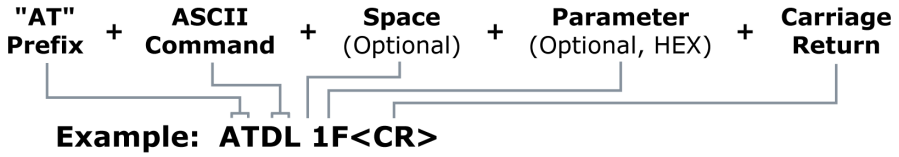
All of the parameter values in the sequence can be modified to reflect user preferences.

NOTE: Failure to enter AT Command Mode is most commonly due to baud rate mismatch. Ensure the 'Baud' setting on the "PC Settings" tab matches the interface data rate of the RF module. By default, the BD parameter = 3 (9600 bps).

To Send AT Commands:

Send AT commands and parameters using the syntax shown below.

Figure 2-08. Syntax for sending AT Commands



To read a parameter value stored in the RF module's register, omit the parameter field.

The preceding example would change the RF module Destination Address (Low) to "0x1F". To store the new value to non-volatile (long term) memory, subsequently send the WR (Write) command.

For modified parameter values to persist in the module's registry after a reset, changes must be saved to non-volatile memory using the WR (Write) Command. Otherwise, parameters are restored to previously saved values after the module is reset.

System Response. When a command is sent to the module, the module will parse and execute the command. Upon successful execution of a command, the module returns an "OK" message. If execution of a command results in an error, the module returns an "ERROR" message.

To Exit AT Command Mode:

1. Send the ATCN (Exit Command Mode) command (followed by a carriage return).
   [OR]
2. If no valid AT Commands are received within the time specified by CT (Command Mode Timeout) Command, the RF module automatically returns to Idle Mode.

For an example of programming the RF module using AT Commands and descriptions of each configurable parameter, refer to the RF Module Configuration chapter [p26].

3. RF Module Configuration

Programming the RF Module

Refer to the Command Mode section [p25] for more information about entering Command Mode, sending AT commands and exiting Command Mode. For information regarding module programming using API Mode, refer to the API Operation sections [p57].

Programming Examples

Setup

The programming examples in this section require the installation of Digi's X-CTU Software and a serial connection to a PC. (Digi stocks RS-232 and USB boards to facilitate interfacing with a PC.)

1. Install Digi's X-CTU Software to a PC by double-clicking the "setup_X-CTU.exe" file. (The file is located on the Digi CD and www.digi.com/xctu.)
2. Mount the RF module to an interface board, then connect the module assembly to a PC.
3. Launch the X-CTU Software and select the 'PC Settings' tab. Verify the baud and parity settings of the Com Port match those of the RF module.

NOTE: Failure to enter AT Command Mode is most commonly due to baud rate mismatch. Ensure the 'Baud' setting on the 'PC Settings' tab matches the interface data rate of the RF module. By default, the BD parameter = 3 (which corresponds to 9600 bps).

Sample Configuration: Modify RF Module Destination Address

Example: Utilize the X-CTU "Terminal" tab to change the RF module's DL (Destination Address Low) parameter and save the new address to non-volatile memory.

After establishing a serial connection between the RF module and a PC [refer to the 'Setup' section above], select the "Terminal" tab of the X-CTU Software and enter the following command lines ('CR' stands for carriage return):

Method 1 (One line per command)

Send AT Command	System Response
+++	OK <CR> (Enter into Command Mode)
ATDL <Enter>	{current value} <CR> (Read Destination Address Low)
ATDL1A0D <Enter>	OK <CR> (Modify Destination Address Low)
ATWR <Enter>	OK <CR> (Write to non-volatile memory)
ATCN <Enter>	OK <CR> (Exit Command Mode)

Method 2 (Multiple commands on one line)

Send AT Command	System Response
+++	OK <CR> (Enter into Command Mode)
ATDL <Enter>	{current value} <CR> (Read Destination Address Low)
ATDL1A0D,WR,CN <Enter>	OK<CR> OK<CR> OK<CR>

Sample Configuration: Restore RF Module Defaults

Example: Utilize the X-CTU "Modem Configuration" tab to restore default parameter values.

After establishing a connection between the module and a PC [refer to the 'Setup' section above], select the "Modem Configuration" tab of the X-CTU Software.

1. Select the 'Read' button.
2. Select the 'Restore' button.

Remote Configuration Commands

The API firmware has provisions to send configuration commands to remote devices using the Remote Command Request API frame (see API Operation). This API frame can be used to send commands to a remote module to read or set command parameters.

The API firmware has provisions to send configuration commands (set or read) to a remote module using the Remote Command Request API frame (see API Operations). Remote commands can be issued to read or set command parameters on a remote device.

Sending a Remote Command

To send a remote command, the Remote Command Request frame should be populated with values for the 64 bit and 16 bit addresses. If 64 bit addressing is desired then the 16 bit address field should be filled with 0xFFFE. If any value other than 0xFFFE is used in the 16 bit address field then the 64 bit address field will be ignored and 16 bit addressing will be used. If a command response is desired, the Frame ID should be set to a non-zero value.

Applying Changes on Remote

When remote commands are used to change command parameter settings on a remote device, parameter changes do not take effect until the changes are applied. For example, changing the BD parameter will not change the actual serial interface rate on the remote until the changes are applied. Changes can be applied using remote commands in one of three ways:

Set the apply changes option bit in the API frame

Issue an AC command to the remote device

Issue a WR + FR command to the remote device to save changes and reset the device.

Remote Command Responses

If the remote device receives a remote command request transmission, and the API frame ID is non-zero, the remote will send a remote command response transmission back to the device that sent the remote command. When a remote command response transmission is received, a device sends a remote command response API frame out its UART. The remote command response indicates the status of the command (success, or reason for failure), and in the case of a command query, it will include the register value.

The device that sends a remote command will not receive a remote command response frame if:

The destination device could not be reached

The frame ID in the remote command request is set to 0.

Command Reference Tables

XBee®/XBee-PRO® RF Modules expect numerical values in hexadecimal. Hexadecimal values are designated by a "0x" prefix. Decimal equivalents are designated by a "d" suffix. Commands are contained within the following command categories (listed in the order that their tables appear):

- Special
- Networking & Security
- RF Interfacing
- Sleep (Low Power)
- Serial Interfacing
- I/O Settings
- Diagnostics
- AT Command Options

All modules within a PAN should operate using the same firmware version.

Special

Table 3-01. XBee-PRO Commands - Special

AT Command	Command Category	Name and Description	Parameter Range	Default
WR	Special	**Write**. Write parameter values to non-volatile memory so that parameter modifications persist through subsequent power-up or reset. Note: Once WR is issued, no additional characters should be sent to the module until after the response "OK\r" is received.	-	-
RE	Special	**Restore Defaults**. Restore module parameters to factory defaults.	-	-
FR (v1.x80*)	Special	**Software Reset**. Responds immediately with an OK then performs a hard reset ~100ms later.	-	-

* Firmware version in which the command was first introduced (firmware versions are numbered in hexadecimal notation.)

Networking & Security

Table 3-02. XBee®/XBee-PRO® Commands - Networking & Security (Sub-categories designated within {brackets})

AT Command	Command Category	Name and Description	Parameter Range	Default
CH	Networking {Addressing}	**Channel**. Set/Read the channel number used for transmitting and receiving data between RF modules (uses 802.15.4 protocol channel numbers).	0x0B - 0x1A (XBee) 0x0C - 0x17 (XBee-PRO)	0x0C (12d)
ID	Networking {Addressing}	**PAN ID**. Set/Read the PAN (Personal Area Network) ID. Use 0xFFFF to broadcast messages to all PANs.	0 - 0xFFFF	0x3332 (13106d)
DH	Networking {Addressing}	**Destination Address High**. Set/Read the upper 32 bits of the 64-bit destination address. When combined with DL, it defines the destination address used for transmission. To transmit using a 16-bit address, set DH parameter to zero and DL less than 0xFFFF. 0x000000000000FFFF is the broadcast address for the PAN.	0 - 0xFFFFFFFF	0
DL	Networking {Addressing}	**Destination Address Low**. Set/Read the lower 32 bits of the 64-bit destination address. When combined with DH, DL defines the destination address used for transmission. To transmit using a 16-bit address, set DH parameter to zero and DL less than 0xFFFF. 0x000000000000FFFF is the broadcast address for the PAN.	0 - 0xFFFFFFFF	0
MY	Networking {Addressing}	**16-bit Source Address.** Set/Read the RF module 16-bit source address. Set MY = 0xFFFF to disable reception of packets with 16-bit addresses. 64-bit source address (serial number) and broadcast address (0x000000000000FFFF) is always enabled.	0 - 0xFFFF	0
SH	Networking {Addressing}	**Serial Number High**. Read high 32 bits of the RF module's unique IEEE 64-bit address. 64-bit source address is always enabled.	0 - 0xFFFFFFFF [read-only]	Factory-set
SL	Networking {Addressing}	**Serial Number Low**. Read low 32 bits of the RF module's unique IEEE 64-bit address. 64-bit source address is always enabled.	0 - 0xFFFFFFFF [read-only]	Factory-set
RR (v1.xA0*)	Networking {Addressing}	**XBee Retries**. Set/Read the maximum number of retries the module will execute in addition to the 3 retries provided by the 802.15.4 MAC. For each XBee retry, the 802.15.4 MAC can execute up to 3 retries.	0 - 6	0
RN	Networking {Addressing}	**Random Delay Slots.** Set/Read the minimum value of the back-off exponent in the CSMA-CA algorithm that is used for collision avoidance. If RN = 0, collision avoidance is disabled during the first iteration of the algorithm (802.15.4 - macMinBE).	0 - 3 [exponent]	0
MM (v1.x80*)	Networking {Addressing}	**MAC Mode.** MAC Mode. Set/Read MAC Mode value. MAC Mode enables/disables the use of a Digi header in the 802.15.4 RF packet. When Modes 0 or 3 are enabled (MM=0,3), duplicate packet detection is enabled as well as certain AT commands. Please see the detailed MM description on page 47 for additional information.	0 - 3 0 = Digi Mode 1 = 802.15.4 (no ACKs) 2 = 802.15.4 (with ACKs) 3 = Digi Mode (no ACKs)	0
NI (v1.x80*)	Networking {Identification}	**Node Identifier.** Stores a string identifier. The register only accepts printable ASCII data. A string can not start with a space. Carriage return ends command. Command will automatically end when maximum bytes for the string have been entered. This string is returned as part of the ND (Node Discover) command. This identifier is also used with the DN (Destination Node) command.	20-character ASCII string	-
ND (v1.x80*)	Networking {Identification}	**Node Discover.** Discovers and reports all RF modules found. The following information is reported for each module discovered (the example cites use of Transparent operation (AT command format) - refer to the long ND command description regarding differences between Transparent and API operation). MY<CR> SH<CR> SL<CR> DB<CR> NI<CR><CR> The amount of time the module allows for responses is determined by the NT parameter. In Transparent operation, command completion is designated by a <CR> (carriage return). ND also accepts a Node Identifier as a parameter. In this case, only a module matching the supplied identifier will respond. If ND self-response is enabled (NO=1) the module initiating the node discover will also output a response for itself.	optional 20-character NI value	
NT (v1.xA0*)	Networking {Identification}	**Node Discover Time.** Set/Read the amount of time a node will wait for responses from other nodes when using the ND (Node Discover) command.	0x01 - 0xFC [x 100 ms]	0x19

Table 3-02. XBee®/XBee-PRO® Commands - Networking & Security (Sub-categories designated within {brackets})

AT Command	Command Category	Name and Description	Parameter Range	Default
NO (v1xC5)	Networking {Identification}	**Node Discover Options**. Enables node discover self-response on the module.	0-1	0
DN (v1.x80*)	Networking {Identification}	**Destination Node.** Resolves an NI (Node Identifier) string to a physical address. The following events occur upon successful command execution: 1. DL and DH are set to the address of the module with the matching Node Identifier. 2. "OK" is returned. 3. RF module automatically exits AT Command Mode If there is no response from a module within 200 msec or a parameter is not specified (left blank), the command is terminated and an "ERROR" message is returned.	20-character ASCII string	-
CE (v1.x80*)	Networking {Association}	**Coordinator Enable**. Set/Read the coordinator setting.	0 - 1 0 = End Device 1 = Coordinator	0
SC (v1.x80*)	Networking {Association}	**Scan Channels**. Set/Read list of channels to scan for all Active and Energy Scans as a bitfield. This affects scans initiated in command mode (AS, ED) and during End Device Association and Coordinator startup: bit 0 - 0x0B bit 4 - 0x0F bit 8 - 0x13 bit12 - 0x17 bit 1 - 0x0C bit 5 - 0x10 bit 9 - 0x14 bit13 - 0x18 bit 2 - 0x0D bit 6 - 0x11 bit 10 - 0x15 bit14 - 0x19 bit 3 - 0x0E bit 7 - 0x12 bit 11 - 0x16 bit 15 - 0x1A	0 - 0xFFFF [bitfield] (bits 0, 14, 15 not allowed on the XBee-PRO)	0x1FFE (all XBee-PRO Channels)
SD (v1.x80*)	Networking {Association}	**Scan Duration**. Set/Read the scan duration exponent. ***End Device*** - Duration of Active Scan during Association. ***Coordinator*** - If 'ReassignPANID' option is set on Coordinator [refer to A2 parameter], SD determines the length of time the Coordinator will scan channels to locate existing PANs. If 'ReassignChannel' option is set, SD determines how long the Coordinator will perform an Energy Scan to determine which channel it will operate on. 'Scan Time' is measured as (# of channels to scan] * (2 ^ SD) * 15.36ms). The number of channels to scan is set by the SC command. The XBee can scan up to 16 channels (SC = 0xFFFF). The XBee PRO can scan up to 13 channels (SC = 0x3FFE). Example: The values below show results for a 13 channel scan: If SD = 0, time = 0.18 sec SD = 8, time = 47.19 sec SD = 2, time = 0.74 sec SD = 10, time = 3.15 min SD = 4, time = 2.95 sec SD = 12, time = 12.58 min SD = 6, time = 11.80 sec SD = 14, time = 50.33 min	0-0x0F [exponent]	4
A1 (v1.x80*)	Networking {Association}	**End Device Association**. Set/Read End Device association options. bit 0 - ReassignPanID 0 - Will only associate with Coordinator operating on PAN ID that matches module ID 1 - May associate with Coordinator operating on any PAN ID bit 1 - ReassignChannel 0 - Will only associate with Coordinator operating on matching CH Channel setting 1 - May associate with Coordinator operating on any Channel bit 2 - AutoAssociate 0 - Device will not attempt Association 1 - Device attempts Association until success Note: This bit is used only for Non-Beacon systems. End Devices in Beacon-enabled system must always associate to a Coordinator bit 3 - PollCoordOnPinWake 0 - Pin Wake will not poll the Coordinator for indirect (pending) data 1 - Pin Wake will send Poll Request to Coordinator to extract any pending data bits 4 - 7 are reserved	0 - 0x0F [bitfield]	0
A2 (v1.x80*)	Networking {Association}	**Coordinator Association**. Set/Read Coordinator association options. bit 0 - ReassignPanID 0 - Coordinator will not perform Active Scan to locate available PAN ID. It will operate on ID (PAN ID). 1 - Coordinator will perform Active Scan to determine an available ID (PAN ID). If a PAN ID conflict is found, the ID parameter will change. bit 1 - ReassignChannel - 0 - Coordinator will not perform Energy Scan to determine free channel. It will operate on the channel determined by the CH parameter. 1 - Coordinator will perform Energy Scan to find a free channel, then operate on that channel. bit 2 - AllowAssociation - 0 - Coordinator will not allow any devices to associate to it. 1 - Coordinator will allow devices to associate to it. bits 3 - 7 are reserved	0 - 7 [bitfield]	0

Table 3-02. XBee®/XBee-PRO® Commands - Networking & Security (Sub-categories designated within {brackets})

AT Command	Command Category	Name and Description	Parameter Range	Default
AI (v1.x80*)	Networking {Association}	**Association Indication**. Read errors with the last association request: 0x00 - Successful Completion - Coordinator successfully started or End Device association complete 0x01 - Active Scan Timeout 0x02 - Active Scan found no PANs 0x03 - Active Scan found PAN, but the CoordinatorAllowAssociation bit is not set 0x04 - Active Scan found PAN, but Coordinator and End Device are not configured to support beacons 0x05 - Active Scan found PAN, but the Coordinator ID parameter does not match the ID parameter of the End Device 0x06 - Active Scan found PAN, but the Coordinator CH parameter does not match the CH parameter of the End Device 0x07 - Energy Scan Timeout 0x08 - Coordinator start request failed 0x09 - Coordinator could not start due to invalid parameter 0x0A - Coordinator Realignment is in progress 0x0B - Association Request not sent 0x0C - Association Request timed out - no reply was received 0x0D - Association Request had an Invalid Parameter 0x0E - Association Request Channel Access Failure. Request was not transmitted - CCA failure 0x0F - Remote Coordinator did not send an ACK after Association Request was sent 0x10 - Remote Coordinator did not reply to the Association Request, but an ACK was received after sending the request 0x11 - [reserved] 0x12 - Sync-Loss - Lost synchronization with a Beaconing Coordinator 0x13 - Disassociated - No longer associated to Coordinator 0xFF - RF Module is attempting to associate	0 - 0x13 [read-only]	-
DA (v1.x80*)	Networking {Association}	**Force Disassociation**. End Device will immediately disassociate from a Coordinator (if associated) and reattempt to associate.	-	-
FP (v1.x80*)	Networking {Association}	**Force Poll**. Request indirect messages being held by a coordinator.	-	-
AS (v1.x80*)	Networking {Association}	**Active Scan**. Send Beacon Request to Broadcast Address (0xFFFF) and Broadcast PAN (0xFFFF) on every channel. The parameter determines the time the radio will listen for Beacons on each channel. A PanDescriptor is created and returned for every Beacon received from the scan. Each PanDescriptor contains the following information: CoordAddress (SH, SL)<CR> CoordPanID (ID)<CR> CoordAddrMode <CR> 0x02 = 16-bit Short Address 0x03 = 64-bit Long Address Channel (CH parameter) <CR> SecurityUse<CR> ACLEntry<CR> SecurityFailure<CR> SuperFrameSpec<CR> (2 bytes): bit 15 - Association Permitted (MSB) bit 14 - PAN Coordinator bit 13 - Reserved bit 12 - Battery Life Extension bits 8-11 - Final CAP Slot bits 4-7 - Superframe Order bits 0-3 - Beacon Order GtsPermit<CR> RSSI<CR> (RSSI is returned as -dBm) TimeStamp<CR> (3 bytes) <CR> A carriage return <CR> is sent at the end of the AS command. The Active Scan is capable of returning up to 5 PanDescriptors in a scan. The actual scan time on each channel is measured as Time = [(2 ^SD PARAM) * 15.36] ms. Note the total scan time is this time multiplied by the number of channels to be scanned (16 for the XBee and 13 for the XBee-PRO). Also refer to SD command description.	0 - 6	-
ED (v1.x80*)	Networking {Association}	**Energy Scan**. Send an Energy Detect Scan. This parameter determines the length of scan on each channel. The maximal energy on each channel is returned & each value is followed by a carriage return. An additional carriage return is sent at the end of the command. The values returned represent the detected energy level in units of -dBm. The actual scan time on each channel is measured as Time = [(2 ^ED) * 15.36] ms. Note the total scan time is this time multiplied by the number of channels to be scanned (refer to SD parameter).	0 - 6	-
EE (v1.xA0*)	Networking {Security}	**AES Encryption Enable**. Disable/Enable 128-bit AES encryption support. Use in conjunction with the KY command.	0 - 1	0 (disabled)
KY (v1.xA0*)	Networking {Security}	**AES Encryption Key**. Set the 128-bit AES (Advanced Encryption Standard) key for encrypting/decrypting data. The KY register cannot be read.	0 - (any 16-Byte value)	-

* Firmware version in which the command was first introduced (firmware versions are numbered in hexadecimal notation.)

RF Interfacing

Table 3-03. XBee/XBee-PRO Commands - RF Interfacing

AT Command	Command Category	Name and Description	Parameter Range	Default
PL	RF Interfacing	**Power Level**. Select/Read the power level at which the RF module transmits conducted power.	0 - 4 (XBee / XBee-PRO) 0 = -10 / 10 dBm 1 = -6 / 12 dBm 2 = -4 / 14 dBm 3 = -2 / 16 dBm 4 = 0 / 18 dBm XBee-PRO International variant: PL=4: 10 dBm PL=3: 8 dBm PL=2: 2 dBm PL=1: -3 dBm PL=0: -3 dBm	4
CA (v1.x80*)	RF Interfacing	**CCA Threshold**. Set/read the CCA (Clear Channel Assessment) threshold. Prior to transmitting a packet, a CCA is performed to detect energy on the channel. If the detected energy is above the CCA Threshold, the module will not transmit the packet.	0x24 - 0x50 [-dBm]	0x2C (-44d dBm)

* Firmware version in which the command was first introduced (firmware versions are numbered in hexadecimal notation.)

Sleep (Low Power)

Table 3-04. XBee®/XBee-PRO® Commands - Sleep (Low Power)

AT Command	Command Category	Name and Description	Parameter Range	Default
SM	Sleep (Low Power)	**Sleep Mode**. Set/Read Sleep Mode configurations.	0 - 5 0 = No Sleep 1 = Pin Hibernate 2 = Pin Doze 3 = Reserved 4 = Cyclic sleep remote 5 = Cyclic sleep remote w/ pin wake-up 6 = [Sleep Coordinator] for backwards compatibility w/ v1.x6 only; otherwise, use CE command.	0
SO	Sleep (Low Power)	Sleep Options Set/Read the sleep mode options. Bit 0 - Poll wakeup disable 0 - Normal operations. A module configured for cyclic sleep will poll for data on waking. 1 - Disable wakeup poll. A module configured for cyclic sleep will not poll for data on waking. Bit 1 - ADC/DIO wakeup sampling disable. 0 - Normal operations. A module configured in a sleep mode with ADC/DIO sampling enabled will automatically perform a sampling on wakeup. 1 - Suppress sample on wakeup. A module configured in a sleep mode with ADC/DIO sampling enabled will not automatically sample on wakeup.	0-4	0
ST	Sleep (Low Power)	**Time before Sleep.** <NonBeacon firmware> Set/Read time period of inactivity (no serial or RF data is sent or received) before activating Sleep Mode. ST parameter is only valid with Cyclic Sleep settings (SM = 4 - 5). Coordinator and End Device ST values must be equal. Also note, the GT parameter value must always be less than the ST value. (If GT > ST, the configuration will render the module unable to enter into command mode.) If the ST parameter is modified, also modify the GT parameter accordingly.	1 - 0xFFFF [x 1 ms]	0x1388 (5000d)
SP	Sleep (Low Power)	**Cyclic Sleep Period.** <NonBeacon firmware> Set/Read sleep period for cyclic sleeping remotes. Coordinator and End Device SP values should always be equal. To send Direct Messages, set SP = 0. *End Device* - SP determines the sleep period for cyclic sleeping remotes. Maximum sleep period is 268 seconds (0x68B0). *Coordinator* - If non-zero, SP determines the time to hold an indirect message before discarding it. A Coordinator will discard indirect messages after a period of (2.5 * SP).	0 - 0x68B0 [x 10 ms]	0
DP (1.x80*)	Sleep (Low Power)	**Disassociated Cyclic Sleep Period.** <NonBeacon firmware> *End Device* - Set/Read time period of sleep for cyclic sleeping remotes that are configured for Association but are not associated to a Coordinator. (i.e. If a device is configured to associate, configured as a Cyclic Sleep remote, but does not find a Coordinator, it will sleep for DP time before reattempting association.) Maximum sleep period is 268 seconds (0x68B0). DP should be > 0 for NonBeacon systems.	1 - 0x68B0 [x 10 ms]	0x3E8 (1000d)

* Firmware version in which the command was first introduced (firmware versions are numbered in hexadecimal notation.)

Serial Interfacing

Table 3-05. XBee-PRO Commands - Serial Interfacing

AT Command	Command Category	Name and Description	Parameter Range	Default
BD	Serial Interfacing	**Interface Data Rate**. Set/Read the serial interface data rate for communications between the RF module serial port and host. Request non-standard baud rates with values above 0x80 using a terminal window. Read the BD register to find actual baud rate achieved.	0 - 7 (standard baud rates) 0 = 1200 bps 1 = 2400 2 = 4800 3 = 9600 4 = 19200 5 = 38400 6 = 57600 7 = 115200 0x80 - 0x3D090 (non-standard baud rates up to 250 Kbps)	3
RO	Serial Interfacing	**Packetization Timeout**. Set/Read number of character times of inter-character delay required before transmission. Set to zero to transmit characters as they arrive instead of buffering them into one RF packet.	0 - 0xFF [x character times]	3
AP (v1.x80*)	Serial Interfacing	**API Enable**. Disable/Enable API Mode.	0 - 2 0 =Disabled 1 = API enabled 2 = API enabled (w/escaped control characters)	0
NB	Serial Interfacing	**Parity.** Set/Read parity settings.	0 - 4 0 = 8-bit no parity 1 = 8-bit even 2 = 8-bit odd 3 = 8-bit mark 4 = 8-bit space	0
PR (v1.x80*)	Serial Interfacing	**Pull-up Resistor Enable**. Set/Read bitfield to configure internal pull-up resistor status for I/O lines Bitfield Map: bit 0 - AD4/DIO4 (pin11) bit 1 - AD3 / DIO3 (pin17) bit 2 - AD2/DIO2 (pin18) bit 3 - AD1/DIO1 (pin19) bit 4 - AD0 / DIO0 (pin20) bit 5 - RTS / AD6 / DIO6 (pin16) bit 6 - DTR / SLEEP_RQ / DI8 (pin9) bit 7 - DIN/CONFIG (pin3) Bit set to "1" specifies pull-up enabled; "0" specifies no pull-up	0 - 0xFF	0xFF

* Firmware version in which the command was first introduced (firmware versions are numbered in hexadecimal notation.)

I/O Settings

Table 3-06. XBee-PRO Commands - I/O Settings (sub-category designated within {brackets})

AT Command	Command Category	Name and Description	Parameter Range	Default
D8	I/O Settings	**DI8 Configuration**. Select/Read options for the DI8 line (pin 9) of the RF module.	0 - 1 0 = Disabled 3 = DI (1,2,4 & 5 n/a)	0
D7 (v1.x80*)	I/O Settings	**DIO7 Configuration**. Select/Read settings for the DIO7 line (pin 12) of the RF module. Options include CTS flow control and I/O line settings.	0 - 1 0 = Disabled 1 = CTS Flow Control 2 = (n/a) 3 = DI 4 = DO low 5 = DO high 6 = RS485 Tx Enable Low 7 = RS485 Tx Enable High	1
D6 (v1.x80*)	I/O Settings	**DIO6 Configuration**. Select/Read settings for the DIO6 line (pin 16) of the RF module. Options include RTS flow control and I/O line settings.	0 - 1 0 = Disabled 1 = RTS flow control 2 = (n/a) 3 = DI 4 = DO low 5 = DO high	0

Table 3-06. XBee-PRO Commands - I/O Settings (sub-category designated within {brackets})

AT Command	Command Category	Name and Description	Parameter Range	Default
D5 (v1.x80*)	I/O Settings	**DIO5 Configuration**. Configure settings for the DIO5 line (pin 15) of the RF module. Options include Associated LED indicator (blinks when associated) and I/O line settings.	0 - 1 0 = Disabled 1 = Associated indicator 2 = ADC 3 = DI 4 = DO low 5 = DO high	1
D0 - D4 (v1.xA0*)	I/O Settings	**(DIO4 -DIO4) Configuration**. Select/Read settings for the following lines: AD0/DIO0 (pin 20), AD1/DIO1 (pin 19), AD2/DIO2 (pin 18), AD3/DIO3 (pin 17), AD4/DIO4 (pin 11). Options include: Analog-to-digital converter, Digital Input and Digital Output.	0 - 1 0 = Disabled 1 = (n/a) 2 = ADC 3 = DI 4 = DO low 5 = DO high	0
IU (v1.xA0*)	I/O Settings	**I/O Output Enable.** Disables/Enables I/O data received to be sent out UART. The data is sent using an API frame regardless of the current AP parameter value.	0 - 1 0 = Disabled 1 = Enabled	1
IT (v1.xA0*)	I/O Settings	**Samples before TX.** Set/Read the number of samples to collect before transmitting data. Maximum number of samples is dependent upon the number of enabled inputs.	1 - 0xFF	1
IS (v1.xA0*)	I/O Settings	**Force Sample.** Force a read of all enabled inputs (DI or ADC). Data is returned through the UART. If no inputs are defined (DI or ADC), this command will return error.	8-bit bitmap (each bit represents the level of an I/O line setup as an output)	-
IO (v1.xA0*)	I/O Settings	**Digital Output Level.** Set digital output level to allow DIO lines that are setup as outputs to be changed through Command Mode.	-	-
IC (v1.xA0*)	I/O Settings	**DIO Change Detect.** Set/Read bitfield values for change detect monitoring. Each bit enables monitoring of DIO0 - DIO7 for changes. If detected, data is transmitted with DIO data only. Any samples queued waiting for transmission will be sent first.	0 - 0xFF [bitfield]	0 (disabled)
IR (v1.xA0*)	I/O Settings	**Sample Rate.** Set/Read sample rate. When set, this parameter causes the module to sample all enabled inputs at a specified interval.	0 - 0xFFFF [x 1 msec]	0
IA (v1.xA0*)	I/O Settings {I/O Line Passing}	**I/O Input Address**. Set/Read addresses of module to which outputs are bound. Setting all bytes to 0xFF will not allow any received I/O packet to change outputs. Setting address to 0xFFFF will allow any received I/O packet to change outputs.	0 - 0xFFFFFFFFFFFFFFFF	0xFFFFFFFF FFFFFFFF
T0 - T7 (v1.xA0*)	I/O Settings {I/O Line Passing}	**(D0 - D7) Output Timeout.** Set/Read Output timeout values for lines that correspond with the D0 - D7 parameters. When output is set (due to I/O line passing) to a non-default level, a timer is started which when expired will set the output to it default level. The timer is reset when a valid I/O packet is received.	0 - 0xFF [x 100 ms]	0xFF
P0	I/O Settings {I/O Line Passing}	**PWM0 Configuration**. Select/Read function for PWM0 pin.	0 - 2 0 = Disabled 1 = RSSI 2 = PWM Output	1
P1 (v1.xA0*)	I/O Settings {I/O Line Passing}	**PWM1 Configuration**. Select/Read function for PWM1 pin.	0 - 2 0 = Disabled 1 = RSSI 2 = PWM Output	0
M0 (v1.xA0*)	I/O Settings {I/O Line Passing}	**PWM0 Output Level**. Set/Read the PWM0 output level.	0 - 0x03FF	-
M1 (v1.xA0*)	I/O Settings {I/O Line Passing}	**PWM1 Output Level**. Set/Read the PWM1 output level.	0 - 0x03FF	-
PT (v1.xA0*)	I/O Settings {I/O Line Passing}	**PWM Output Timeout.** Set/Read output timeout value for both PWM outputs. When PWM is set to a non-zero value: Due to I/O line passing, a time is started which when expired will set the PWM output to zero. The timer is reset when a valid I/O packet is received.]	0 - 0xFF [x 100 ms]	0xFF
RP	I/O Settings {I/O Line Passing}	**RSSI PWM Timer.** Set/Read PWM timer register. Set the duration of PWM (pulse width modulation) signal output on the RSSI pin. The signal duty cycle is updated with each received packet and is shut off when the timer expires.]	0 - 0xFF [x 100 ms]	0x28 (40d)

* Firmware version in which the command was first introduced (firmware versions are numbered in hexadecimal notation.)

Diagnostics

Table 3-07. XBee®/XBee-PRO® Commands - Diagnostics

AT Command	Command Category	Name and Description	Parameter Range	Default
VR	Diagnostics	**Firmware Version**. Read firmware version of the RF module.	0 - 0xFFFF [read-only]	Factory-set
VL (v1.x80*)	Diagnostics	**Firmware Version - Verbose**. Read detailed version information (including application build date, MAC, PHY and bootloader versions). The VL command has been deprecated in version 10C9. It is not supported in firmware versions after 10C8	-	-

Table 3-07. XBee®/XBee-PRO® Commands - Diagnostics

AT Command	Command Category	Name and Description	Parameter Range	Default
HV (v1.x80*)	Diagnostics	**Hardware Version**. Read hardware version of the RF module.	0 - 0xFFFF [read-only]	Factory-set
DB	Diagnostics	**Received Signal Strength**. Read signal level [in dB] of last good packet received (RSSI). Absolute value is reported. (For example: 0x58 = -88 dBm) Reported value is accurate between -40 dBm and RX sensitivity.	0x17-0x5C (XBee) 0x24-0x64 (XBee-PRO) [read-only]	-
EC (v1.x80*)	Diagnostics	**CCA Failures.** Reset/Read count of CCA (Clear Channel Assessment) failures. This parameter value increments when the module does not transmit a packet because it detected energy above the CCA threshold level set with CA command. This count saturates at its maximum value. Set count to “0” to reset count.	0 - 0xFFFF	-
EA (v1.x80*)	Diagnostics	**ACK Failures**. Reset/Read count of acknowledgment failures. This parameter value increments when the module expires its transmission retries without receiving an ACK on a packet transmission. This count saturates at its maximum value. Set the parameter to “0” to reset count.	0 - 0xFFFF	-
ED (v1.x80*)	Diagnostics	**Energy Scan**. Send ‘Energy Detect Scan’. ED parameter determines the length of scan on each channel. The maximal energy on each channel is returned and each value is followed by a carriage return. Values returned represent detected energy levels in units of -dBm. Actual scan time on each channel is measured as Time = [(2 ^ SD) * 15.36] ms. Total scan time is this time multiplied by the number of channels to be scanned.	0 - 6	-

* Firmware version in which the command was first introduced (firmware versions are numbered in hexadecimal notation.)

AT Command Options

Table 3-08. XBee®/XBee-PRO® Commands - AT Command Options

AT Command	Command Category	Name and Description	Parameter Range	Default
CT	AT Command Mode Options	**Command Mode Timeout.** Set/Read the period of inactivity (no valid commands received) after which the RF module automatically exits AT Command Mode and returns to Idle Mode.	2 - 0xFFFF [x 100 ms]	0x64 (100d)
CN	AT Command Mode Options	**Exit Command Mode**. Explicitly exit the module from AT Command Mode.	--	--
AC (v1.xA0*)	AT Command Mode Options	**Apply Changes**. Explicitly apply changes to queued parameter value(s) and re-initialize module.	--	--
GT	AT Command Mode Options	**Guard Times**. Set required period of silence before and after the Command Sequence Characters of the AT Command Mode Sequence (GT+ CC + GT). The period of silence is used to prevent inadvertent entrance into AT Command Mode.	2 - 0x0CE4 [x 1 ms]	0x3E8 (1000d)
CC	AT Command Mode Options	**Command Sequence Character**. Set/Read the ASCII character value to be used between Guard Times of the AT Command Mode Sequence (GT+CC+GT). The AT Command Mode Sequence enters the RF module into AT Command Mode.	0 - 0xFF	0x2B ('+' ASCII)

* Firmware version in which the command was first introduced (firmware versions are numbered in hexadecimal notation.)

Command Descriptions

Command descriptions in this section are listed alphabetically. Command categories are designated within "< >" symbols that follow each command title. XBee®/XBee-PRO® RF Modules expect parameter values in hexadecimal (designated by the "0x" prefix).

All modules operating within the same network should contain the same firmware version.

A1 (End Device Association) Command

<Networking {Association}> The A1 command is used to set and read association options for an End Device.

Use the table below to determine End Device behavior in relation to the A1 parameter.

AT Command: ATA1

Parameter Range: 0 – 0x0F [bitfield]

Default Parameter Value: 0

Related Commands: ID (PAN ID), NI (Node Identifier), CH (Channel), CE (Coordinator Enable), A2 (Coordinator Association)

Minimum Firmware Version Required: v1.x80

Bit number	End Device Association Option
0 - ReassignPanID	0 - Will only associate with Coordinator operating on PAN ID that matches Node Identifier
	1 - May associate with Coordinator operating on any PAN ID
1 - ReassignChannel	0 - Will only associate with Coordinator operating on Channel that matches CH setting
	1 - May associate with Coordinator operating on any Channel
2 - AutoAssociate	0 - Device will not attempt Association
	1 - Device attempts Association until success Note: This bit is used only for Non-Beacon systems. End Devices in a Beaconing system must always associate to a Coordinator
3 - PollCoordOnPinWake	0 - Pin Wake will not poll the Coordinator for pending (indirect) Data
	1 - Pin Wake will send Poll Request to Coordinator to extract any pending data
4 - 7	[reserved]

A2 (Coordinator Association) Command

<Networking {Association}> The A2 command is used to set and read association options of the Coordinator.

Use the table below to determine Coordinator behavior in relation to the A2 parameter.

AT Command: ATA2

Parameter Range: 0 – 7 [bitfield]

Default Parameter Value: 0

Related Commands: ID (PAN ID), NI (Node Identifier), CH (Channel), CE (Coordinator Enable), A1 (End Device Association), AS Active Scan), ED (Energy Scan)

Minimum Firmware Version Required: v1.x80

Bit number	End Device Association Option
0 - ReassignPanID	0 - Coordinator will not perform Active Scan to locate available PAN ID. It will operate on ID (PAN ID).
	1 - Coordinator will perform Active Scan to determine an available ID (PAN ID). If a PAN ID conflict is found, the ID parameter will change.
1 - ReassignChannel	0 - Coordinator will not perform Energy Scan to determine free channel. It will operate on the channel determined by the CH parameter.
	1 - Coordinator will perform Energy Scan to find a free channel, then operate on that channel.
2 - AllowAssociate	0 - Coordinator will not allow any devices to associate to it.
	1 - Coordinator will allow devices to associate to it.
3 - 7	[reserved]

The binary equivalent of the default value (0x06) is 00000110. 'Bit 0' is the last digit of the sequence.

AC (Apply Changes) Command

<AT Command Mode Options> The AC command is used to explicitly apply changes to module parameter values. 'Applying changes' means that the module is re-initialized based on changes made to its parameter values. Once changes are applied, the module immediately operates according to the new parameter values.

AT Command: ATAC

Minimum Firmware Version Required: v1.xA0

This behavior is in contrast to issuing the WR (Write) command. The WR command saves parameter values to non-volatile memory, but the module still operates according to previously saved values until the module is re-booted or the CN (Exit AT Command Mode) command is issued.

Refer to the "AT Command – Queue Parameter Value" API type for more information.

AI (Association Indication) Command

<Networking {Association}> The AI command is used to indicate occurrences of errors during the last association request.

Use the table below to determine meaning of the returned values.

AT Command: ATAI

Parameter Range: 0 – 0x13 [read-only]

Related Commands: AS (Active Scan), ID (PAN ID), CH (Channel), ED (Energy Scan), A1 (End Device Association), A2 (Coordinator Association), CE (Coordinator Enable)

Minimum Firmware Version Required: v1.x80

Returned Value (Hex)	Association Indication
0x00	Successful Completion - Coordinator successfully started or End Device association complete
0x01	Active Scan Timeout
0x02	Active Scan found no PANs
0x03	Active Scan found PAN, but the Coordinator Allow Association bit is not set
0x04	Active Scan found PAN, but Coordinator and End Device are not configured to support beacons
0x05	Active Scan found PAN, but Coordinator ID (PAN ID) value does not match the ID of the End Device
0x06	Active Scan found PAN, but Coordinator CH (Channel) value does not match the CH of the End Device
0x07	Energy Scan Timeout
0x08	Coordinator start request failed
0x09	Coordinator could not start due to Invalid Parameter
0x0A	Coordinator Realignment is in progress
0x0B	Association Request not sent
0x0C	Association Request timed out - no reply was received
0x0D	Association Request had an Invalid Parameter
0x0E	Association Request Channel Access Failure - Request was not transmitted - CCA failure
0x0F	Remote Coordinator did not send an ACK after Association Request was sent
0x10	Remote Coordinator did not reply to the Association Request, but an ACK was received after sending the request
0x11	[reserved]
0x12	Sync-Loss - Lost synchronization with a Beaconing Coordinator
0x13	Disassociated - No longer associated to Coordinator
0xFF	RF Module is attempting to associate

AP (API Enable) Command

<Serial Interfacing> The AP command is used to enable the RF module to operate using a frame-based API instead of using the default Transparent (UART) mode.

AT Command: ATAP

Parameter Range:0 – 2

Parameter	Configuration
0	Disabled (Transparent operation)
1	API enabled
2	API enabled (with escaped characters)

Default Parameter Value:0

Minimum Firmware Version Required: v1.x80

Refer to the API Operation section when API operation is enabled (AP = 1 or 2).

AS (Active Scan) Command

<Network {Association}> The AS command is used to send a Beacon Request to a Broadcast (0xFFFF) and Broadcast PAN (0xFFFF) on every channel. The parameter determines the amount of time the RF module will listen for Beacons on each channel. A 'PanDescriptor' is created and returned for every Beacon received from the scan. Each PanDescriptor contains the following information:

AT Command: ATAS

Parameter Range: 0 – 6

Related Command: SD (Scan Duration), DL (Destination Low Address), DH (Destination High Address), ID (PAN ID), CH (Channel)

Minimum Firmware Version Required: v1.x80

CoordAddress (SH + SL parameters)<CR> (NOTE: If MY on the coordinator is set less than 0xFFFF, the MY value is displayed)
CoordPanID (ID parameter)<CR>
CoordAddrMode <CR>
  0x02 = 16-bit Short Address
  0x03 = 64-bit Long Address
Channel (CH parameter) <CR>
SecurityUse<CR>
ACLEntry<CR>
SecurityFailure<CR>
SuperFrameSpec<CR> (2 bytes):
  bit 15 - Association Permitted (MSB)
  bit 14 - PAN Coordinator
  bit 13 - Reserved
  bit 12 - Battery Life Extension
  bits 8-11 - Final CAP Slot
  bits 4-7 - Superframe Order
  bits 0-3 - Beacon Order
GtsPermit<CR>
RSSI<CR> (- RSSI is returned as -dBm)
TimeStamp<CR> (3 bytes)
<CR> (A carriage return <CR> is sent at the end of the AS command.

The Active Scan is capable of returning up to 5 PanDescriptors in a scan. The actual scan time on each channel is measured as Time = [(2 ^ (SD Parameter)) * 15.36] ms. Total scan time is this time multiplied by the number of channels to be scanned (16 for the XBee, 12 for the XBee-PRO).

NOTE: Refer the scan table in the SD description to determine scan times. If using API Mode, no <CR>'s are returned in the response. Refer to the API Mode Operation section.

BD (Interface Data Rate) Command

<Serial Interfacing> The BD command is used to set and read the serial interface data rate used between the RF module and host. This parameter determines the rate at which serial data is sent to the module from the host. Modified interface data rates do not take effect until the CN (Exit AT Command Mode) command is issued and the system returns the 'OK' response.

When parameters 0-7 are sent to the module, the respective interface data rates are used (as shown in the table on the right).

The RF data rate is not affected by the BD parameter. If the interface data rate is set higher than the RF data rate, a flow control configuration may need to be implemented.

AT Command: ATBD

Parameter Range:0 – 7 (standard rates) 0x80–0x3D090 (non–standard rates up to 250 Kbps)

Parameter	Configuration (bps)
0	1200
1	2400
2	4800
3	9600
4	19200
5	38400
6	57600
7	115200

Default Parameter Value:3

Non-standard Interface Data Rates:
Any value above 0x07 will be interpreted as an actual baud rate. When a value above 0x07 is sent, the closest interface data rate represented by the number is stored in the BD register. For example, a rate of 19200 bps can be set by sending the following command line "ATBD4B00". NOTE: When using Digi's X-CTU Software, non-standard interface data rates can only be set and read using the X-CTU 'Terminal' tab. Non-standard rates are not accessible through the 'Modem Configuration' tab.

When the BD command is sent with a non-standard interface data rate, the UART will adjust to accommodate the requested interface rate. In most cases, the clock resolution will cause the stored BD parameter to vary from the parameter that was sent (refer to the table below). Reading the BD command (send "ATBD" command without an associated parameter value) will return the value actually stored in the module's BD register.

Parameters Sent Versus Parameters Stored

BD Parameter Sent (HEX)	Interface Data Rate (bps)	BD Parameter Stored (HEX)
0	1200	0
4	19,200	4
7	115,200*	7
12C	300	12B
1C200	115,200	1B207

* The 115,200 baud rate setting is actually at 111,111 baud (-3.5% target UART speed).

CA (CCA Threshold) Command

<RF Interfacing> CA command is used to set and read CCA (Clear Channel Assessment) thresholds.

Prior to transmitting a packet, a CCA is performed to detect energy on the transmit channel. If the detected energy is above the CCA Threshold, the RF module will not transmit the packet.

AT Command: ATCA

Parameter Range: 0 – 0x50 [–dBm]

Default Parameter Value: 0x2C (–44 decimal dBm)

Minimum Firmware Version Required: v1.x80

CC (Command Sequence Character) Command

<AT Command Mode Options> The CC command is used to set and read the ASCII character used between guard times of the AT Command Mode Sequence (GT + CC + GT). This sequence enters the RF module into AT Command Mode so that data entering the module from the host is recognized as commands instead of payload.

AT Command: ATCC

Parameter Range: 0 – 0xFF

Default Parameter Value: 0x2B (ASCII “+”)

Related Command: GT (Guard Times)

The AT Command Sequence is explained further in the AT Command Mode section.

CE (Coordinator Enable) Command

<Networking {Association} The CE command is used to set and read the behavior (End Device vs. Coordinator) of the RF module.

AT Command: ATCE

Parameter Range:0 – 1

Parameter	Configuration
0	End Device
1	Coordinator

Default Parameter Value:0

Minimum Firmware Version Required: v1.x80

CH (Channel) Command

<Networking {Addressing}> The CH command is used to set/read the operating channel on which RF connections are made between RF modules. The channel is one of three addressing options available to the module. The other options are the PAN ID (ID command) and destination addresses (DL & DH commands).

AT Command: ATCH

Parameter Range: 0x0B – 0x1A (XBee)
0x0C – 0x17 (XBee-PRO)

Default Parameter Value: 0x0C (12 decimal)

Related Commands: ID (PAN ID), DL (Destination Address Low, DH (Destination Address High)

In order for modules to communicate with each other, the modules must share the same channel number. Different channels can be used to prevent modules in one network from listening to transmissions of another. Adjacent channel rejection is 23 dB.

The module uses channel numbers of the 802.15.4 standard.

Center Frequency = 2.405 + (CH - 11d) * 5 MHz (d = decimal)

Refer to the XBee/XBee-PRO Addressing section for more information.

CN (Exit Command Mode) Command

<AT Command Mode Options> The CN command is used to explicitly exit the RF module from AT Command Mode.

AT Command: ATCN

CT (Command Mode Timeout) Command

<AT Command Mode Options> The CT command is used to set and read the amount of inactive time that elapses before the RF module automatically exits from AT Command Mode and returns to Idle Mode.

Use the CN (Exit Command Mode) command to exit AT Command Mode manually.

AT Command: ATCT

Parameter Range:2 – 0xFFFF
[x 100 milliseconds]

Default Parameter Value: 0x64 (100 decimal (which equals 10 decimal seconds))

Number of bytes returned: 2

Related Command: CN (Exit Command Mode)

D0 - D4 (DIOn Configuration) Commands

<I/O Settings> The D0, D1, D2, D3 and D4 commands are used to select/read the behavior of their respective AD/DIO lines (pins 20, 19, 18, 17 and 11 respectively).

Options include:

- Analog-to-digital converter
- Digital input
- Digital output

AT Commands:
ATD0, ATD1, ATD2, ATD3, ATD4

Parameter Range:0 – 5

Parameter	Configuration
0	Disabled
1	n/a
2	ADC
3	DI
4	DO low
5	DO high

Default Parameter Value:0

Minimum Firmware Version Required: 1.x.A0

D5 (DIO5 Configuration) Command

<I/O Settings> The D5 command is used to select/read the behavior of the DIO5 line (pin 15).

Options include:

- Associated Indicator (LED blinks when the module is associated)
- Analog-to-digital converter
- Digital input
- Digital output

AT Command: ATD5

Parameter Range:0 – 5

Parameter	Configuration
0	Disabled
1	Associated Indicator
2	ADC
3	DI
4	DO low
5	DO high

Default Parameter Value:1

Parameters 2–5 supported as of firmware version 1.xA0

D6 (DIO6 Configuration) Command

<I/O Settings> The D6 command is used to select/read the behavior of the DIO6 line (pin 16). Options include:

- RTS flow control
- Analog-to-digital converter
- Digital input
- Digital output

AT Command: ATD6

Parameter Range:0 – 5

Parameter	Configuration
0	Disabled
1	RTS Flow Control
2	n/a
3	DI
4	DO low
5	DO high

Default Parameter Value:0

Parameters 3–5 supported as of firmware version 1.xA0

D7 (DIO7 Configuration) Command

<I/O Settings> The D7 command is used to select/read the behavior of the DIO7 line (pin 12). Options include:

- CTS flow control
- Analog-to-digital converter
- Digital input
- Digital output
- RS485 TX Enable (this output is 3V CMOS level, and is useful in a 3V CMOS to RS485 conversion circuit)

AT Command: ATD7

Parameter Range:0 – 5

Parameter	Configuration
0	Disabled
1	CTS Flow Control
2	n/a
3	DI
4	DO low
5	DO high
6	RS485 TX Enable Low
7	RS485 TX Enable High

Default Parameter Value:1

Parameters 3–7 supported as of firmware version 1.x.A0

D8 (DI8 Configuration) Command

<I/O Settings> The D8 command is used to select/read the behavior of the DI8 line (pin 9). This command enables configuring the pin to function as a digital input. This line is also used with Pin Sleep.

AT Command: ATD8

Parameter Range:0 – 5
(1, 2, 4 & 5 n/a)

Parameter	Configuration
0	Disabled
3	DI

Default Parameter Value:0

Minimum Firmware Version Required: 1.xA0

DA (Force Disassociation) Command

<(Special)> The DA command is used to immediately disassociate an End Device from a Coordinator and reattempt to associate.

AT Command: ATDA

Minimum Firmware Version Required: v1.x80

DB (Received Signal Strength) Command

<Diagnostics> DB parameter is used to read the received signal strength (in dBm) of the last RF packet received. Reported values are accurate between -40 dBm and the RF module's receiver sensitivity.

AT Command: ATDB

Parameter Range [read-only]: 0x17–0x5C (XBee), 0x24–0x64 (XBee-PRO)

Absolute values are reported. For example: 0x58 = -88 dBm (decimal). If no packets have been received (since last reset, power cycle or sleep event), "0" will be reported.

DH (Destination Address High) Command

<Networking {Addressing}> The DH command is used to set and read the upper 32 bits of the RF module's 64-bit destination address. When combined with the DL (Destination Address Low) parameter, it defines the destination address used for transmission.

An module will only communicate with other modules having the same channel (CH parameter), PAN ID (ID parameter) and destination address (DH + DL parameters).

AT Command: ATDH

Parameter Range: 0 – 0xFFFFFFFF

Default Parameter Value: 0

Related Commands: DL (Destination Address Low), CH (Channel), ID (PAN VID), MY (Source Address)

To transmit using a 16-bit address, set the DH parameter to zero and the DL parameter less than 0xFFFF. 0x000000000000FFFF (DL concatenated to DH) is the broadcast address for the PAN.

Refer to the XBee/XBee-PRO Addressing section for more information.

DL (Destination Address Low) Command

<Networking {Addressing}> The DL command is used to set and read the lower 32 bits of the RF module's 64-bit destination address. When combined with the DH (Destination Address High) parameter, it defines the destination address used for transmission.

AT Command: ATDL
Parameter Range: 0 - 0xFFFFFFFF
Default Parameter Value: 0
Related Commands: DH (Destination Address High), CH (Channel), ID (PAN VID), MY (Source Address)

A module will only communicate with other modules having the same channel (CH parameter), PAN ID (ID parameter) and destination address (DH + DL parameters).

To transmit using a 16-bit address, set the DH parameter to zero and the DL parameter less than 0xFFFF. 0x000000000000FFFF (DL concatenated to DH) is the broadcast address for the PAN.

Refer to the XBee/XBee-PRO Addressing section for more information.

DN (Destination Node) Command

<Networking {Identification}> The DN command is used to resolve a NI (Node Identifier) string to a physical address. The following events occur upon successful command execution:

AT Command: ATDN
Parameter Range: 20–character ASCII String
Minimum Firmware Version Required: v1.x80

1. DL and DH are set to the address of the module with the matching NI (Node Identifier).
2. 'OK' is returned.
3. RF module automatically exits AT Command Mode.

If there is no response from a modem within 200 msec or a parameter is not specified (left blank), the command is terminated and an 'ERROR' message is returned.

DP (Disassociation Cyclic Sleep Period) Command

<Sleep Mode (Low Power)>

NonBeacon Firmware

End Device - The DP command is used to set and read the time period of sleep for cyclic sleeping remotes that are configured for Association but are not associated to a Coordinator. (i.e. If a device is configured to associate, configured as a Cyclic Sleep remote, but does not find a Coordinator; it will sleep for DP time before reattempting association.) Maximum sleep period is 268 seconds (0x68B0). DP should be > 0 for NonBeacon systems.

AT Command: ATDP
Parameter Range: 1 – 0x68B0 [x 10 milliseconds]
Default Parameter Value:0x3E8 (1000 decimal)
Related Commands: SM (Sleep Mode), SP (Cyclic Sleep Period), ST (Time before Sleep)
Minimum Firmware Version Required: v1.x80

EA (ACK Failures) Command

<Diagnostics> The EA command is used to reset and read the count of ACK (acknowledgement) failures. This parameter value increments when the module expires its transmission retries without receiving an ACK on a packet transmission. This count saturates at its maximum value.

AT Command: ATEA
Parameter Range:0 – 0xFFFF
Minimum Firmware Version Required: v1.x80

Set the parameter to "0" to reset count.

EC (CCA Failures) Command

<Diagnostics> The EC command is used to read and reset the count of CCA (Clear Channel Assessment) failures. This parameter value increments when the RF module does not transmit a packet due to the detection of energy that is above the CCA threshold level (set with CA command). This count saturates at its maximum value.

Set the EC parameter to "0" to reset count.

AT Command: ATEC

Parameter Range:0 – 0xFFFF

Related Command: CA (CCA Threshold)

Minimum Firmware Version Required: v1.x80

ED (Energy Scan) Command

<Networking {Association}> The ED command is used to send an "Energy Detect Scan". This parameter determines the length of scan on each channel. The maximal energy on each channel is returned and each value is followed by a carriage return. An additional carriage return is sent at the end of the command.

AT Command: ATED

Parameter Range:0 – 6

Related Command: SD (Scan Duration), SC (Scan Channel)

Minimum Firmware Version Required: v1.x80

The values returned represent the detected energy level in units of -dBm. The actual scan time on each channel is measured as Time = [(2 ^ ED PARAM) * 15.36] ms.

Note: Total scan time is this time multiplied by the number of channels to be scanned. Also refer to the SD (Scan Duration) table. Use the SC (Scan Channel) command to choose which channels to scan.

EE (AES Encryption Enable) Command

<Networking {Security}> The EE command is used to set/read the parameter that disables/enables 128-bit AES encryption.

The XBee®/XBee-PRO® firmware uses the 802.15.4 Default Security protocol and uses AES encryption with a 128-bit key. AES encryption dictates that all modules in the network use the same key and the maximum RF packet size is 95 Bytes.

When encryption is enabled, the module will always use its 64-bit long address as the source address for RF packets. This does not affect how the MY (Source Address), DH (Destination Address High) and DL (Destination Address Low) parameters work

AT Command: ATEE

Parameter Range:0 – 1

Parameter	Configuration
0	Disabled
1	Enabled

Default Parameter Value:0

Related Commands: KY (Encryption Key), AP (API Enable), MM (MAC Mode)

Minimum Firmware Version Required: v1.xA0

If MM (MAC Mode) > 0 and AP (API Enable) parameter > 0:
With encryption enabled and a 16-bit short address set, receiving modules will only be able to issue RX (Receive) 64-bit indicators. This is not an issue when MM = 0.

If a module with a non-matching key detects RF data, but has an incorrect key: When encryption is enabled, non-encrypted RF packets received will be rejected and will not be sent out the UART.

Transparent Operation --> All RF packets are sent encrypted if the key is set.

API Operation --> Receive frames use an option bit to indicate that the packet was encrypted.

FP (Force Poll) Command

<Networking (Association)> The FP command is used to request indirect messages being held by a Coordinator.

AT Command: ATFP

Minimum Firmware Version Required: v1.x80

FR (Software Reset) Command

<Special> The FR command is used to force a software reset on the RF module. The reset simulates powering off and then on again the module.

AT Command: ATFR

Minimum Firmware Version Required: v1.x80

GT (Guard Times) Command

<AT Command Mode Options> GT Command is used to set the DI (data in from host) time-of-silence that surrounds the AT command sequence character (CC Command) of the AT Command Mode sequence (GT + CC + GT).

The DI time-of-silence is used to prevent inadvertent entrance into AT Command Mode.

Refer to the Command Mode section for more information regarding the AT Command Mode Sequence.

AT Command: ATGT

Parameter Range:2 – 0x0CE4 [x 1 millisecond]

Default Parameter Value:0x3E8 (1000 decimal)

Related Command: CC (Command Sequence Character)

HV (Hardware Version) Command

<Diagnostics> The HV command is used to read the hardware version of the RF module.

AT Command: ATHV

Parameter Range:0 – 0xFFFF [Read-only]

Minimum Firmware Version Required: v1.x80

IA (I/O Input Address) Command

<I/O Settings {I/O Line Passing}> The IA command is used to bind a module output to a specific address. Outputs will only change if received from this address. The IA command can be used to set/read both 16 and 64-bit addresses.

Setting all bytes to 0xFF will not allow the reception of any I/O packet to change outputs. Setting the IA address to 0xFFFF will cause the module to accept all I/O packets.

AT Command: ATIA

Parameter Range:0 – 0xFFFFFFFFFFFFFFFF

Default Parameter Value:0xFFFFFFFFFFFFFFFF (will not allow any received I/O packet to change outputs)

Minimum Firmware Version Required: v1.xA0

IC (DIO Change Detect) Command

<I/O Settings> Set/Read bitfield values for change detect monitoring. Each bit enables monitoring of DIO0 - DIO7 for changes.

If detected, data is transmitted with DIO data only. Any samples queued waiting for transmission will be sent first.

Refer to the "ADC and Digital I/O Line Support" sections of the "RF Module Operations" chapter for more information.

AT Command: ATIC

Parameter Range:0 – 0xFF [bitfield]

Default Parameter Value:0 (disabled)

Minimum Firmware Version Required: 1.xA0

ID (Pan ID) Command

<Networking {Addressing}> The ID command is used to set and read the PAN (Personal Area Network) ID of the RF module. Only modules with matching PAN IDs can communicate with each other. Unique PAN IDs enable control of which RF packets are received by a module.

Setting the ID parameter to 0xFFFF indicates a global transmission for all PANs. It does not indicate a global receive.

AT Command: ATID

Parameter Range: 0 – 0xFFFF

Default Parameter Value:0x3332 (13106 decimal)

IO (Digital Output Level) Command

<I/O Settings> The IO command is used to set digital output levels. This allows DIO lines setup as outputs to be changed through Command Mode.

AT Command: ATIO
Parameter Range: 8-bit bitmap (where each bit represents the level of an I/O line that is setup as an output.)
Minimum Firmware Version Required: v1.xA0

IR (Sample Rate) Command

<I/O Settings> The IR command is used to set/read the sample rate. When set, the module will sample all enabled DIO/ADC lines at a specified interval. This command allows periodic reads of the ADC and DIO lines in a non-Sleep Mode setup. A sample rate which requires transmissions at a rate greater than once every 20ms is not recommended.

AT Command: ATIR
Parameter Range: 0 - 0xFFFF [x 1 msec] (cannot guarantee 1 ms timing when IT=1)
Default Parameter Value:0
Related Command: IT (Samples before TX)
Minimum Firmware Version Required: v1.xA0

Example: When IR = 0x14, the sample rate is 20 ms (or 50 Hz).

IS (Force Sample) Command

<I/O Settings> The IS command is used to force a read of all enabled DIO/ADC lines. The data is returned through the UART.

When operating in Transparent Mode (AP=0), the data is retuned in the following format:

AT Command: ATIS
Parameter Range: 1 - 0xFF
Default Parameter Value:1
Minimum Firmware Version Required: v1.xA0

All bytes are converted to ASCII:
number of samples<CR>
channel mask<CR>
DIO data<CR> (If DIO lines are enabled<CR>
ADC channel Data<cr> <-This will repeat for every enabled ADC channel<CR>
<CR> (end of data noted by extra <CR>)

When operating in API mode (AP > 0), the command will immediately return an 'OK' response. The data will follow in the normal API format for DIO data.

IT (Samples before TX) Command

<I/O Settings> The IT command is used to set/read the number of DIO and ADC samples to collect before transmitting data.

AT Command: ATIT
Parameter Range: 1 - 0xFF
Default Parameter Value:1
Minimum Firmware Version Required: v1.xA0

One ADC sample is considered complete when all enabled ADC channels have been read. The module can buffer up to 93 Bytes of sample data.
Since the module uses a 10-bit A/D converter, each sample uses two Bytes. This leads to a maximum buffer size of 46 samples or IT=0x2E.

When Sleep Modes are enabled and IR (Sample Rate) is set, the module will remain awake until IT samples have been collected.

IU (I/O Output Enable) Command

<I/O Settings> The IU command is used to disable/enable I/O UART output. When enabled (IU = 1), received I/O line data packets are sent out the UART. The data is sent using an API frame regardless of the current AP parameter value.

AT Command: ATIU

Parameter Range:0 – 1

Parameter	Configuration
0	Disabled – Received I/O line data packets will be NOT sent out UART.
1	Enabled – Received I/O line data will be sent out UART

Default Parameter Value:1

Minimum Firmware Version Required: 1.xA0

KY (AES Encryption Key) Command

<Networking {Security}> The KY command is used to set the 128-bit AES (Advanced Encryption Standard) key for encrypting/decrypting data. Once set, the key cannot be read out of the module by any means.

AT Command: ATKY

Parameter Range:0 – (any 16–Byte value)

Default Parameter Value:0

Related Command: EE (Encryption Enable)

Minimum Firmware Version Required: v1.xA0

The entire payload of the packet is encrypted using the key and the CRC is computed across the ciphertext. When encryption is enabled, each packet carries an additional 16 Bytes to convey the random CBC Initialization Vector (IV) to the receiver(s). The KY value may be "0" or any 128-bit value. Any other value, including entering KY by itself with no parameters, is invalid. All ATKY entries (valid or not) are received with a returned 'OK'.

A module with the wrong key (or no key) will receive encrypted data, but the data driven out the serial port will be meaningless. A module with a key and encryption enabled will receive data sent from a module without a key and the correct unencrypted data output will be sent out the serial port. Because CBC mode is utilized, repetitive data appears differently in different transmissions due to the randomly-generated IV.

When queried, the system will return an 'OK' message and the value of the key will not be returned.

M0 (PWM0 Output Level) Command

<I/O Settings> The M0 command is used to set/read the output level of the PWM0 line (pin 6).

Before setting the line as an output:

1. Enable PWM0 output (P0 = 2)
2. Apply settings (use CN or AC)

The PWM period is 64 µsec and there are 0x03FF (1023 decimal) steps within this period. When M0 = 0 (0% PWM), 0x01FF (50% PWM), 0x03FF (100% PWM), etc.

AT Command: ATM0

Parameter Range:0 – 0x03FF [steps]

Default Parameter Value:0

Related Commands: P0 (PWM0 Enable), AC (Apply Changes), CN (Exit Command Mode)

Minimum Firmware Version Required: v1.xA0

M1 (PWM1 Output Level) Command

<I/O Settings> The M1 command is used to set/read the output level of the PWM1 line (pin 7).

Before setting the line as an output:

1. Enable PWM1 output (P1 = 2)
2. Apply settings (use CN or AC)

AT Command: ATM1

Parameter Range:0 – 0x03FF

Default Parameter Value:0

Related Commands: P1 (PWM1 Enable), AC (Apply Changes), CN (Exit Command Mode)

Minimum Firmware Version Required: v1.xA0

MM (MAC Mode) Command

<Networking {Addressing}> The MM command is used to set and read the MAC Mode value. The MM command disables/enables the use of a Digi header contained in the 802.15.4 RF packet. By default (MM = 0), Digi Mode is enabled and the module adds an extra header to the data portion of the 802.15.4 packet. This enables the following features:

- ND and DN command support
- Duplicate packet detection when using ACKs
- "RR command
- "DIO/AIO sampling support

The MM command allows users to turn off the use of the extra header. Modes 1 and 2 are strict 802.15.4 modes. If the Digi header is disabled, ND and DN parameters are also disabled.

Note: When MM=0 or 3, application and CCA failure retries are not supported.

AT Command: ATMM

Parameter Range:0 – 3

Parameter	Configuration
0	Digi Mode (802.15.4 + Digi header)
1	802.15.4 (no ACKs)
2	802.15.4 (with ACKs)
3	Digi Mode (no ACKs)

Default Parameter Value:0

Related Commands: ND (Node Discover), DN (Destination Node)

Minimum Firmware Version Required: v1.x80

MY (16-bit Source Address) Command

<Networking {Addressing}> The MY command is used to set and read the 16-bit source address of the RF module.

By setting MY to 0xFFFF, the reception of RF packets having a 16-bit address is disabled. The 64-bit address is the module's serial number and is always enabled.

AT Command: ATMY

Parameter Range: 0 – 0xFFFF

Default Parameter Value: 0

Related Commands: DH (Destination Address High), DL (Destination Address Low), CH (Channel), ID (PAN ID)

NB (Parity) Command

<Serial Interfacing> The NB command is used to select/read the parity settings of the RF module for UART communications.

Note: the module does not actually calculate and check the parity; it only interfaces with devices at the configured parity and stop bit settings.

AT Command: ATNB

Parameter Range: 0 – 4

Parameter	Configuration
0	8-bit no parity
1	8-bit even
2	8-bit odd
3	8-bit mark
4	8-bit space

Default Parameter Value: 0

Number of bytes returned: 1

ND (Node Discover) Command

<Networking {Identification}> The ND command is used to discover and report all modules on its current operating channel (CH parameter) and PAN ID (ID parameter). ND also accepts an NI (Node Identifier) value as a parameter. In this case, only a module matching the supplied identifier will respond.

AT Command: ATND
Range: optional 20-character NI value
Related Commands: CH (Channel), ID (Pan ID), MY (Source Address), SH (Serial Number High), SL (Serial Number Low), NI (Node Identifier), NT (Node Discover Time)
Minimum Firmware Version Required: v1.x80

ND uses a 64-bit long address when sending and responding to an ND request. The ND command causes a module to transmit a globally addressed ND command packet. The amount of time allowed for responses is determined by the NT (Node Discover Time) parameter.

In AT Command mode, command completion is designated by a carriage return (0x0D). Since two carriage returns end a command response, the application will receive three carriage returns at the end of the command. If no responses are received, the application should only receive one carriage return. When in API mode, the application should receive a frame (with no data) and status (set to 'OK') at the end of the command. When the ND command packet is received, the remote sets up a random time delay (up to 2.2 sec) before replying as follows:

Node Discover Response (AT command mode format - Transparent operation):
MY (Source Address) value<CR>
SH (Serial Number High) value<CR>
SL (Serial Number Low) value<CR>
DB (Received Signal Strength) value<CR>
NI (Node Identifier) value<CR>
<CR> (This is part of the response and not the end of command indicator.)

Node Discover Response (API format - data is binary (except for NI)):
2 bytes for MY (Source Address) value
4 bytes for SH (Serial Number High) value
4 bytes for SL (Serial Number Low) value
1 byte for DB (Received Signal Strength) value
NULL-terminated string for NI (Node Identifier) value (max 20 bytes w/out NULL terminator)

NI (Node Identifier) Command

<Networking {Identification}> The NI command is used to set and read a string for identifying a particular node.

AT Command: ATNI
Parameter Range: 20-character ASCII string
Related Commands: ND (Node Discover), DN (Destination Node)
Minimum Firmware Version Required: v1.x80

Rules:

- Register only accepts printable ASCII data.
- A string can not start with a space.
- A carriage return ends command
- Command will automatically end when maximum bytes for the string have been entered.

This string is returned as part of the ND (Node Discover) command. This identifier is also used with the DN (Destination Node) command.

NO (Node Discover Options) Command

<Networking {Identification}> The NO command is used to suppress/include a self-response to Node Discover commands. When NO=1 a module doing a Node Discover will include a response entry for itself.

AT Command: ATNO
Parameter Range: "0-1
Related Commands: ND (Node Discover), DN (Destination Node)
Minimum Firmware Version Required: v1.xC5

NT (Node Discover Time) Command

<Networking {Identification}> The NT command is used to set the amount of time a base node will wait for responses from other nodes when using the ND (Node Discover) command. The NT value is transmitted with the ND command.

Remote nodes will set up a random hold-off time based on this time. The remotes will adjust this time down by 250 ms to give each node the ability to respond before the base ends the command. Once the ND command has ended, any response received on the base will be discarded.

AT Command: ATNT
Parameter Range: 0x01 – 0xFC [x 100 msec]
Default: 0x19 (2.5 decimal seconds)
Related Commands: ND (Node Discover)
Minimum Firmware Version Required: 1.xA0

P0 (PWM0 Configuration) Command

<I/O Setting {I/O Line Passing}> The P0 command is used to select/read the function for PWM0 (Pulse Width Modulation output 0). This command enables the option of translating incoming data to a PWM so that the output can be translated back into analog form.

With the IA (I/O Input Address) parameter correctly set, AD0 values can automatically be passed to PWM0.

AT Command: ATP0
The second character in the command is the number zero (“0”), not the letter “O”.

Parameter Range: 0 – 2

Parameter	Configuration
0	Disabled
1	RSSI
2	PWM0 Output

Default Parameter Value: 1

P1 (PWM1 Configuration) Command

<I/O Setting {I/O Line Passing}> The P1 command is used to select/read the function for PWM1 (Pulse Width Modulation output 1). This command enables the option of translating incoming data to a PWM so that the output can be translated back into analog form.

With the IA (I/O Input Address) parameter correctly set, AD1 values can automatically be passed to PWM1.

AT Command: ATP1

Parameter Range: 0 – 2

Parameter	Configuration
0	Disabled
1	RSSI
2	PWM1 Output

Default Parameter Value: 0

Minimum Firmware Version Required: v1.xA0

PL (Power Level) Command

<RF Interfacing> The PL command is used to select and read the power level at which the RF module transmits conducted power.

When operating in Europe, XBee-PRO 802.15.4 modules must operate at or below a transmit power output level of 10dBm. Customers have 2 choices for transmitting at or below 10dBm:

- Order the standard XBee-PRO module and change the PL command to "0" (10dBm),
- Order the International variant of the XBee-PRO module, which has a maximum transmit output power of 10dBm.

AT Command: ATPL

Parameter Range: 0 – 4

Parameter	XBee	XBee-PRO	XBee-PRO International variant
0	-10 dBm	10 dBm	PL=4: 10 dBm
1	-6 dBm	12 dBm	PL=3: 8 dBm
2	-4 dBm	14 dBm	PL=2: 2 dBm
3	-2 dBm	16 dBm	PL=1: -3 dBm
4	0 dBm	18 dBm	PL=0: -3 dBm

Default Parameter Value: 4

PR (Pull-up Resistor) Command

<Serial Interfacing> The PR command is used to set and read the bit field that is used to configure internal the pull-up resistor status for I/O lines. "1" specifies the pull-up resistor is enabled. "0" specifies no pull up.

AT Command: ATPR

Parameter Range: 0 – 0xFF

Default Parameter Value: 0xFF
(all pull–up resistors are enabled)

Minimum Firmware Version Required: v1.x80

bit 0 - AD4/DIO4 (pin 11)
bit 1 - AD3/DIO3 (pin 17)
bit 2 - AD2/DIO2 (pin 18)
bit 3 - AD1/DIO1 (pin 19)
bit 4 - AD0/DIO0 (pin 20)
bit 5 - AD6/DIO6 (pin 16)
bit 6 - DI8 (pin 9)
bit 7 - DIN/CONFIG (pin 3)

For example: Sending the command "ATPR 6F" will turn bits 0, 1, 2, 3, 5 and 6 ON; and bits 4 & 7 will be turned OFF. (The binary equivalent of "0x6F" is "01101111". Note that 'bit 0' is the last digit in the bitfield.

PT (PWM Output Timeout) Command

<I/O Settings {I/O Line Passing}> The PT command is used to set/read the output timeout value for both PWM outputs.

When PWM is set to a non-zero value: Due to I/O line passing, a time is started which when expired will set the PWM output to zero. The timer is reset when a valid I/O packet is received.

AT Command: ATPT

Parameter Range: 0 – 0xFF [x 100 msec]

Default Parameter Value: 0xFF

Minimum Firmware Version Required: 1.xA0

RE (Restore Defaults) Command

<(Special)> The RE command is used to restore all configurable parameters to their factory default settings. The RE command does not write restored values to non-volatile (persistent) memory. Issue the WR (Write) command subsequent to issuing the RE command to save restored parameter values to non-volatile memory.

AT Command: ATRE

RN (Random Delay Slots) Command

<Networking & Security> The RN command is used to set and read the minimum value of the back-off exponent in the CSMA-CA algorithm. The CSMA-CA algorithm was engineered for collision avoidance (random delays are inserted to prevent data loss caused by data collisions).

AT Command: ATRN

Parameter Range: 0 – 3 [exponent]

Default Parameter Value: 0

If RN = 0, collision avoidance is disabled during the first iteration of the algorithm (802.15.4 - macMinBE).

CSMA-CA stands for "Carrier Sense Multiple Access - Collision Avoidance". Unlike CSMA-CD (reacts to network transmissions after collisions have been detected), CSMA-CA acts to prevent data collisions before they occur. As soon as a module receives a packet that is to be transmitted, it checks if the channel is clear (no other module is transmitting). If the channel is clear, the packet is sent over-the-air. If the channel is not clear, the module waits for a randomly selected period of time, then checks again to see if the channel is clear. After a time, the process ends and the data is lost.

RO (Packetization Timeout) Command

<Serial Interfacing> RO command is used to set and read the number of character times of inter-character delay required before transmission.

AT Command: ATRO
Parameter Range:0 – 0xFF [x character times]
Default Parameter Value: 3

RF transmission commences when data is detected in the DI (data in from host) buffer and RO character times of silence are detected on the UART receive lines (after receiving at least 1 byte).

RF transmission will also commence after 100 Bytes (maximum packet size) are received in the DI buffer.

Set the RO parameter to '0' to transmit characters as they arrive instead of buffering them into one RF packet.

RP (RSSI PWM Timer) Command

<I/O Settings {I/O Line Passing}> The RP command is used to enable PWM (Pulse Width Modulation) output on the RF module. The output is calibrated to show the level a received RF signal is above the sensitivity level of the module. The PWM pulses vary from 24 to 100%. Zero percent means PWM output is inactive. One to 24% percent means the received RF signal is at or below the published sensitivity level of the module. The following table shows levels above sensitivity and PWM values.

AT Command: ATRP
Parameter Range:0 – 0xFF [x 100 msec]
Default Parameter Value: 0x28 (40 decimal)

The total period of the PWM output is 64 µs. Because there are 445 steps in the PWM output, the minimum step size is 144 ns.

PWM Percentages

dB above Sensitivity	PWM percentage (high period / total period)
10	41%
20	58%
30	75%

A non-zero value defines the time that the PWM output will be active with the RSSI value of the last received RF packet. After the set time when no RF packets are received, the PWM output will be set low (0 percent PWM) until another RF packet is received. The PWM output will also be set low at power-up until the first RF packet is received. A parameter value of 0xFF permanently enables the PWM output and it will always reflect the value of the last received RF packet.

RR (XBee Retries) Command

<Networking {Addressing}> The RR command is used set/read the maximum number of retries the module will execute in addition to the 3 retries provided by the 802.15.4 MAC. For each XBee retry, the 802.15.4 MAC can execute up to 3 retries.

AT Command: ATRR
Parameter Range: 0 – 6
Default: 0
Minimum Firmware Version Required: 1.xA0

This values does not need to be set on all modules for retries to work. If retries are enabled, the transmitting module will set a bit in the Digi RF Packet header which requests the receiving module to send an ACK (acknowledgement). If the transmitting module does not receive an ACK within 200 msec, it will re-send the packet within a random period up to 48 msec. Each XBee retry can potentially result in the MAC sending the packet 4 times (1 try plus 3 retries). Note that retries are not attempted for packets that are purged when transmitting with a Cyclic Sleep Coordinator.

SC (Scan Channels) Command

<Networking {Association}> The SC command is used to set and read the list of channels to scan for all Active and Energy Scans as a bit field.

This affects scans initiated in command mode [AS (Active Scan) and ED (Energy Scan) commands] and during End Device Association and Coordinator startup.

AT Command: ATSC

Parameter Range: 1–0xFFFF [Bitfield] (bits 0, 14, 15 are not allowed when using the XBee–PRO)

Default Parameter Value: 0x1FFE (all XBee–PRO channels)

Related Commands: ED (Energy Scan), SD (Scan Duration)

Minimum Firmware Version Required: v1.x80

bit 0 - 0x0B	bit 4 - 0x0F	bit 8 - 0x13	bit 12 - 0x17
bit 1 - 0x0C	bit 5 - 0x10	bit 9 - 0x14	bit 13 - 0x18
bit 2 - 0x0D	bit 6 - 0x11	bit 10 - 0x15	bit 14 - 0x19
bit 3 - 0x0E	bit 7 - 0x12	bit 11 - 0x16	bit 15 - 0x1A

SD (Scan Duration) Command

<Networking {Association}> The SD command is used to set and read the exponent value that determines the duration (in time) of a scan.

End Device (Duration of Active Scan during Association) - In a Beacon system, set SD = BE of the Coordinator. SD must be set at least to the highest BE parameter of any Beaconing Coordinator with which an End Device or Coordinator wish to discover.

AT Command: ATSD

Parameter Range: 0 – 0x0F

Default Parameter Value: 4

Related Commands: ED (Energy Scan), SC (Scan Channel)

Minimum Firmware Version Required: v1.x80

Coordinator - If the 'ReassignPANID' option is set on the Coordinator [refer to A2 parameter], the SD parameter determines the length of time the Coordinator will scan channels to locate existing PANs. If the 'ReassignChannel' option is set, SD determines how long the Coordinator will perform an Energy Scan to determine which channel it will operate on.

Scan Time is measured as ((# of Channels to Scan) * (2 ^ SD) * 15.36ms). The number of channels to scan is set by the SC command. The XBee RF Module can scan up to 16 channels (SC = 0xFFFF). The XBee PRO RF Module can scan up to 12 channels (SC = 0x1FFE).

Examples: Values below show results for a 12-channel scan

If SD = 0, time = 0.18 sec	SD = 8, time = 47.19 sec
SD = 2, time = 0.74 sec	SD = 10, time = 3.15 min
SD = 4, time = 2.95 sec	SD = 12, time = 12.58 min
SD = 6, time = 11.80 sec	SD = 14, time = 50.33 min

SH (Serial Number High) Command

<Diagnostics> The SH command is used to read the high 32 bits of the RF module's unique IEEE 64-bit address.

The module serial number is set at the factory and is read-only.

AT Command: ATSH

Parameter Range: 0 – 0xFFFFFFFF [read–only]

Related Commands: SL (Serial Number Low), MY (Source Address)

SL (Serial Number Low) Command

<Diagnostics> The SL command is used to read the low 32 bits of the RF module's unique IEEE 64-bit address.

The module serial number is set at the factory and is read-only.

AT Command: ATSL

Parameter Range: 0 – 0xFFFFFFFF [read–only]

Related Commands: SH (Serial Number High), MY (Source Address)

SM (Sleep Mode) Command

<Sleep Mode (Low Power)> The SM command is used to set and read Sleep Mode settings. By default, Sleep Modes are disabled (SM = 0) and the RF module remains in Idle/ Receive Mode. When in this state, the module is constantly ready to respond to either serial or RF activity.

* The Sleep Coordinator option (SM=6) only exists for backwards compatibility with firmware version 1.x06 only. In all other cases, use the CE command to enable a Coordinator.

AT Command: ATSM

Parameter Range: 0 – 6

Parameter	Configuration
0	Disabled
1	Pin Hibernate
2	Pin Doze
3	(reserved)
4	Cyclic Sleep Remote
5	Cyclic Sleep Remote (with Pin Wake-up)
6	Sleep Coordinator*

Default Parameter Value: 0

SO (Sleep Mode Command)

Sleep (Low Power) Sleep Options Set/Read the sleep mode options.

Bit 0 - Poll wakeup disable

- 0 - Normal operations. A module configured for cyclic sleep will poll for data on waking.
- 1 - Disable wakeup poll. A module configured for cyclic sleep will not poll for data on waking.

Bit 1 - ADC/DIO wakeup sampling disable.

- 0 - Normal operations. A module configured in a sleep mode with ADC/DIO sampling enabled will automatically perform a sampling on wakeup.
- 1 - Suppress sample on wakeup. A module configured in a sleep mode with ADC/DIO sampling enabled will not automatically sample on wakeup.

AT Command: ATSO

Parameter Range:	0–4

Default Parameter Value:

Related Commands: SM (Sleep Mode), ST (Time before Sleep), DP (Disassociation Cyclic Sleep Period, BE (Beacon Order)

SP (Cyclic Sleep Period) Command

<Sleep Mode (Low Power)> The SP command is used to set and read the duration of time in which a remote RF module sleeps. After the cyclic sleep period is over, the module wakes and checks for data. If data is not present, the module goes back to sleep. The maximum sleep period is 268 seconds (SP = 0x68B0).

The SP parameter is only valid if the module is configured to operate in Cyclic Sleep (SM = 4-6). Coordinator and End Device SP values should always be equal.

To send Direct Messages, set SP = 0.

AT Command: ATSP

Parameter Range:	NonBeacon Firmware: 0–0x68B0 [x 10 milliseconds]

Default Parameter Value:

Related Commands: SM (Sleep Mode), ST (Time before Sleep), DP (Disassociation Cyclic Sleep Period, BE (Beacon Order)

NonBeacon Firmware

End Device - SP determines the sleep period for cyclic sleeping remotes. Maximum sleep period is 268 seconds (0x68B0).

Coordinator - If non-zero, SP determines the time to hold an indirect message before discarding it. A Coordinator will discard indirect messages after a period of (2.5 * SP).

ST (Time before Sleep) Command

<Sleep Mode (Low Power)> The ST command is used to set and read the period of inactivity (no serial or RF data is sent or received) before activating Sleep Mode.

AT Command: ATST	
Parameter Range:	NonBeacon Firmware: 1 – 0xFFFF [x 1 millisecond]
Default Parameter Value:	
Related Commands: SM (Sleep Mode), ST (Time before Sleep)	

NonBeacon Firmware

Set/Read time period of inactivity (no serial or RF data is sent or received) before activating Sleep Mode. ST parameter is only valid with Cyclic Sleep settings (SM = 4 - 5).

Coordinator and End Device ST values must be equal.

T0 - T7 ((D0-D7) Output Timeout) Command

<I/O Settings {I/O Line Passing}> The T0, T1, T2, T3, T4, T5, T6 and T7 commands are used to set/read output timeout values for the lines that correspond with the D0 - D7 parameters. When output is set (due to I/O line passing) to a non-default level, a timer is started which when expired, will set the output to its default level. The timer is reset when a valid I/O packet is received. The Tn parameter defines the permissible amount of time to stay in a non-default (active) state. If Tn = 0, Output Timeout is disabled (output levels are held indefinitely).

AT Commands: ATT0 – ATT7
Parameter Range:0 – 0xFF [x 100 msec]
Default Parameter Value:0xFF
Minimum Firmware Version Required: v1.xA0

VL (Firmware Version - Verbose)

<Diagnostics> The VL command is used to read detailed version information about the RF module. The information includes:
application build date; MAC, PHY and bootloader versions; and build dates. This command was removed from firmware 1xC9 and later versions.

AT Command: ATVL
Parameter Range:0 – 0xFF [x 100 milliseconds]
Default Parameter Value: 0x28 (40 decimal)
Minimum Firmware Version Required: v1.x80 – v1.xC8

VR (Firmware Version) Command

<Diagnostics> The VR command is used to read which firmware version is stored in the module.

AT Command: ATVR
Parameter Range: 0 – 0xFFFF [read only]

XBee version numbers will have four significant digits. The reported number will show three or four numbers and is stated in hexadecimal notation. A version can be reported as "ABC" or "ABCD". Digits ABC are the main release number and D is the revision number from the main release. "D" is not required and if it is not present, a zero is assumed for D. "B" is a variant designator. The following variants exist:

- "0" = Non-Beacon Enabled 802.15.4 Code
- "1" = Beacon Enabled 802.15.4 Code

WR (Write) Command

<(Special)> The WR command is used to write configurable parameters to the RF module's non-volatile memory. Parameter values remain in the module's memory until overwritten by subsequent use of the WR Command.

AT Command: ATWR

If changes are made without writing them to non-volatile memory, the module reverts back to previously saved parameters the next time the module is powered-on.

NOTE: Once the WR command is sent to the module, no additional characters should be sent until after the “OK/r” response is received.

API Operation

By default, XBee®/XBee-PRO® RF Modules act as a serial line replacement (Transparent Operation) - all UART data received through the DI pin is queued up for RF transmission. When the module receives an RF packet, the data is sent out the DO pin with no additional information.

Inherent to Transparent Operation are the following behaviors:

- If module parameter registers are to be set or queried, a special operation is required for transitioning the module into Command Mode.
- In point-to-multipoint systems, the application must send extra information so that the receiving module(s) can distinguish between data coming from different remotes.

As an alternative to the default Transparent Operation, API (Application Programming Interface) Operations are available. API operation requires that communication with the module be done through a structured interface (data is communicated in frames in a defined order). The API specifies how commands, command responses and module status messages are sent and received from the module using a UART Data Frame.

API Frame Specifications

Two API modes are supported and both can be enabled using the AP (API Enable) command. Use the following AP parameter values to configure the module to operate in a particular mode:

- AP = 0 (default): Transparent Operation (UART Serial line replacement)
  API modes are disabled.
- AP = 1: API Operation
- AP = 2: API Operation (with escaped characters)

Any data received prior to the start delimiter is silently discarded. If the frame is not received correctly or if the checksum fails, the data is silently discarded.

API Operation (AP parameter = 1)

When this API mode is enabled (AP = 1), the UART data frame structure is defined as follows:

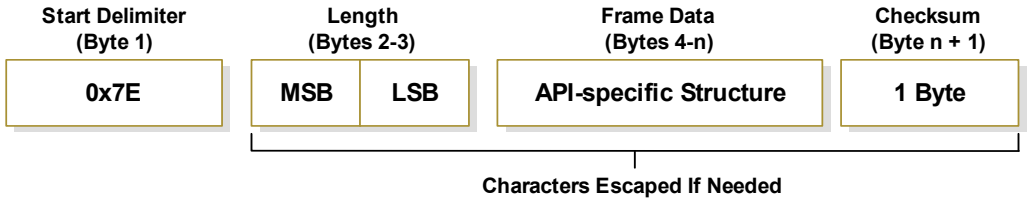
Figure 3-01. UART Data Frame Structure:

MSB = Most Significant Byte, LSB = Least Significant Byte

API Operation - with Escape Characters (AP parameter = 2)

When this API mode is enabled (AP = 2), the UART data frame structure is defined as follows:

Figure 3-02. UART Data Frame Structure - with escape control characters:



MSB = Most Significant Byte, LSB = Least Significant Byte

Escape characters. When sending or receiving a UART data frame, specific data values must be escaped (flagged) so they do not interfere with the UART or UART data frame operation. To escape an interfering data byte, insert 0x7D and follow it with the byte to be escaped XOR'd with 0x20.

Data bytes that need to be escaped:

- 0x7E – Frame Delimiter
- 0x7D – Escape
- 0x11 – XON
- 0x13 – XOFF

Example - Raw UART Data Frame (before escaping interfering bytes):
0x7E 0x00 0x02 0x23 0x11 0xCB

0x11 needs to be escaped which results in the following frame:
0x7E 0x00 0x02 0x23 0x7D 0x31 0xCB

Note: In the above example, the length of the raw data (excluding the checksum) is 0x0002 and the checksum of the non-escaped data (excluding frame delimiter and length) is calculated as: 0xFF - (0x23 + 0x11) = (0xFF - 0x34) = 0xCB.

Checksum

To test data integrity, a checksum is calculated and verified on non-escaped data.

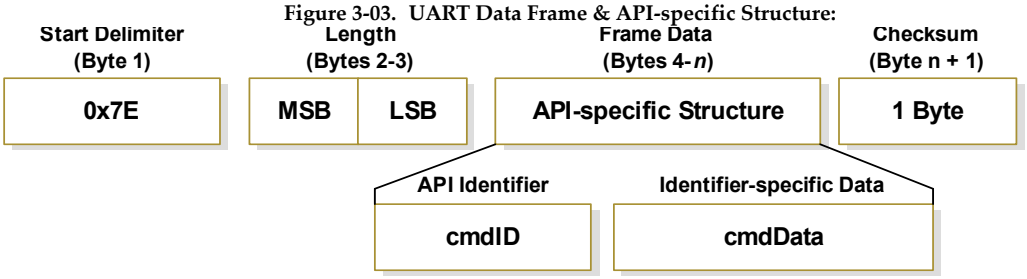
To calculate: Not including frame delimiters and length, add all bytes keeping only the lowest 8 bits of the result and subtract from 0xFF.

To verify: Add all bytes (include checksum, but not the delimiter and length). If the checksum is correct, the sum will equal 0xFF.

API Types

Frame data of the UART data frame forms an API-specific structure as follows:

Figure 3-03. UART Data Frame & API-specific Structure:

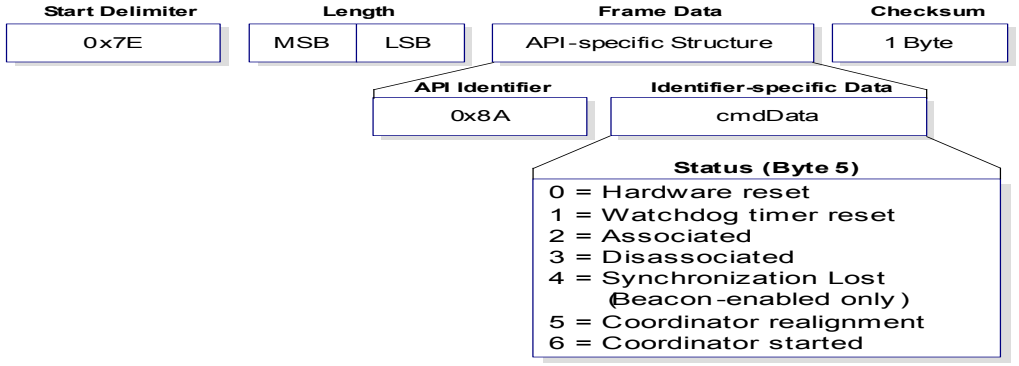


The cmdID frame (API-identifier) indicates which API messages will be contained in the cmdData frame (Identifier-specific data). Refer to the sections that follow for more information regarding the supported API types. Note that multi-byte values are sent big endian.

Modem Status

API Identifier: 0x8A
RF module status messages are sent from the module in response to specific conditions.

Figure 3-04. Modem Status Frames



AT Command

API Identifier Value: 0x08
The "AT Command" API type allows for module parameters to be queried or set. When using this command ID, new parameter values are applied immediately. This includes any register set with the "AT Command - Queue Parameter Value" (0x09) API type.

Figure 3-05. AT Command Frames

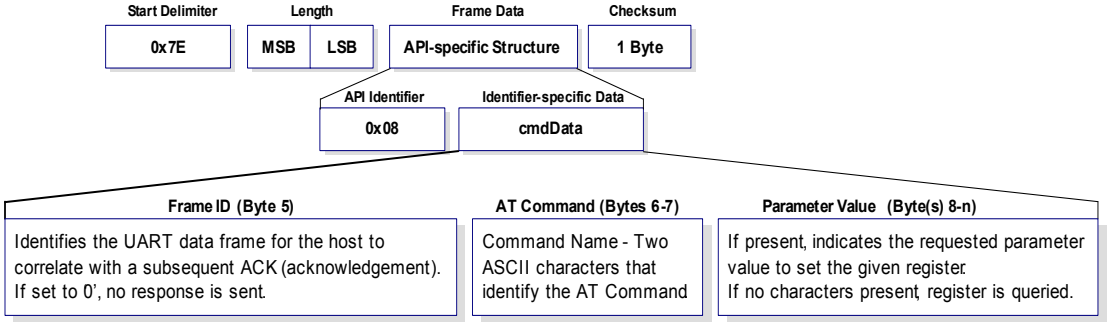
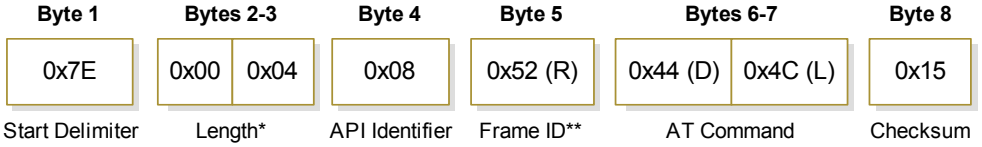


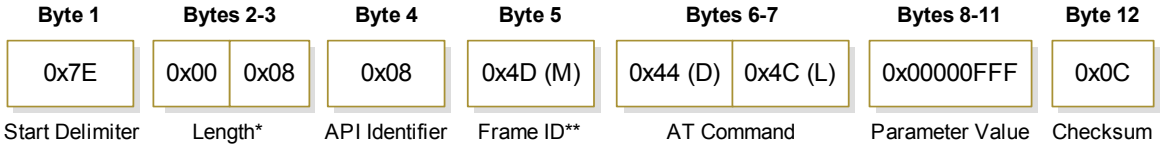
Figure 3-06. Example: API frames when reading the DL parameter value of the module.



** Length [Bytes] = API Identifier + Frame ID + AT Command*

*** "R" value was arbitrarily selected.*

Figure 3-07. Example: API frames when modifying the DL parameter value of the module.



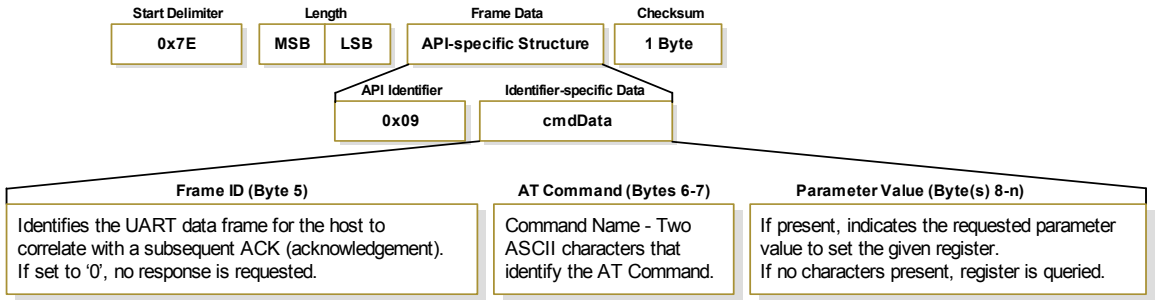
** Length [Bytes] = API Identifier + Frame ID + AT Command + Parameter Value*

*** "M" value was arbitrarily selected.*

AT Command - Queue Parameter Value

API Identifier Value: 0x09
This API type allows module parameters to be queried or set. In contrast to the "AT Command" API type, new parameter values are queued and not applied until either the "AT Command" (0x08) API type or the AC (Apply Changes) command is issued. Register queries (reading parameter values) are returned immediately.

Figure 3-08. AT Command Frames
(Note that frames are identical to the "AT Command" API type except for the API identifier.)



AT Command Response

API Identifier Value: 0x88
Response to previous command.

In response to an AT Command message, the module will send an AT Command Response message. Some commands will send back multiple frames (for example, the ND (Node Discover) and AS (Active Scan) commands). These commands will end by sending a frame with a status of ATCMD_OK and no cmdData.

Figure 3-09. AT Command Response Frames.

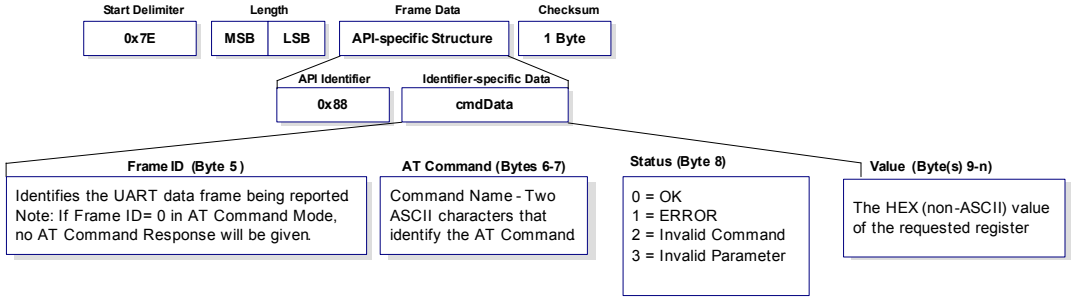
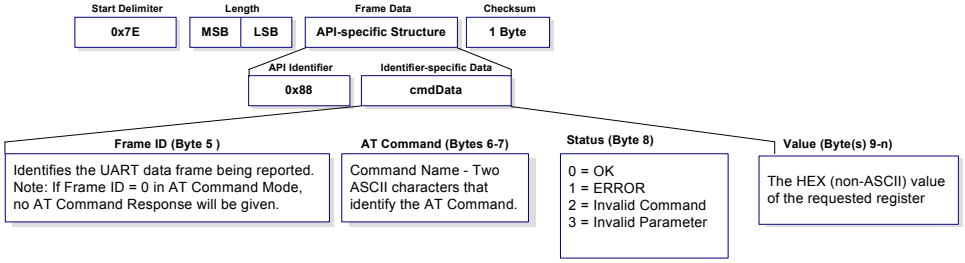


Figure 3-10. AT Command Response Frames.

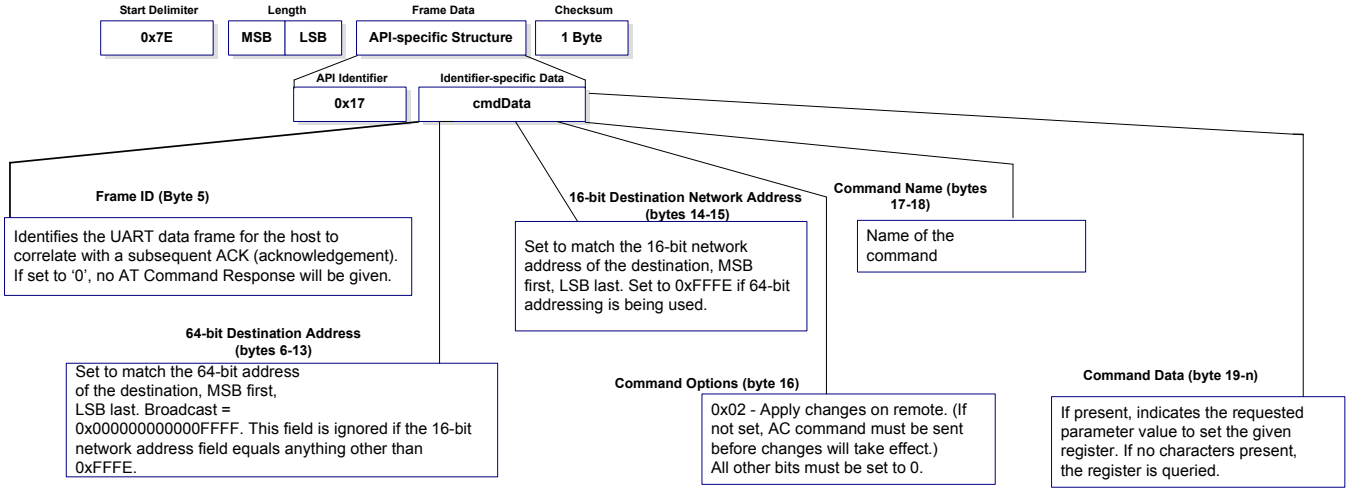


Remote AT Command Request

API Identifier Value: 0x17

Allows for module parameter registers on a remote device to be queried or set

Figure 3-11. Remote AT Command Request

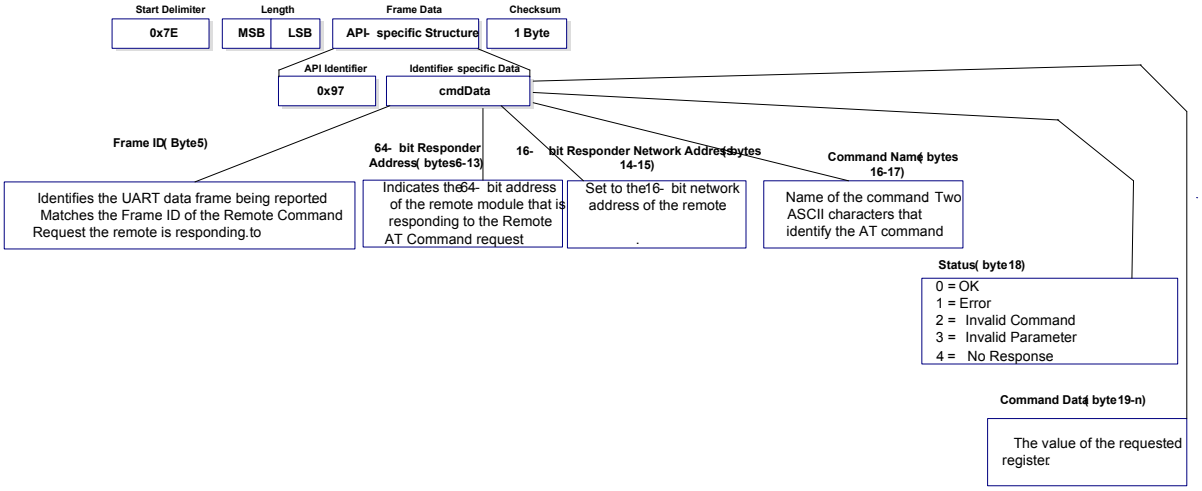


Remote Command Response

API Identifier Value: 0x97

If a module receives a remote command response RF data frame in response to a Remote AT Command Request, the module will send a Remote AT Command Response message out the UART. Some commands may send back multiple frames--for example, Node Discover (ND) command.

Figure 3-12. Remote AT Command Response.

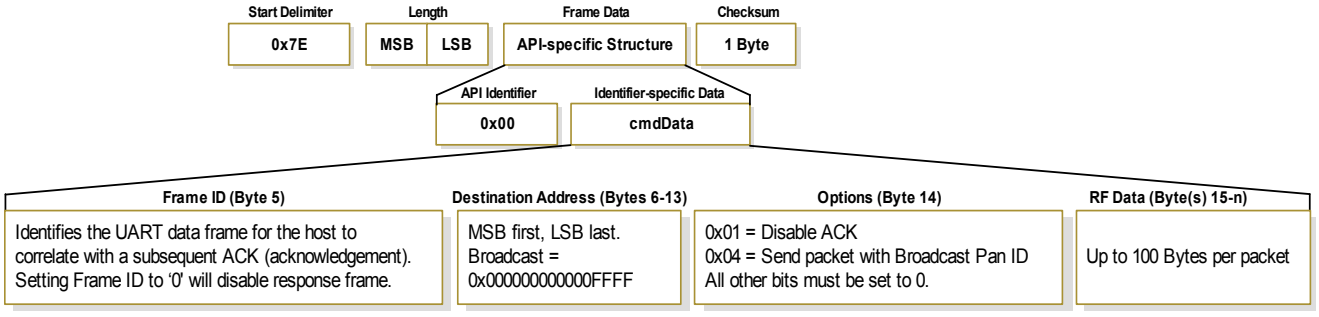


TX (Transmit) Request: 64-bit address

API Identifier Value: 0x00

A TX Request message will cause the module to send RF Data as an RF Packet.

Figure 3-13. TX Packet (64-bit address) Frames

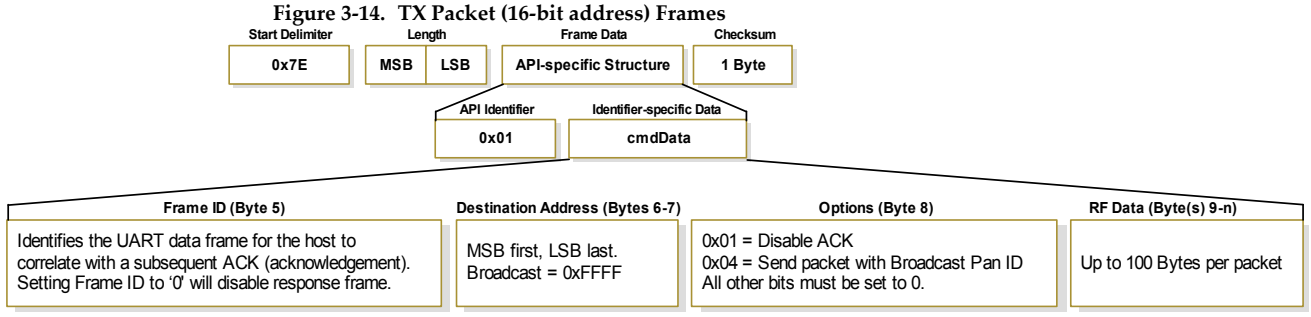


TX (Transmit) Request: 16-bit address

API Identifier Value: 0x01

A TX Request message will cause the module to send RF Data as an RF Packet.

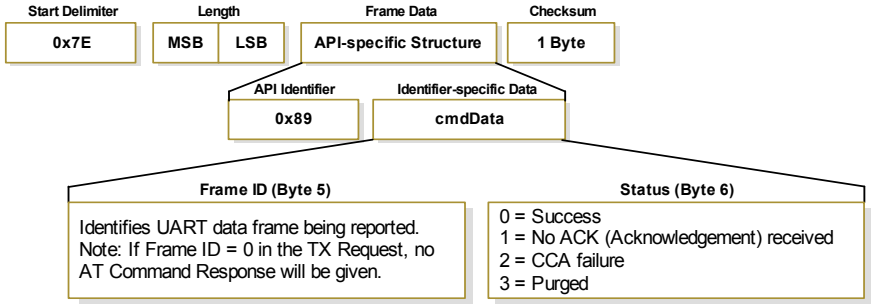
Figure 3-14. TX Packet (16-bit address) Frames



TX (Transmit) Status

API Identifier Value: 0x89
When a TX Request is completed, the module sends a TX Status message. This message will indicate if the packet was transmitted successfully or if there was a failure.

Figure 3-15. TX Status Frames



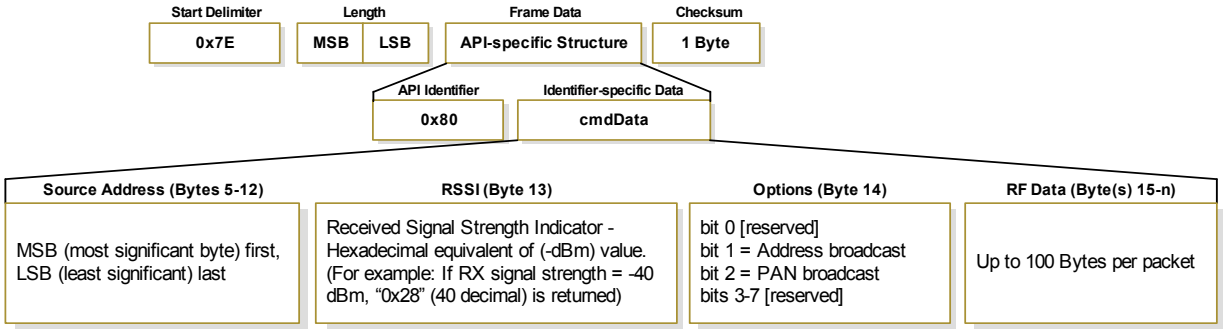
NOTES:

- "STATUS = 1" occurs when all retries are expired and no ACK is received.
- If transmitter broadcasts (destination address = 0x000000000000FFFF), only "STATUS = 0 or 2" will be returned.
- "STATUS = 3" occurs when Coordinator times out of an indirect transmission. Timeout is defined as (2.5 x SP (Cyclic Sleep Period) parameter value).

RX (Receive) Packet: 64-bit Address

API Identifier Value: 0x80
When the module receives an RF packet, it is sent out the UART using this message type.

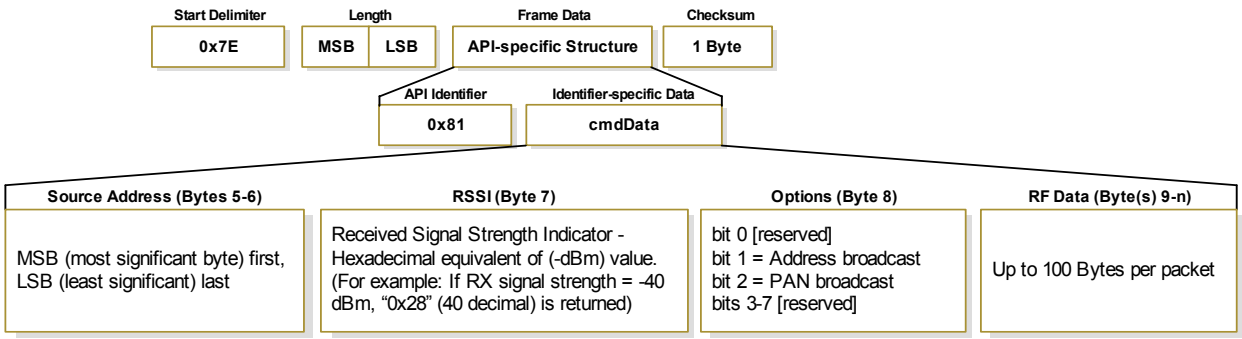
Figure 3-16. RX Packet (64-bit address) Frames



RX (Receive) Packet: 16-bit Address

API Identifier Value: 0x81
When the module receives an RF packet, it is sent out the UART using this message type.

Figure 3-17. RX Packet (16-bit address) Frames



Appendix A: Agency Certifications

United States (FCC)

XBee®/XBee-PRO® RF Modules comply with Part 15 of the FCC rules and regulations. Compliance with the labeling requirements, FCC notices and antenna usage guidelines is required.

To fulfill FCC Certification requirements, the OEM must comply with the following regulations:

1. The system integrator must ensure that the text on the external label provided with this device is placed on the outside of the final product [Figure A-01].
2. XBee®/XBee-PRO® RF Modules may only be used with antennas that have been tested and approved for use with this module [refer to the antenna tables in this section].

OEM Labeling Requirements

WARNING: The Original Equipment Manufacturer (OEM) must ensure that FCC labeling requirements are met. This includes a clearly visible label on the outside of the final product enclosure that displays the contents shown in the figure below.

Figure 4-01. Required FCC Label for OEM products containing the XBee®/XBee-PRO® RF Module

Contains FCC ID: OUR-XBEE/OUR-XBEEPRO**

The enclosed device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (*i.*) this device may not cause harmful interference and (*ii.*) this device must accept any interference received, including interference that may cause undesired operation.

* The FCC ID for the XBee is "OUR-XBEE". The FCC ID for the XBee-PRO is "OUR-XBEEPRO".

FCC Notices

IMPORTANT: The XBee®/XBee-PRO® RF Module has been certified by the FCC for use with other products without any further certification (as per FCC section 2.1091). Modifications not expressly approved by Digi could void the user's authority to operate the equipment.

IMPORTANT: OEMs must test final product to comply with unintentional radiators (FCC section 15.107 & 15.109) before declaring compliance of their final product to Part 15 of the FCC Rules.

IMPORTANT: The RF module has been certified for remote and base radio applications. If the module will be used for portable applications, the device must undergo SAR testing.

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation.

If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures: Re-orient or relocate the receiving antenna, Increase the separation between the equipment and receiver, Connect equipment and receiver to outlets on different circuits, or Consult the dealer or an experienced radio/TV technician for help.

FCC-Approved Antennas (2.4 GHz)

XBee/XBee-PRO RF Modules can be installed using antennas and cables constructed with standard connectors (Type-N, SMA, TNC, etc.) if the installation is performed professionally and according to FCC guidelines. For installations not performed by a professional, non-standard connectors (RPSMA, RPTNC, etc) must be used.

The modules are FCC-approved for fixed base station and mobile applications on channels 0x0B - 0x1A (XBee) and 0x0C - 0x17 (XBee-PRO). If the antenna is mounted at least 20cm (8 in.) from nearby persons, the application is considered a mobile application. Antennas not listed in the table must be tested to comply with FCC Section 15.203 (Unique Antenna Connectors) and Section 15.247 (Emissions).

XBee RF Modules (1 mW): XBee Modules have been tested and approved for use with all of the antennas listed in the tables below (Cable-loss IS NOT required).

XBee-PRO RF Modules (60 mW): XBee-PRO Modules have been tested and approved for use with the antennas listed in the tables below (Cable-loss IS required when using antennas listed in the second table.

The antennas in the tables below have been approved for use with this module. Digi does not carry all of these antenna variants. Contact Digi Sales for available antennas.

Antennas approved for use with the XBee®/XBee-PRO® RF Modules (Cable-loss is not required.)

Part Number	Type (Description)	Gain	Application*	Min. Separation
A24-HASM-450	Dipole (Half-wave articulated RPSMA - 4.5")	2.1 dBi	Fixed/Mobile	20 cm
A24-HABSM	Dipole (Articulated RPSMA)	2.1 dBi	Fixed	20 cm
A24-HABUF-P5I	Dipole (Half-wave articulated bulkhead mount U.FL. w/ 5" pigtail)	2.1 dBi	Fixed	20 cm
A24-HASM-525	Dipole (Half-wave articulated RPSMA - 5.25")	2.1 dBi	Fixed/Mobile	20 cm
A24-QI	Monopole (Integrated whip)	1.5 dBi	Fixed	20 cm

Antennas approved for use with the XBee RF Modules (Cable-loss is required)

Part Number	Type (Description)	Gain	Application*	Min. Separation	Required Cable-loss
Omni-Directional Class Antennas					
A24-Y6NF	Yagi (6-element)	8.8 dBi	Fixed	2 m	1.7 dB
A24-Y7NF	Yagi (7-element)	9.0 dBi	Fixed	2 m	1.9 dB
A24-Y9NF	Yagi (9-element)	10.0 dBi	Fixed	2 m	2.9 dB
A24-Y10NF	Yagi (10-element)	11.0 dBi	Fixed	2 m	3.9 dB
A24-Y12NF	Yagi (12-element)	12.0 dBi	Fixed	2 m	4.9 dB
A24-Y13NF	Yagi (13-element)	12.0 dBi	Fixed	2 m	4.9 dB
A24-Y15NF	Yagi (15-element)	12.5 dBi	Fixed	2 m	5.4 dB
A24-Y16NF	Yagi (16-element)	13.5 dBi	Fixed	2 m	6.4 dB
A24-Y16RM	Yagi (16-element, RPSMA connector)	13.5 dBi	Fixed	2 m	6.4 dB
A24-Y18NF	Yagi (18-element)	15.0 dBi	Fixed	2 m	7.9 dB
Omni-Directional Class Antennas					
A24-C1	Surface Mount	-1.5 dBi	Fixed/Mobile	20 cm	-
A24-F2NF	Omni-directional (Fiberglass base station)	2.1 dBi	Fixed/Mobile	20 cm	
A24-F3NF	Omni-directional (Fiberglass base station)	3.0 dBi	Fixed/Mobile	20 cm	
A24-F5NF	Omni-directional (Fiberglass base station)	5.0 dBi	Fixed/Mobile	20 cm	
A24-F8NF	Omni-directional (Fiberglass base station)	8.0 dBi	Fixed	2 m	
A24-F9NF	Omni-directional (Fiberglass base station)	9.5 dBi	Fixed	2 m	0.2 dB
A24-F10NF	Omni-directional (Fiberglass base station)	10.0 dBi	Fixed	2 m	0.7 dB
A24-F12NF	Omni-directional (Fiberglass base station)	12.0 dBi	Fixed	2 m	2.7 dB
A24-F15NF	Omni-directional (Fiberglass base station)	15.0 dBi	Fixed	2 m	5.7 dB
A24-W7NF	Omni-directional (Base station)	7.2 dBi	Fixed	2 m	
A24-M7NF	Omni-directional (Mag-mount base station)	7.2 dBi	Fixed	2 m	
Panel Class Antennas					
A24-P8SF	Flat Panel	8.5 dBi	Fixed	2 m	1.5 dB
A24-P8NF	Flat Panel	8.5 dBi	Fixed	2 m	1.5 dB
A24-P13NF	Flat Panel	13.0 dBi	Fixed	2 m	6 dB
A24-P14NF	Flat Panel	14.0 dBi	Fixed	2 m	7 dB
A24-P15NF	Flat Panel	15.0 dBi	Fixed	2 m	8 dB
A24-P16NF	Flat Panel	16.0 dBi	Fixed	2 m	9 dB

Antennas approved for use with the XBee®/XBee-PRO® RF Modules (Cable-loss is required)

Part Number	Type (Description)	Gain	Application*	Min. Separation	Required Cable-loss
A24-C1	Surface Mount	-1.5 dBi	Fixed/Mobile	20 cm	-
A24-Y4NF	Yagi (4-element)	6.0 dBi	Fixed	2 m	8.1 dB
A24-Y6NF	Yagi (6-element)	8.8 dBi	Fixed	2 m	10.9 dB
A24-Y7NF	Yagi (7-element)	9.0 dBi	Fixed	2 m	11.1 dB
A24-Y9NF	Yagi (9-element)	10.0 dBi	Fixed	2 m	12.1 dB
A24-Y10NF	Yagi (10-element)	11.0 dBi	Fixed	2 m	13.1 dB
A24-Y12NF	Yagi (12-element)	12.0 dBi	Fixed	2 m	14.1 dB
A24-Y13NF	Yagi (13-element)	12.0 dBi	Fixed	2 m	14.1 dB
A24-Y15NF	Yagi (15-element)	12.5 dBi	Fixed	2 m	14.6 dB
A24-Y16NF	Yagi (16-element)	13.5 dBi	Fixed	2 m	15.6 dB
A24-Y16RM	Yagi (16-element, RPSMA connector)	13.5 dBi	Fixed	2 m	15.6 dB
A24-Y18NF	Yagi (18-element)	15.0 dBi	Fixed	2 m	17.1 dB
A24-F2NF	Omni-directional (Fiberglass base station)	2.1 dBi	Fixed/Mobile	20 cm	4.2 dB
A24-F3NF	Omni-directional (Fiberglass base station)	3.0 dBi	Fixed/Mobile	20 cm	5.1 dB
A24-F5NF	Omni-directional (Fiberglass base station)	5.0 dBi	Fixed/Mobile	20 cm	7.1 dB
A24-F8NF	Omni-directional (Fiberglass base station)	8.0 dBi	Fixed	2 m	10.1 dB
A24-F9NF	Omni-directional (Fiberglass base station)	9.5 dBi	Fixed	2 m	11.6 dB
A24-F10NF	Omni-directional (Fiberglass base station)	10.0 dBi	Fixed	2 m	12.1 dB
A24-F12NF	Omni-directional (Fiberglass base station)	12.0 dBi	Fixed	2 m	14.1 dB
A24-F15NF	Omni-directional (Fiberglass base station)	15.0 dBi	Fixed	2 m	17.1 dB
A24-W7NF	Omni-directional (Base station)	7.2 dBi	Fixed	2 m	9.3 dB
A24-M7NF	Omni-directional (Mag-mount base station)	7.2 dBi	Fixed	2 m	9.3 dB
A24-P8SF	Flat Panel	8.5 dBi	Fixed	2 m	8.6 dB
A24-P8NF	Flat Panel	8.5 dBi	Fixed	2 m	8.6 dB
A24-P13NF	Flat Panel	13.0 dBi	Fixed	2 m	13.1 dB
A24-P14NF	Flat Panel	14.0 dBi	Fixed	2 m	14.1 dB
A24-P15NF	Flat Panel	15.0 dBi	Fixed	2 m	15.1 dB
A24-P16NF	Flat Panel	16.0 dBi	Fixed	2 m	16.1 dB
A24-P19NF	Flat Panel	19.0 dBi	Fixed	2 m	19.1 dB

*** If using the RF module in a portable application** (For example - If the module is used in a handheld device and the antenna is less than 20cm from the human body when the device is operation): The integrator is responsible for passing additional SAR (Specific Absorption Rate) testing based on FCC rules 2.1091 and FCC Guidelines for Human Exposure to Radio Frequency Electromagnetic Fields, OET Bulletin and Supplement C. The testing results will be submitted to the FCC for approval prior to selling the integrated unit. The required SAR testing measures emissions from the module and how they affect the person.

RF Exposure

WARNING: To satisfy FCC RF exposure requirements for mobile transmitting devices, a separation distance of 20 cm or more should be maintained between the antenna of this device and persons during device operation. To ensure compliance, operations at closer than this distance is not recommended. The antenna used for this transmitter must not be co-located in conjunction with any other antenna or transmitter.

The preceding statement must be included as a CAUTION statement in OEM product manuals in order to alert users of FCC RF Exposure compliance.

Europe (ETSI)

The XBee RF Modules have been certified for use in several European countries. For a complete list, refer to www.digi.com

If the XBee RF Modules are incorporated into a product, the manufacturer must ensure compliance of the final product to the European harmonized EMC and low-voltage/safety standards. A Declaration of Conformity must be issued for each of these standards and kept on file as described in Annex II of the R&TTE Directive.

Furthermore, the manufacturer must maintain a copy of the XBee user manual documentation and ensure the final product does not exceed the specified power ratings, antenna specifications, and/or installation requirements as specified in the user manual. If any of these specifications are exceeded in the final product, a submission must be made to a notified body for compliance testing to all required standards.

OEM Labeling Requirements

The 'CE' marking must be affixed to a visible location on the OEM product.

CE Labeling Requirements

The CE mark shall consist of the initials "CE" taking the following form:

- If the CE marking is reduced or enlarged, the proportions given in the above graduated drawing must be respected.
- The CE marking must have a height of at least 5mm except where this is not possible on account of the nature of the apparatus.
- The CE marking must be affixed visibly, legibly, and indelibly.

Restrictions

Power Output: When operating in Europe, XBee-PRO 802.15.4 modules must operate at or below a transmit power output level of 10dBm. Customers have two choices for transmitting at or below 10dBm:

a. Order the standard XBee-PRO module and change the PL command to 0 (10dBm)

b. Order the International variant of the XBee-PRO module, which has a maximum transmit output power of 10dBm (@ PL=4).

Additionally, European regulations stipulate an EIRP power maximum of 12.86 dBm (19 mW) for the XBee-PRO and 12.11 dBm for the XBee when integrating antennas.

France: Outdoor use limited to 10 mW EIRP within the band 2454-2483.5 MHz.

Norway: Norway prohibits operation near Ny-Alesund in Svalbard. More information can be found at the Norway Posts and Telecommunications site (www.npt.no).

Declarations of Conformity

Digi has issued Declarations of Conformity for the XBee RF Modules concerning emissions, EMC and safety. Files can be obtained by contacting Digi Support.

Important Note:

Digi does not list the entire set of standards that must be met for each country. Digi customers assume full responsibility for learning and meeting the required guidelines for each country in their distribution market. For more information relating to European compliance of an OEM product incorporating the XBee RF Module, contact Digi, or refer to the following web sites:

CEPT ERC 70-03E - Technical Requirements, European restrictions and general requirements: Available at www.ero.dk/.

R&TTE Directive - Equipment requirements, placement on market: Available at www.ero.dk/.

Approved Antennas

When integrating high-gain antennas, European regulations stipulate EIRP power maximums. Use the following guidelines to determine which antennas to design into an application.

XBee-PRO RF Module

The following antenna types have been tested and approved for use with the XBee Module:

Antenna Type: Yagi
RF module was tested and approved with 15 dBi antenna gain with 1 dB cable-loss (EIRP Maximum of 14 dBm). Any Yagi type antenna with 14 dBi gain or less can be used with no cable-loss.

Antenna Type: Omni-directional
RF module was tested and approved with 15 dBi antenna gain with 1 dB cable-loss (EIRP Maxi-

mum of 14 dBm). Any Omni-directional type antenna with 14 dBi gain or less can be used with no cable-loss.

Antenna Type: Flat Panel
RF module was tested and approved with 19 dBi antenna gain with 4.8 dB cable-loss (EIRP Maximum of 14.2 dBm). Any Flat Panel type antenna with 14.2 dBi gain or less can be used with no cable-loss.

XBee-PRO RF Module (@ 10 dBm Transmit Power, PL parameter value must equal 0, or use International variant)

The following antennas have been tested and approved for use with the embedded XBee-PRO RF Module:

- Dipole (2.1 dBi, Omni-directional, Articulated RPSMA, Digi part number A24-HABSM)
- Chip Antenna (-1.5 dBi)
- Attached Monopole Whip (1.5 dBi)

The RF modem encasement was designed to accommodate the RPSMA antenna option.

Canada (IC)

Labeling Requirements

Labeling requirements for Industry Canada are similar to those of the FCC. A clearly visible label on the outside of the final product enclosure must display the following text:

Contains Model XBee Radio, IC: 4214A-XBEE
Contains Model XBee-PRO Radio, IC: 4214A-XBEEPRO

The integrator is responsible for its product to comply with IC ICES-003 & FCC Part 15, Sub. B - Unintentional Radiators. ICES-003 is the same as FCC Part 15 Sub. B and Industry Canada accepts FCC test report or CISPR 22 test report for compliance with ICES-003.

Japan

In order to gain approval for use in Japan, the XBee RF module or the International variant of the XBee-PRO RF module (which has 10 dBm transmit output power) must be used.

Labeling Requirements

A clearly visible label on the outside of the final product enclosure must display the following text:

ID: 005NYCA0378

Appendix B. Additional Information

1-Year Warranty

XBee®/XBee-PRO® RF Modules from Digi Intenational, Inc. (the "Product") are warranted against defects in materials and workmanship under normal use, for a period of 1-year from the date of purchase. In the event of a product failure due to materials or workmanship, Digi will repair or replace the defective product. For warranty service, return the defective product to Digi, shipping prepaid, for prompt repair or replacement.

The foregoing sets forth the full extent of Digi's warranties regarding the Product. Repair or replacement at Digi's option is the exclusive remedy. THIS WARRANTY IS GIVEN IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, AND DIGI SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL DIGI, ITS SUPPLIERS OR LICENSORS BE LIABLE FOR DAMAGES IN EXCESS OF THE PURCHASE PRICE OF THE PRODUCT, FOR ANY LOSS OF USE, LOSS OF TIME, INCONVENIENCE, COMMERCIAL LOSS, LOST PROFITS OR SAVINGS, OR OTHER INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PRODUCT, TO THE FULL EXTENT SUCH MAY BE DISCLAIMED BY LAW. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES. THEREFORE, THE FOREGOING EXCLUSIONS MAY NOT APPLY IN ALL CASES. This warranty provides specific legal rights. Other rights which vary from state to state may also apply.

Web Site: www.parallax.com
Forums: forums.parallax.com
Sales: sales@parallax.com
Technical: support@parallax.com

Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267

XBee SIP Adapter (#32402)

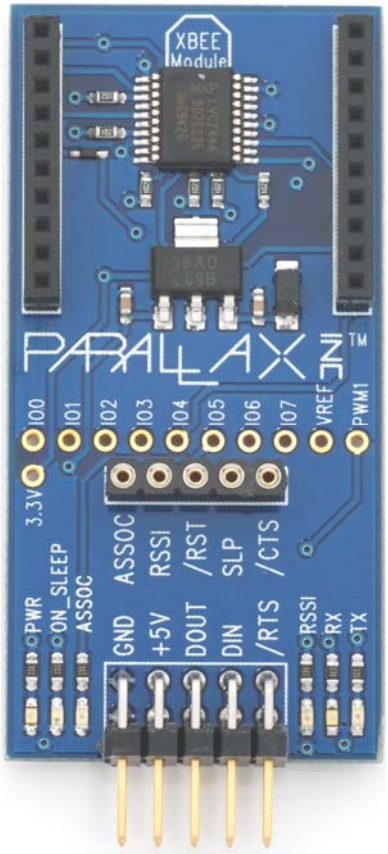
The XBee SIP Adapter comes fully assembled and provides a small-footprint solution for interfacing your microcontroller to any XBee or XBee-Pro module. A 3.3 volt regulator and 74LVC244A buffer on board provide safe interfacing to a 5 volt supply and easy compatibility with any Parallax microcontroller.

The 2 x 5 dual SIP header makes a sturdy connection to your breadboard or through-hole board, and brings the basic connections to your prototyping area. The more advanced XBee features are still accessible, through an additional header and plated through-holes on the board.

Two very simple example applications that are compatible with all BASIC Stamp 2 models are included in this document.

Features

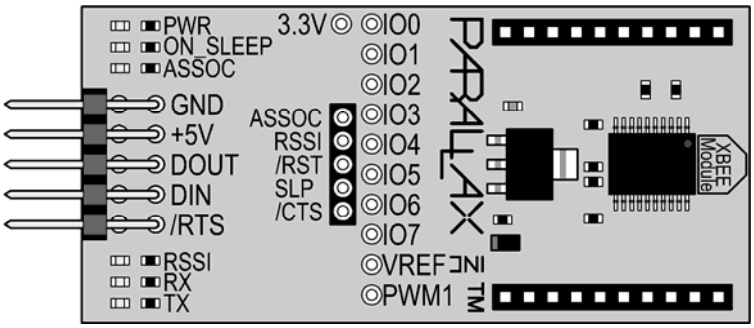
- Onboard 3.3 V regulator
- 5 V to 3.3 V logic translator buffers common I/O pins
- Six status indicator LEDs for Power, Tx, Rx, RSSI, Associate and mode (Sleep/ON)
- Small footprint dual SIP header provides support and allows easy interfacing to DOUT (TX), DIN (RX), RTS, 5 V supply and ground
- 5-pin female header connections provides interfacing to other XBee pins such as sleep, reset and associate
- A row of 10 plated through-holes with 01" spacing allows the option of soldering jumper wires or a header (not included) for access to the remaining XBee pins in advanced applications
- An additional plated through-hole gives access to 3.3 V output for ADC reference (VREF) when required
- Adapter board is pre-assembled—no soldering is required for using most common XBee features
- Compatible with all Parallax microcontrollers, including the 5 V BASIC Stamp modules and 3.3 V Propeller P8X32A



Key Specifications

- Power Requirements5 VDC
- Communication: Serial pass-through to XBee module
- Operating temperature: -40 to + 158 °F (-40 to +70 °C)
- Dimensions: 1 x 2.4 x 0.36 in (25.4 x 61 x 9.14 mm)

Pin Definitions



5-Pin Dual Row Male Header

Pin	Name	Type	Function
1	GND	G	Ground
2	+5V	P	5 V supply
3	DOUT	O	Serial Data output from XBee
4	DIN	I	Serial Data input to XBee
5	/RTS	I	Ready-to-Send input to XBee for flow control when configured

Pin Type: P = Power, G = Ground, I = Input, O = Output

5-Pin Female Header

Pin	Name	Type	Function
1	ASSOC	O	Associate indication output
2	RSSI	O	Received Signal Strength Indicator PWM output
3	/RST	I	Low-level reset to XBee
4	SLP	I	Pin-Sleep input to XBee when configured
5	/CTS	O	Clear-to-Send from XBee when configured

Pin Type: P = Power, G = Ground, I = Input, O = Output

11 Plated Through-holes

CAUTION: These connections are NOT buffered. Connections to 5 V may damage the XBee module. Use of these through-holes to access XBee pins is recommended for advanced users only.

Pin	Name	Type	Function
1	IO0	I/O	AD0/DIO0 – Analog input and digital I/O
2	IO1	I/O	AD1/DIO0 – Analog input and digital I/O
3	IO2	I/O	AD2/DIO0 – Analog input and digital I/O
4	IO3	I/O	AD3/DIO0 – Analog input and digital I/O
5	IO4	I/O	Associate Indicator, AD0/DIO0 – Analog input and digital I/O
6	IO5	I/O	AD5/DIO0 – Analog input and digital I/O
7	IO6	I/O	/RTS, AD6/DIO0 – Ready to send, Analog input and digital I/O
8	IO7	I/O	AD7/DIO0 – Analog input and digital I/O
9	VREF	I	Analog reference input
10	PWM1	O	PWM output
11	3.3V	P	3.3 V Output for ADC reference (VREF) when required

Pin Type: P = Power, G = Ground, I = Input, O = Output

LED Indicators

LED Name	Indication
PWR	Indicates power available, supplied from 3.3 V on-board regulator
ON_SLEEP	Indicates sleep status of XBee, On = awake
ASSOC	Indicates status when using association feature of XBee to join networks. Solid = Associated, Blinking = Not Associated. Most users manually setup the network instead of associating.
RSSI	Receive Signal Strength Indicator. PWM controlled. Lights for 5 seconds following reception of RF data. A dimming may be noticeable with poor signal strength.
RX	Indicates data received by BASIC Stamp from the XBee (DOUT pin) due to received RF data or command communications with XBee.
TX	Indicates data sent to the XBee (DIN pin) from the BASIC Stamp for RF transmission or command communications.

I/O Pin Buffering

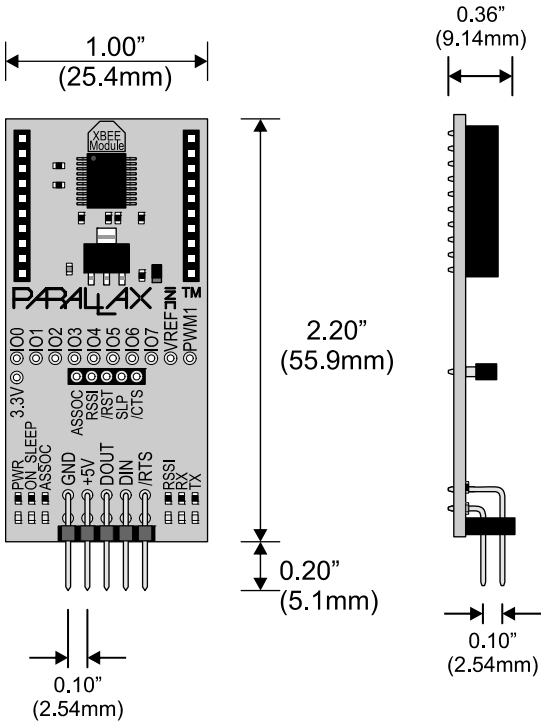
XBee RF Modules are 3.3 V devices. The XBee SIP Adapter provides a means to interface the most frequently used XBee functions with a BASIC Stamp or other 5 V microcontrollers. All I/O on the male and female 5-pin headers are buffered; inputs to the XBee are buffered to translate 5 V to 3.3 V, while outputs from the XBee are buffered to protect the XBee I/O.

CAUTION: the 11 plated through-holes are NOT buffered. Connections to 5 V using these through-holes may damage the XBee. Use of these through-holes to access XBee pins is recommended for advanced users only. For more information on XBee configuration and I/O uses, please consult Digi International's XBee manual.

Communication Protocol

Communications interfacing between the BASIC Stamp and XBee uses non-inverted asynchronous serial data at 9600 bps, 8-N-1. The baud rate of the XBee is configurable in Command Mode. Please see the example programs and the Digi's XBee documentation for more information.

Module Dimensions

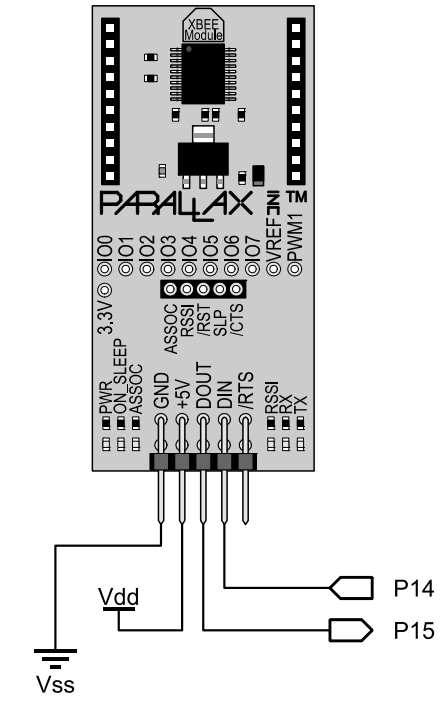


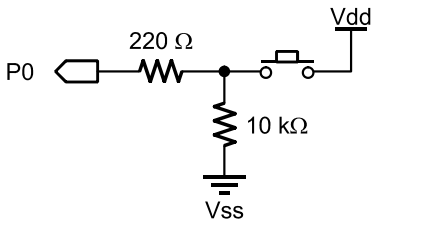
BASIC Stamp® Example Applications

The following two simple example applications use two BASIC Stamp 2 microcontroller modules, two XBee SIP Adapters, and two XBee modules. You may use any BASIC Stamp 2 series modules; since the example code uses conditional compilation it is not necessary to use two of the same model.

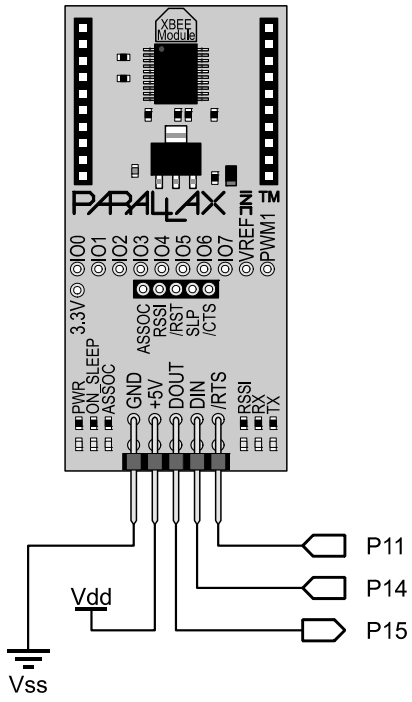
It IS necessary to use two XBee modules from the same series; Series 1 and Series 2 XBee modules are not cross-compatible.

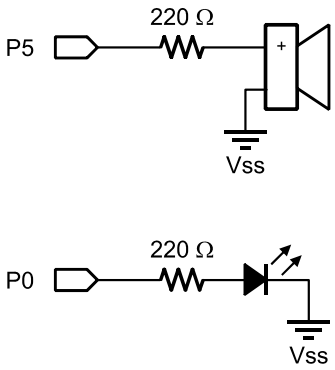
Connection Diagrams





Transmitter Configuration





Receiver Configuration

Example 1: Single Byte Transmission

This example illustrates sending a single byte between BASIC Stamp modules. The transmit code (Simple_Byte_Tx.bs2) will send the value of the pushbutton as a byte. The receiver (Simple_Byte_Rx.bs2) will accept the byte. If byte value is 1, the LED will light and sound buzzer. If 0, the LED will turn off and not sound the buzzer.

Transmitter Code

```
' ************************************************
' Simple Byte Tx.bs2
' Sends the state of pushbutton every 250mSec
' ************************************************

' {$STAMP BS2}
' {$PBASIC 2.5}

#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T9600       CON     84
  #CASE BS2SX, BS2P
    T9600       CON     240
  #CASE BS2PX
    T9600       CON     396
#ENDSELECT


' ***************** Variables, Constants and Pins

Baud            CON     T9600   ' Set baud rate

Rx              CON     15      ' XBee DOUT
Tx              CON     14      ' XBee DIN

PB              PIN     0       ' Pushbutton

State           VAR     Bit


' ***************** Main Loop

DO
 State = PB                     ' Read pushbutton
 SEROUT Tx, Baud,[State]        ' Send pushbutton value as byte
 PAUSE 250                      ' short delay
LOOP
```

Receiver Code

```
' ************************************************
' Simple Byte Rx.bs2
' Receives byte value (0/1) to control LED and buzzer
' ************************************************

' {$STAMP BS2}
' {$PBASIC 2.5}

#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T9600       CON     84
  #CASE BS2SX, BS2P
    T9600       CON     240
  #CASE BS2PX
    T9600       CON     396
#ENDSELECT
```

```
' ***************** Variables, Constants and Pins

Baud            CON     T9600 ' Set Baud rate

Rx              CON     15    ' XBee DOUT
Tx              CON     14    ' XBee DIN

State           VAR     Bit

Led             PIN     0
Buzzer          PIN     5


' ***************** Main Loop

DO
 SERIN Rx, Baud, [State]         ' Wait for byte and accept
 IF State = 1 THEN               ' Based on value,
    HIGH LED                     ' if 1, Turn on LED
    FREQOUT Buzzer,200,3000      ' Sound buzzer
 ELSE
    LOW LED                      ' if 0, turn off LED
 ENDIF
LOOP
```

Example 2: Multiple Decimal Values with Addressing & Flow Control

This example illustrates sending multiple decimal values between BASIC Stamp modules along with using RTS flow control on the receiver and addressing of nodes through the XBee Command Mode. The transmitting hardware uses the BASIC Stamp Editor's Debug Terminal. The receiving hardware may be connected to a Debug Terminal as well, but it is not required. Multiple receivers may be configured to test addressing.

Transmitter Code:

This code requests data from the user via the Debug Terminal to control an LED and buzzer connected to a remote BASIC Stamp. Major actions of the code:

- Configures the XBee by entering AT Command Mode and sending the AT Command to set the guard time to a low value. This allows for quickly entering AT Command mode to change the destination address of a packet "on-the-fly."
- Requests from the user the destination address of data (remote node address). *In testing, use the address set in receiver code (1 by default), a non-assigned address, or use the broadcast address of FFFF to send to all receivers.*
- Requests from the user the remote LED state (0/1) and remote frequency to sound.
- Enters AT Command mode to quickly set the destination address of the data packet.
- Sends a start delimiting character (!) and the two decimal values for LED state and buzzer frequency.

```
' ******************************************************
' Multiple_data_with_Config_Tx.bs2
' This program:
'   - Configures XBee for fast AT Command Mode
'   - Requests destination address, LED state
'     & buzzer frequency in Debug Terminal
'   - Sets address and sends start delimiter (!)
'     and data to selected node address
'   - Requires 802.15.4 XBee (Series 1)
' ******************************************************

' {$STAMP BS2}
' {$PBASIC 2.5}
```

```
#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T9600       CON     84
  #CASE BS2SX, BS2P
    T9600       CON     240
  #CASE BS2PX
    T9600       CON     396
#ENDSELECT


' ***************** Variables, Constants and Pins

Baud            CON     T9600  ' Set Baud rate

Rx              CON     15     ' XBee DOUT
Tx              CON     14     ' XBee DIN

Freq            VAR     Word   ' Frequency to send
State           VAR     Bit    ' State of remote LED
DL_Addr         VAR     Word   ' Destination address for data


' ***************** Configure XBee in AT Command Mode

PAUSE 500
DEBUG CLS,"Configuring XBee..."

PAUSE 3000                      ' Guard time
SEROUT Tx,Baud,["+++"]          ' Command Mode Sequence
PAUSE 2000                      ' Guard time
SEROUT Tx,Baud,["ATGT 3",CR]    ' Set low guard time
SEROUT TX,Baud,["ATCN",CR]      ' Exit Command Mode


' ***************** Main Loop

DO
  ' Request address, LED state and frequency in DEBUG
  DEBUG CLS,"Enter Node Address in Hex (1-FFFF):"
  DEBUGIN HEX DL_Addr
  DEBUG CR,"Enter LED State (0/1):"
  DEBUGIN DEC State
  DEBUG CR,"Enter Frequency:"
  DEBUGIN DEC Freq

  ' Configure XBee for destination node address
  PAUSE 10                                  ' Short guard time
  SEROUT Tx,Baud,["+++"]                    ' Command Mode sequence
  PAUSE 10                                  ' Short guard time
  SEROUT TX,Baud,["ATDL ", HEX DL_Addr,CR]  ' Set Destination Node Address
  SEROUT Tx,Baud,["ATCN",CR]                ' Exit Command Mode

  ' Send Data - Extra CR's help ensure data accepted properly
  SEROUT Tx,Baud,["!",CR,CR]                ' Send start delimiter
  SEROUT Tx,Baud,[DEC State,CR,CR]          ' Send LED state
  SEROUT Tx,Baud,[DEC Freq,CR,CR]           ' Send buzzer freq
  DEBUG "Data Sent!",CR

  PAUSE 2000
LOOP
```

Receiver Code:

This code receives data consisting of a start delimiter (!), LED state and buzzer frequency. It uses RTS flow control to allow the XBee only to send received data to the BASIC Stamp when it is ready for it. While DEBUG is used, monitoring is not necessary in the Debug Terminal. Major actions of the code:

- Configures the XBee using AT Command Mode to enable RTS flow control and to set the node's MY address. *Modify this address as desired (My_Addr constant) from $1 to $FFFE to test. Multiple receivers may be on the network. Use this address for the transmitting BASIC Stamp when requested in Debug Terminal.*
- Waits for a byte with timeout. If the byte is the start delimiter (!), accepts decimal values for LED state and buzzer frequency. The use of the timeout illustrates that other actions may be occurring on your BASIC Stamp since data is buffered on the XBee. In this example a dot will be displayed for each timeout.
- Controls the LED and buzzer as specified by the data received. Displays the values for the user if the Debug Terminal is used.

```
' ******************************************************
' Multiple_data_with_Config_Rx.bs2
' This program:
'   - Configures XBee for address (Modify Address below)
'     and to use RTS flow control
'   - Accepts LED state & buzzer frequency
'   - Sets LED state and sounds tone on buzzer
'   - Requires 802.15.4 XBee (Series 1)
'   You may monitor in Debug Terminal, but not required
' ******************************************************

' {$STAMP BS2}
' {$PBASIC 2.5}

#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T9600       CON     84
  #CASE BS2SX, BS2P
    T9600       CON     240
  #CASE BS2PX
    T9600       CON     396
#ENDSELECT


'****************** Variable, Constants and Pins

Baud            CON     T9600

LED             PIN     0
Buzzer          PIN     5

Rx              CON     15   ' XBee DOUT
Tx              CON     14   ' XBee DIN
RTS             CON     11   ' XBee RTS

Freq            VAR     Word ' Received frequency for buzzer
State           VAR     Bit  ' Received state of LED
DataIn          VAR     Byte ' General byte data

My_Addr         CON     $1   ' Set address of node, modify as desired, $1-$FFFE


' ***************** Configure XBee to use RTS and set Address

PAUSE 500
DEBUG CLS,"Configuring XBee...",CR
PAUSE 3000                      ' Guard time
SEROUT Tx,Baud,["+++"]          ' Command Mode Sequence
PAUSE 2000                      ' Guard time
SEROUT Tx,Baud,["ATD6 1",CR]    ' Enable RTS
```

```
SEROUT Tx,Baud,["ATMY ", HEX My_Addr,CR]   ' Set node address
SEROUT TX,Baud,["ATCN",CR]       ' Exit Command Mode
DEBUG "Awaiting data..."


' ***************** Main Loop

DO
  SERIN Rx\RTS,Baud,10,Timeout,[DataIn]       ' Accept byte
  IF DataIn = "!" THEN                        ' Is start delimiter?
    SERIN Rx\RTS,Baud,1000,Timeout,[DEC State]' Accept LED state
    SERIN Rx\RTS,Baud,1000,Timeout,[DEC Freq] ' Accept buzzer frequency

    DEBUG CR
    DEBUG ? state               ' Display data
    DEBUG ? freq
    DEBUG CR

    IF State = 1 THEN           ' Set LED based on State
      HIGH 0
    ELSE
      LOW 0
    ENDIF

    FREQOUT 5,1000,Freq         ' Sound buzzer based on frequency
  ENDIF

Timeout:
DEBUG "."                       ' Show dot while waiting for data
LOOP
```

ZigBit™ 2.4 GHz Wireless Modules

ATZB-24-A2/B0

Datasheet

8226B–MCU Wireless–06/09

Table of Contents

Section 1

1.1 Summary 1-1
1.2 Applications 1-1
1.3 Key Features 1-2
1.4 Benefits 1-2
1.5 Abbreviations and Acronyms 1-2
1.6 Related Documents 1-4

Section 2

2.1 Overview 2-5

Section 3

3.1 Electrical Characteristics 3-7
3.1.1 Absolute Maximum Ratings 3-7
3.1.2 Test Conditions 3-7
3.1.3 RF Characteristics 3-8
3.1.4 ATmega1281V Microcontroller Characteristics 3-8
3.1.5 Module Interfaces characteristics 3-8
3.2 Physical/Environmental Characteristics and Outline 3-9
3.3 Pin Configuration 3-10
3.4 Mounting Information 3-14
3.5 Sample Antenna Reference Designs 3-15
3.5.1 General recommendations 3-16
3.6 Antenna specifications 3-17
3.6.1 ATZB-24-B0 3-17
3.6.2 ATZB-24-A2 3-19

Section 4

4.1 UNITED STATES (FCC) 4-21
4.2 CANADA (IC) 4-22
4.3 EUROPEAN UNION (ETSI) 4-23
4.4 Approved Antenna List 4-23

Section 5

5.1 Ordering Information 5-24

Section 1

Introduction

1.1 Summary

ZigBit™ is an ultra-compact, low-power, high-sensitivity 2.4 GHz IEEE 802.15.4/ZigBee® OEM module based on the innovative Atmel's mixed-signal hardware platform. It is designed for wireless sensing, control and data acquisition applications. ZigBit modules eliminate the need for costly and time-consuming RF development, and shortens time to market for a wide range of wireless applications.

Two different versions of 2.4 GHz ZigBit modules are available: ATZB-24-B0 module with balanced RF port for applications where the benefits of PCB or external antenna can be utilized and ATZB-24-A2 module with dual chip antenna satisfying the needs of applications requiring integrated, small-footprint antenna design.

1.2 Applications

ZigBit module is compatible with robust IEEE 802.15.4/ZigBee stack that supports a self-healing, self-organizing mesh network, while optimizing network traffic and minimizing power consumption. Atmel offers two stack configurations: BitCloud and SerialNet. BitCloud is a ZigBee PRO certified software development platform supporting reliable, scalable, and secure wireless applications running on Atmel's ZigBit modules. SerialNet allows programming of the module via serial AT-command interface.

The applications include, but are not limited to:

- **Building automation & monitoring**
  - **Lighting controls**
  - **Wireless smoke and CO detectors**
  - **Structural integrity monitoring**
- **HVAC monitoring & control**
- **Inventory management**
- **Environmental monitoring**
- **Security**
- **Water metering**
- **Industrial monitoring**
  - **Machinery condition and performance monitoring**
  - **Monitoring of plant system parameters such as temperature, pressure, flow, tank level, humidity, vibration, etc.**
- **Automated meter reading (AMR)**

1.3 Key Features

- **Ultra compact size (24 x 13.5 x 2.0 mm for ATZB-24-A2 module and 18.8 x 13.5 x 2.0 mm for ATZB-24-B0 module)**
- **Innovative (patent-pending) balanced dual chip antenna design with antenna gain of approximately 0 dBi (for ATZB-24-A2 version)**
- **High RX sensitivity (-101 dBm)**
- **Outperforming link budget (104 dB)**
- **Up to 3 dBm output power**
- **Very low power consumption:**
  - **< 6 µA in Sleep mode,**
  - **19 mA in RX mode,**
  - **18 mA in TX mode**
- **Ample memory resources (128K bytes of flash memory, 8K bytes RAM, 4K bytes EEPROM)**
- **Wide range of interfaces (both analog and digital):**
  - **9 spare GPIO, 2 spare IRQ lines**
  - **4 ADC lines + 1 line for supply voltage control (up to 9 lines with JTAG disabled)**
  - **UART with CTS/RTS control**
  - **USART**
  - **I^2C**
  - **SPI**
  - **1-Wire**
  - **Up to 30 lines configurable as GPIO**
  - **Capability to write own MAC address into the EEPROM**
  - **Optional antenna reference designs**
  - **IEEE 802.15.4 compliant transceiver**
  - **2.4 GHz ISM band**
  - **BitCloud embedded software, including serial bootloader and AT command set**

1.4 Benefits

- **Small physical footprint and low profile for optimum fit in even the smallest of devices**
- **Best-in-class RF link range**
- **Extended battery life**
- **Easy prototyping with 2-layer PCB**
- **Ample memory for user software application**
- **Mesh networking capability**
- **Easy-to-use low cost Evaluation Kit**
- **Single source of support for HW and SW**
- **Worldwide license-free operation**

1.5 Abbreviations and Acronyms

ADC	Analog-to -Digital Converter
API	Application Programming Interface
DC	Direct Current

DTR	Data Terminal Ready
DIP	Duap In-line package
EEPROM	Electrically Erasable Programmable Read-Only Memory
ESD	Electrostatic Discharge
GPIO	General Purpose Input/Output
HAL	Hardware Abstraction Layer
HVAC	Heating, Ventilating and Air Conditioning
HW	Hardware
I^2C	Inter-Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IRQ	Interrupt Request
ISM	Industrial, Scientific and Medical radio band
JTAG	Digital interface for debugging of embedded device, also known as IEEE 1149.1 standard interface
MAC	Medium Access Control layer
MCU	Microcontroller Unit. In this document it also means the processor, which is the core of ZigBit module
NWK	Network layer
OEM	Original Equipment Manufacturer
OTA	Over-The-Air upgrade
PCB	Printed Circuit Board
PER	Package Error Ratio
PHY	Physical layer
RAM	Random Access Memory
RF	Radio Frequency
RTS/CTS	Request to Send/ Clear to Send
RX	Receiver
SMA	Surface Mount Assembly
SPI	Serial Peripheral Interface
SW	Software
TTM	Time To Market
TX	Transmitter
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
ZDK	ZigBit Development Kit
ZigBee, ZigBee PRO	Wireless networking standards targeted at low-power applications
802.15.4	The IEEE 802.15.4-2003 standard applicable to low-rate wireless Personal Area Network

1.6 Related Documents

[1] Atmel 8-bit AVR Microcontroller with 64K/128K/256K Bytes In-System Programmable Flash. 2549F AVR 04/06

[2] Atmel Low-Power Transceiver for ZigBee Applications. AT86RF230 datasheet. doc5131.pdf

[3] IEEE Std 802.15.4-2003 IEEE Standard for Information technology - Part 15.4 Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)

[4] ZigBee Specification. ZigBee Document 053474r17, October 19, 2007

[5] BitCloud™ IEEE 802.15.4/ZigBee Software. AVR2050: BitCloud User Guide. Atmels doc8199.pdf

[6] ZigBit™ Development Kit. User's Guide. MeshNetics Doc. S-ZDK-451 - TBD

Section 2

Zigbit™ Module Overview

2.1 Overview

ZigBit is a low-power, high-sensitivity IEEE 802.15.4/ ZigBee-compliant OEM module. This multi-functional device occupies less than a square inch of space, which is comparable to a typical size of a single chip. Based on a solid combination of Atmel's latest MCU Wireless hardware platform [1], the ZigBit offers superior radio performance, ultra-low power consumption, and exceptional ease of integration.

Figure 2-1. ATZB-24-B0 Block Diagram

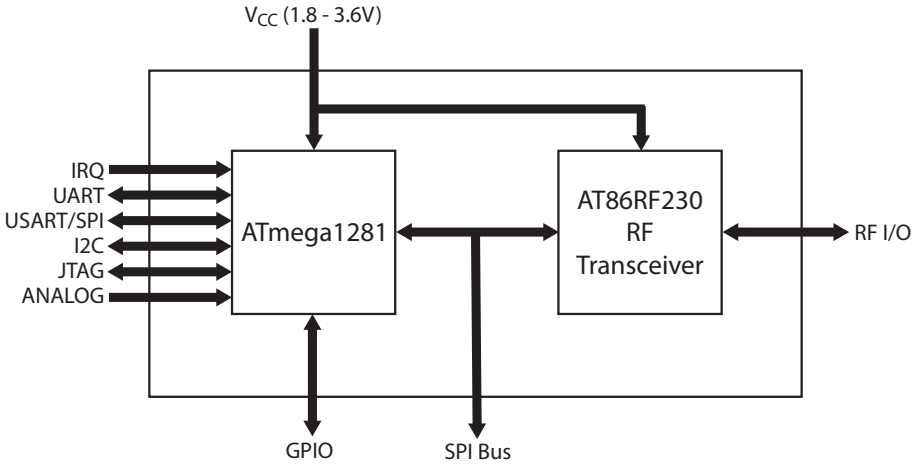
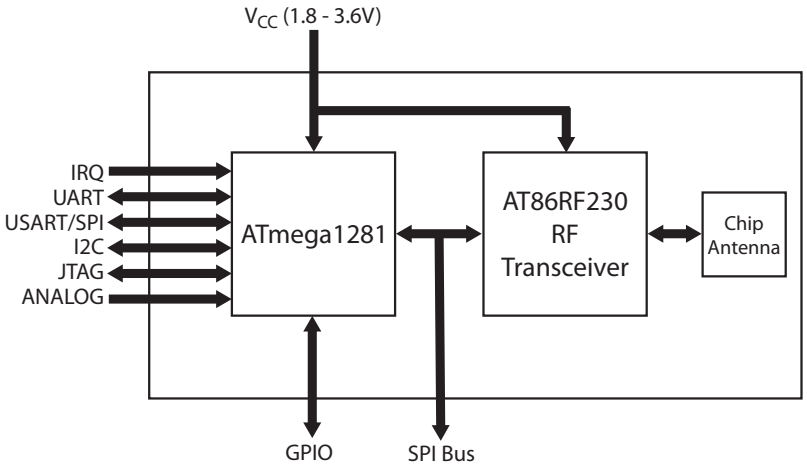


Figure 2-2. ATZB-24-A2 Block Diagram



ZigBit modules comply with the FCC (Part 15), IC and ETSI (CE) rules applicable to the devices radiating in uncontrolled environment. For details, see “Agency Certifications” on page 4-21.

ZigBit fully satisfies the requirements of the “Directive 2002/95/EC of the European Parliament and the Council of 27January 2003 on the restriction of the use of certain hazardous substances in electrical and electronic equipment” (RoHS). Atmel provides fully compliant product in all regions where the directive is enforced since July 1, 2006.

ZigBit contains Atmel’s ATmega1281V Microcontroller [1] and AT86RF230 RF Transceiver [2]. The module features 128 Kbytes flash memory and 8 Kbytes RAM.

The ZigBit already contains a complete RF/MCU-related design with all the necessary passive components included. The module can be easily mounted on a simple 2-layer PCB. Compared to a custom RF/MCU design, a module-based solution offers considerable savings in development time and NRE cost per unit during the design, prototyping, and mass production phases of product development.

Innovative (patent-pending) dual chip antenna design in ATZB-24-A2 module eliminates the balun and achieves good performance over 2.4 GHz frequency band.

To jumpstart evaluation and development, Atmel also offers a complete set of evaluation and development tools. The ZigBit Development Kit [6] (ATZB-DK-24) comes with everything you need to create custom applications featuring ZigBit module.

The kit features MeshBean development boards with an easy-to-access extension connector for attaching third party sensors and other peripherals, and a JTAG connector for easy application uploading and debugging.

The kit also includes reference applications to speed up application development, source code for hardware interface layer and reference drivers for the all the module interfaces, intuitive development environment from Atmel, and comprehensive set of application notes and product documentation.

ZigBit modules comes bundled with BitCloud, a 2nd generation embedded software stack from Atmel. BitCloud is fully compliant with ZigBee PRO and ZigBee standards for wireless sensing and control [3], [4], [5] and it provides an augmented set of APIs which, while maintaining 100% compliance with the standard, offer extended functionality designed with developer's convenience and ease-of-use in mind.

Depending on end-user design requirements, ZigBit can operate as a self-contained sensor node, where it would function as a single MCU, or it can be paired with a host processor driving the module over a serial interface. In the former case, a user application may be used with the BitCloud software allowing customization of embedded applications through BitCloud’s C API.

In the latter case, the host processor controls data transmission and manages module peripherals via an extensive set of SerialNet AT commands. Thus, no firmware customization is required for a successful module design-in. Additionally, third-party sensors can be connected directly to the module, thus expanding the existing set of peripheral interfaces.

Section 3

Specifications

3.1 Electrical Characteristics

3.1.1 Absolute Maximum Ratings

Table 3-1. Absolute Maximum Ratings[1][2]

Parameters	Min	Max
Voltage on any pin, except RESET with respect to Ground	-0.5V	VCC + 0.5V
DC Current per I/O Pin		40 mA
DC Current DVCC and DGND pins		200 mA
Input RF Level		+10 dBm

Notes:
1. **Absolute Maximum Ratings** are the values beyond which damage to the device may occur. Under no circumstances must the absolute maximum ratings given in this table be violated. Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device.

   This is a stress rating only. Functional operation of the device at these or other conditions, beyond those indicated in the operational sections of this specification, is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

2. **Attention!** ZigBit is an ESD-sensitive device. Precaution should be taken when handling the device in order to prevent permanent damage.

3.1.2 Test Conditions

,

Table 3-2. Test conditions (unless otherwise stated), V_{CC} = 3V, T_{amb} = 25°C

Parameters	Range	Unit
Supply Voltage, V_{CC}	1.8 to 3.6	V
Current Consumption: RX mode	19	mA
Current Consumption: TX mode[1]	18	mA
Current Consumption: Radio is turned off, MCU is active 50% of the time[1]	14	mA
Current Consumption: Power-save mode[1]	6	µA

Note: 1. The parameters are measured under the following conditions:
a) BitCloud Software is running at 4 MHz clock rate, DTR line management is turned off
b) All interfaces are set to the default state (see Pin Assignment Table)
c) Output TX power is 0 dBm
d) JTAG is not connected

Current consumption actually depends on multiple factors, including but not limited to, the board design and materials, BitCloud settings, network activity, EEPROM read/write operations. It also depends on MCU load and/or peripherals used by an application.

3.1.3 RF Characteristics

Table 3-3. RF Characteristics

Parameters	Condition	Range	Unit
Frequency Band		2.4000 to 2.4835	GHz
Numbers of Channels		16	
Channel Spacing		5	MHz
Transmitter Output Power	Adjusted in 16 steps	-17 to +3	dBm
Receiver Sensitivity	PER = 1%		
On-Air Data Rate		250	kbps
TX Output/ RX Input Nominal Impedance	For balanced output	100	Ω

3.1.4 ATmega1281V Microcontroller Characteristics

Table 3-4. ATmega1281V Characteristics

Parameters	Condition	Range	Unit
On-chip Flash Memory size		128K	bytes
On-chip RAM size		8K	bytes
On-chip EEPROM size		4K	bytes
Operation Frequency		4	MHz

3.1.5 Module Interfaces characteristics

Table 3-5. Module Interfaces characteristics

Parameters	Condition	Range	Unit
UART Maximum Baud Rate		38.4	kbps
ADC Resolution/ Conversion Time	In single conversion mode	10/200	Bits/µs
ADC Input Resistance		>1	MΩ
ADC Reference Voltage (VREF)		1.0 to V_{CC} - 3	V
ADC Input Voltage		0 - VREF	V
I^2C Maximum Clock		222	kHz
GPIO Output Voltage (High/Low)	-10/ 5 mA	2.3/ 0.5	V
Real Time Oscillator Frequency		32.768	kHz

3.2 Physical/Environmental Characteristics and Outline

Parameters	Value	Comments
Size	18.8 x 13.5 x 2.0 mm	ATZB-24-B0
	24.0 x 13.5 x 2.0 mm	ATZB-24-A2
Weight	1.3g	ATZB-24-B0
	1.5g	ATZB-24-A2
Operating Temperature Range	-20°C to +70°C	-40°C to +85°C operational(1)
Operating Relative Humidity Range	no more than 80%	

Note: 1. Minor degration of clock stability may occur.

Figure 3-1. ATZB-24-B0 Mechanical drawing

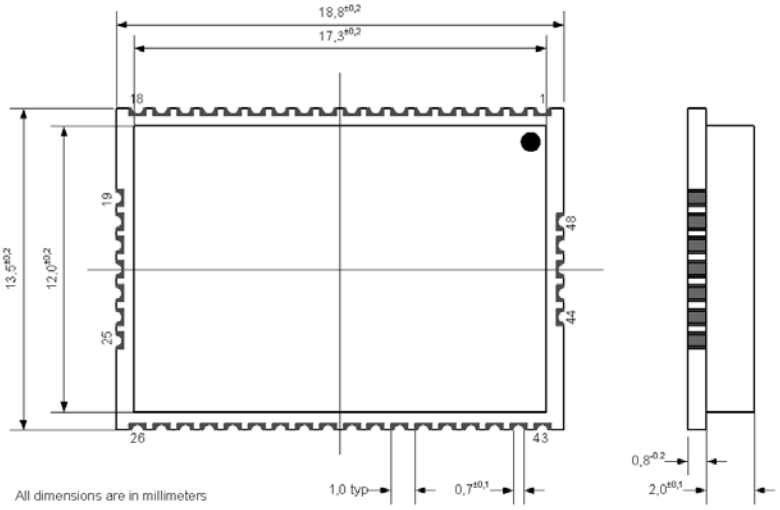
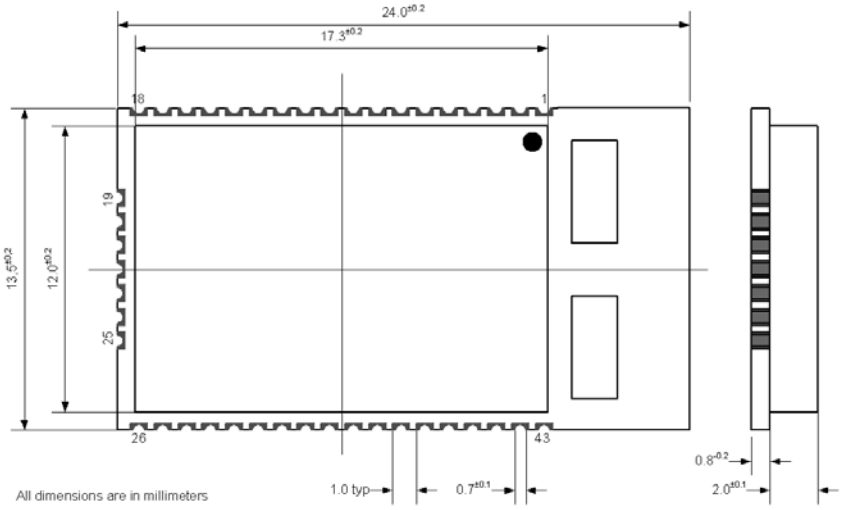


Figure 3-2. ATZB-24-A2 Mechanical drawing



3.3 Pin Configuration

Figure 3-3. ATZB-24-B0 Pinout

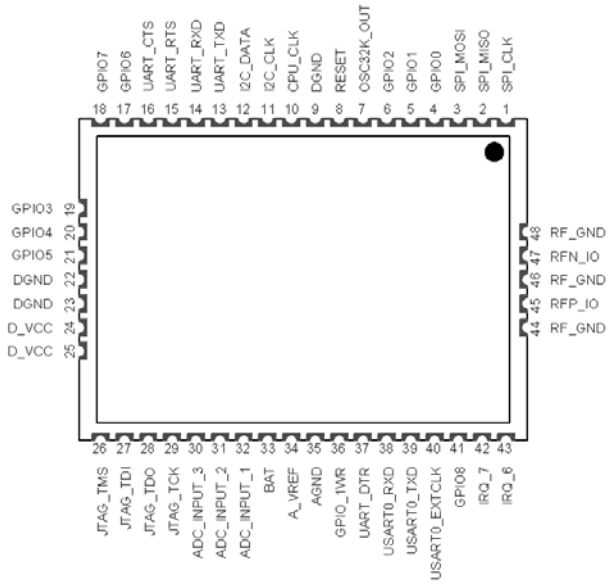


Figure 3-4. ATZB-24-A2 Pinout

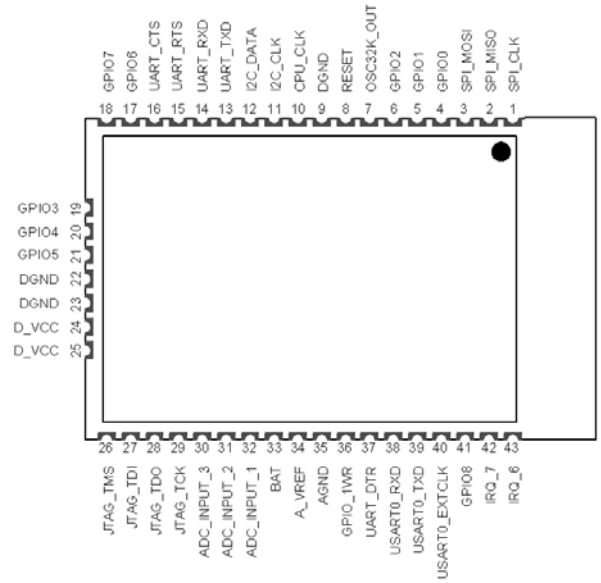


Table 3-6. Pin descriptions

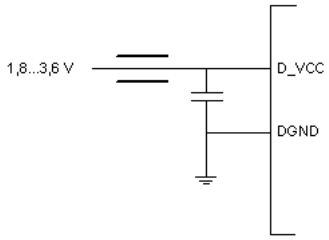
Connector Pin	Pin Name	Description	I/O	Default State after Power on
1	SPI_CLK	Reserved for stack operation[4]	O	
2	SPI_MISO	Reserved for stack operation[4]	I/O	
3	SPI_MOSI	Reserved for stack operation[4]	I/O	
4	GPIO0	General Purpose digital Input/Output 0[2][3][4][7]	I/O	tri-state
5	GPIO1	General Purpose digital Input/Output 1[2][3][4][7]	I/O	tri-state
6	GPIO2	General Purpose digital Input/Output 2[2][3][4][7]	I/O	tri-state
7	OSC32K_OUT	32.768 kHz clock output[4][5]	O	
8	RESET	Reset input (active low)[4]		
9,22,23	DGND	Digital Ground		
10	CPU_CLK	RF clock output. When module is in active state, 4 MHz signal is present on this line. While module is in the sleeping state, clock generation is also stopped[4].	O	
11	I2C_CLK	I^2C Serial clock output[2][3][4][7]	O	tri-state
12	I2C_DATA	I^2C Serial data input/output[2][3][4][7]	I/O	tri-state
13	UART_TXD	UART receive input[1][2][3][4][7]	I	tri-state
14	UART_RXD	UART transmit output[1][2][3][4][7]	O	tri-state
15	UART_RTS	RTS input (Request to send) for UART hardware flow control. Active low[2][3][4][7]	I	tri-state
16	UART_CTS	CTS output (Clear to send) for UART hardware flow control. Active low[2][3][4][7][8]	O	tri-state
17	GPIO6	General Purpose digital Input/Output 6[2][3][4][7]	I/O	tri-state
18	GPIO7	General Purpose digital Input/Output 7[2][3][4][7]	I/O	tri-state
19	GPIO3	General Purpose digital Input/Output 3[2][3][4][7]	I/O	tri-state
20	GPIO4	General Purpose digital Input/Output 4[2][3][4][7]	I/O	tri-state
21	GPIO5	General Purpose digital Input/Output 5[2][3][4][7]	I/O	tri-state
24,25	D_VCC	Digital Supply Voltage (V_{CC})[9]		
26	JTAG_TMS	JTAG Test Mode Select[2][3][4][6]	I	
27	JTAG_TDI	JTAG Test Data Input[2][3][4][6]	I	
28	JTAG_TDO	JTAG Test Data Output[2][3][4][6]	O	
29	JTAG_TCK	JTAG Test Clock[2][3][4][6]	I	
30	ADC_INPUT_3	ADC Input Channel 3[2][3][7]	I	tri-state
31	ADC_INPUT_2	ADC Input Channel 2[2][3][7]	I	tri-state
32	ADC_INPUT_1	ADC Input Channel 1[2][3][7]	I	tri-state
33	BAT	ADC Input Channel 0, used for battery level measurement. This pin equals $V_{CC}/3$.[2][3][7]	I	tri-state
34	A_VREF	Input/Output reference voltage for ADC	I/O	tri-state

Table 3-6. Pin descriptions

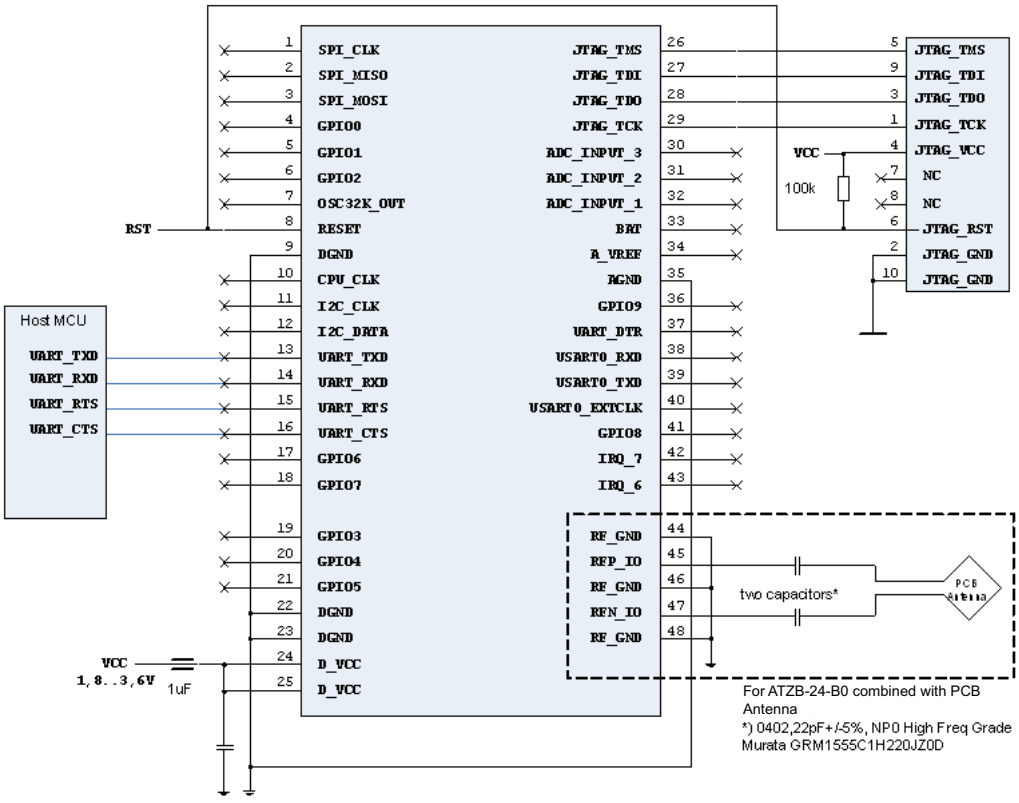
Connector Pin	Pin Name	Description	I/O	Default State after Power on
35	AGND	Analog ground		
36	GPIO_1WR	1-wire interface(2)(3)(4)(7)	I/O	
37	UART_DTR	DTR input (Data Terminal Ready) for UART. Active low(2)(3)(4)(7)	I	tri-state
38	USART0_RXD	USART/SPI Receive pin(2)(3)(4)(7)	I	tri-state
39	USART0_TXD	USART /SPI Transmit pin(2)(3)(4)(7)	O	tri-state
40	USART0_EXTCLK	USART/SPI External Clock(2)(3)(4)(7)(11)	I/O	tri-state
41	GPIO8	General Purpose Digital Input/Output	I/O	tri-state
42	IRQ_7	Digital Input Interrupt request 7(2)(3)(4)(7)	I	tri-state
43	IRQ_6	Digital Input Interrupt request 6(2)(3)(4)(7)	I	tri-state
44,46,48	RF GND	RF Analog Ground(2)(3)(4)(7)		
45	RFP_IO	Differential RF Input/Output(10)	I/O	
47	RFN_IO	Differential RF Input/Output(10)	I/O	

Notes:
1. The UART_TXD pin is intended for input (i.e. its designation as "TXD" implies some complex system containing ZigBit as its RF terminal unit), while UART_RXD pin, vice versa, is for output.
2. Most of pins can be configured for general purpose I/O or for some alternate functions as described in details in the ATmega1281V Datasheet [1].
3. GPIO pins can be programmed either for output, or for input with/without pull-up resistors. Output pin drivers are strong enough to drive LED displays directly (refer to figures on pages 387-388, [1]).
4. All digital pins are provided with protection diodes to D_VCC and DGND
5. It is strongly recommended to avoid assigning an alternate function for OSC32K_OUT pin because it is used by BitCloud. However, this signal can be used if another peripheral or host processor requires 32.768 kHz clock, otherwise this pin can be disconnected.
6. Normally, JTAG_TMS, JTAG_TDI, JTAG_TDO, JTAG_TCK pins are used for on-chip debugging and flash burning. They can be used for A/D conversion if JTAGEN fuse is disabled.
7. The following pins can be configured with the BitCloud software to be general-purpose I/O lines: GPIO0, GPIO1, GPIO2, GPIO3, GPIO4, GPIO5, GPIO6, GPIO7, GPIO8, GPIO_1WR, I2C_CLK, I2C_DATA, UART_TXD, UART_RXD, UART_RTS, UART_CTS, ADC_INPUT_3, ADC_INPUT_2, ADC_INPUT_1, BAT, UART_DTR, USART0_RXD, USART0_TXD, USART0_EXTCLK, IRQ_7, IRQ_6. Additionally, four JTAG lines can be programmed with software as GPIO as well, but this requires changing the fuse bits and will disable JTAG debugging.
8. With BitCloud, CTS pin can be configured to indicate sleep/active condition of the module thus providing mechanism for power management of host processor. If this function is necessary, connection of this pin to external pull-down resistor is recommended to prevent the undesirable transients during module reset process.

9. Using ferrite bead and 1 µF capacitor located closely to the power supply pin is recommended, as shown below.



10. Pins 44 through 48 are not designed for the ATZB-24-A2 module. Note these pins are used in ATZB-24-B0, see them in antenna schematics below.



11. In SPI mode, USART0_EXTCLK is output. In USART mode, this pin can be configured as either input or output pin.

3.4 Mounting Information

The below diagrams show the PCB layout recommended for ZigBit module. Neither via-holes nor wires are allowed on the PCB upper layer in area occupied by the module. As a critical requirement, RF_GND pins should be grounded via several holes to be located right next to the pins thus minimizing inductance and preventing both mismatch and losses.

Figure 3-5. ATZB-24-B0 PCB Recommended Layout, Top View

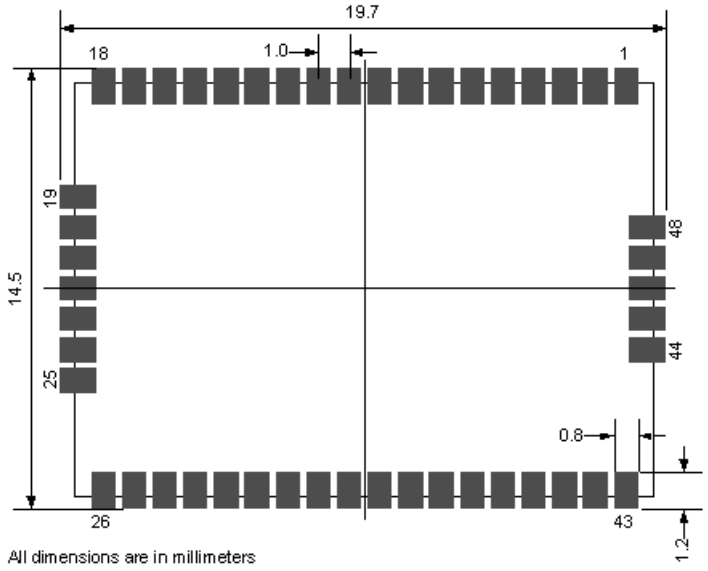
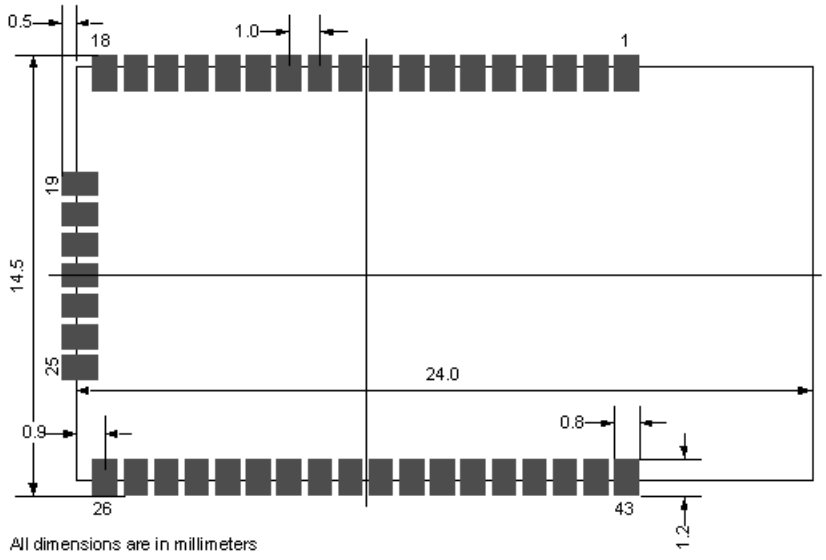


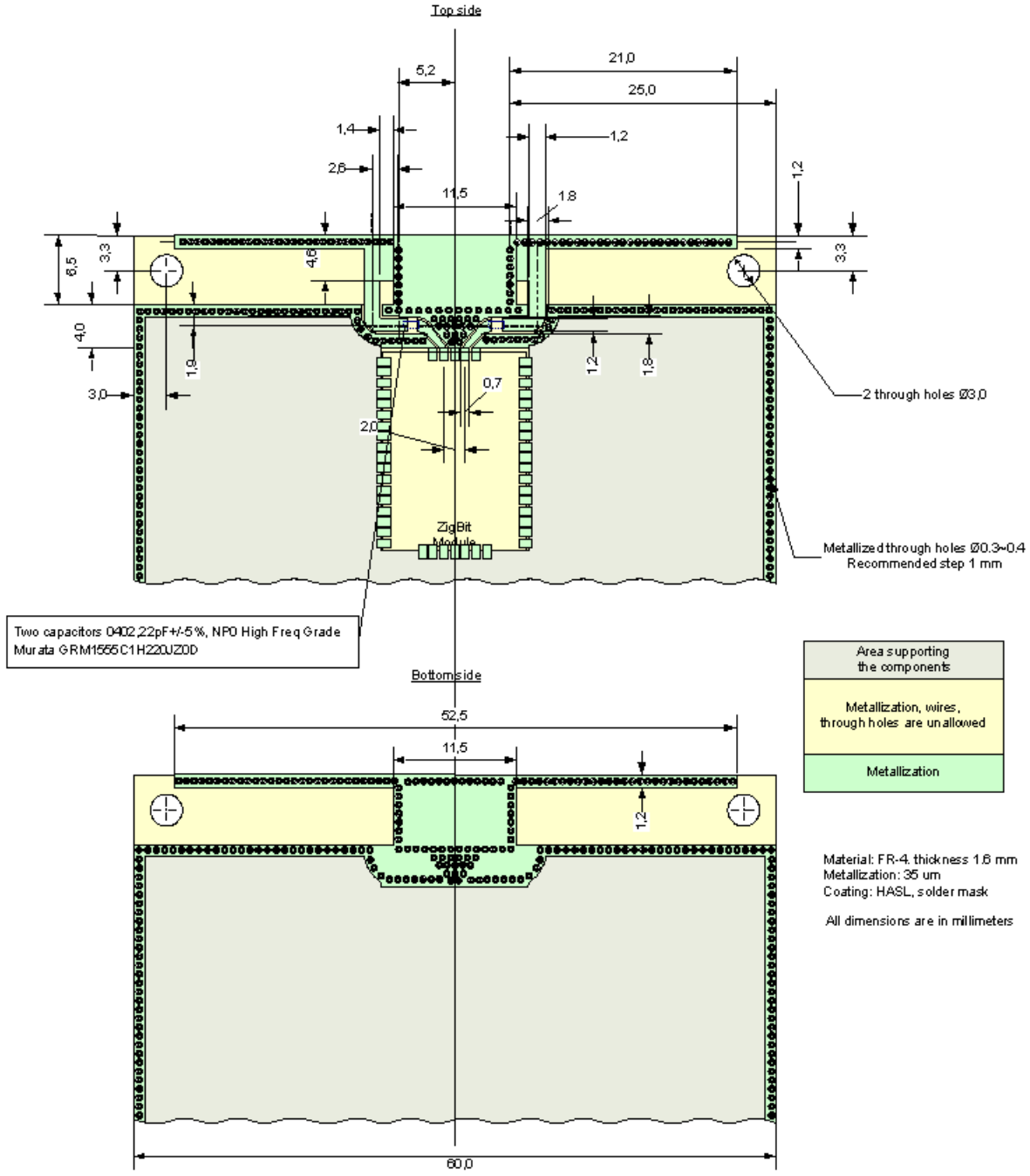
Figure 3-6. ATZB-24-A2 PCB Recommended Layout, Top View



3.5 Sample Antenna Reference Designs

This section presents PCB designs which combine ZigBit with different antennas: PCB onboard antenna, external antenna and dual chip antenna. These antenna reference designs are recommended for successful design-in.

Figure 3-7. PCB Layout: Symmetric Dipole Antenna recommended for ATZB-24-B0



The symmetric dipole antenna above has been tuned for the particular design. The 'cut-and-paste' approach would not guarantee optimal performance because of multiple factors affecting proper antenna

match, hence, affecting the pattern. The particular factors are the board material and thickness, shields, the material used for enclosure, the board neighborhood, and other components adjacent to antenna.

3.5.1 General recommendations

- Metal enclosure should not be used. Using low profile enclosure might also affect antenna tuning.
- Placing high profile components next to antenna should be avoided.
- Having holes punched around the periphery of the board eliminates parasitic radiation from the board edges also distorting antenna pattern.
- ZigBit module should not be placed next to consumer electronics which might interfere with ZigBit's RF frequency band.

3.6 Antenna specifications

3.6.1 ATZB-24-B0

Figure 3-8. Symmetric Dipole Antenna Pattern (horizontal and vertical plane) for ATZB-24-B0

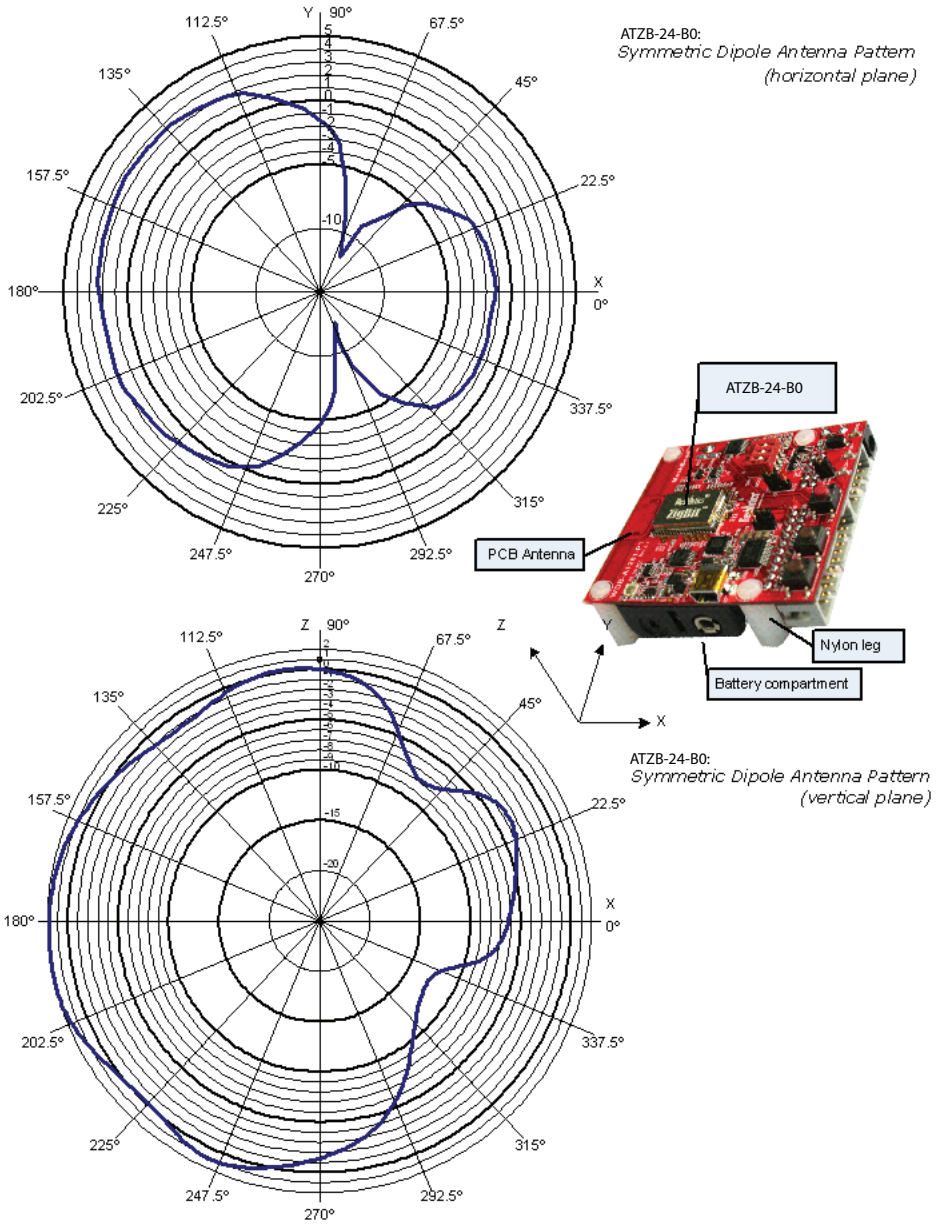
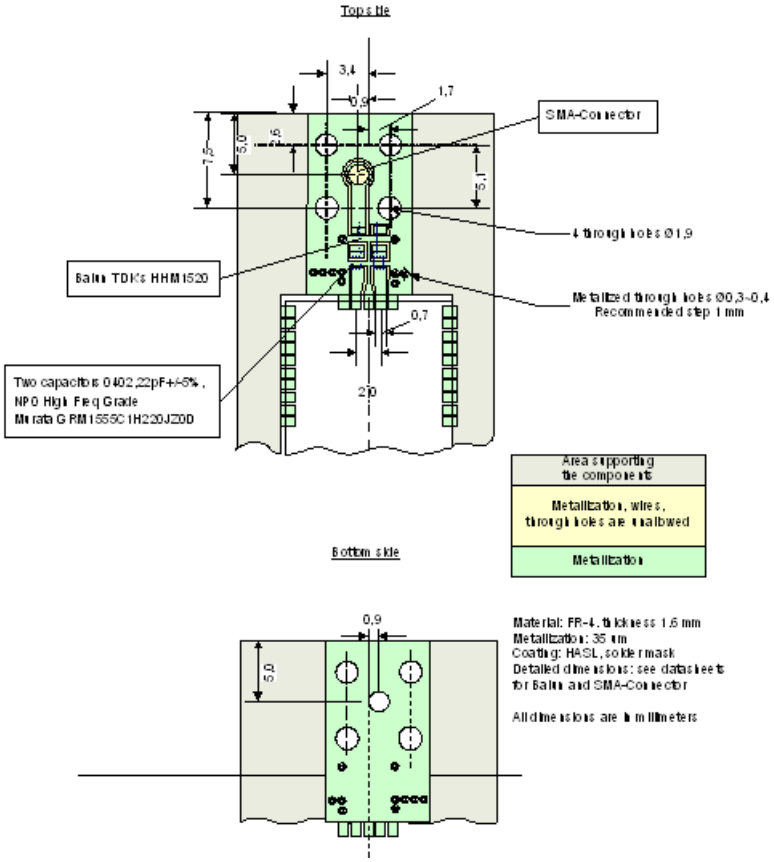
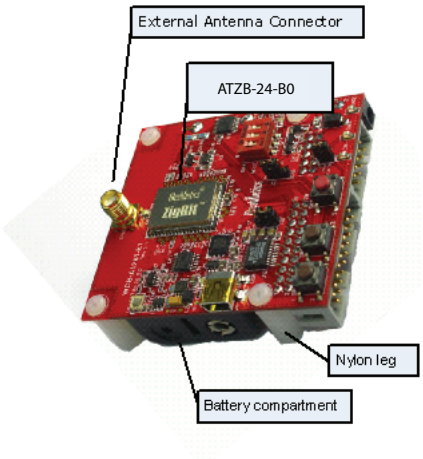


Figure 3-9. PCB Layout with 50 Ohm External Antenna recommended for ATZB-24-B0



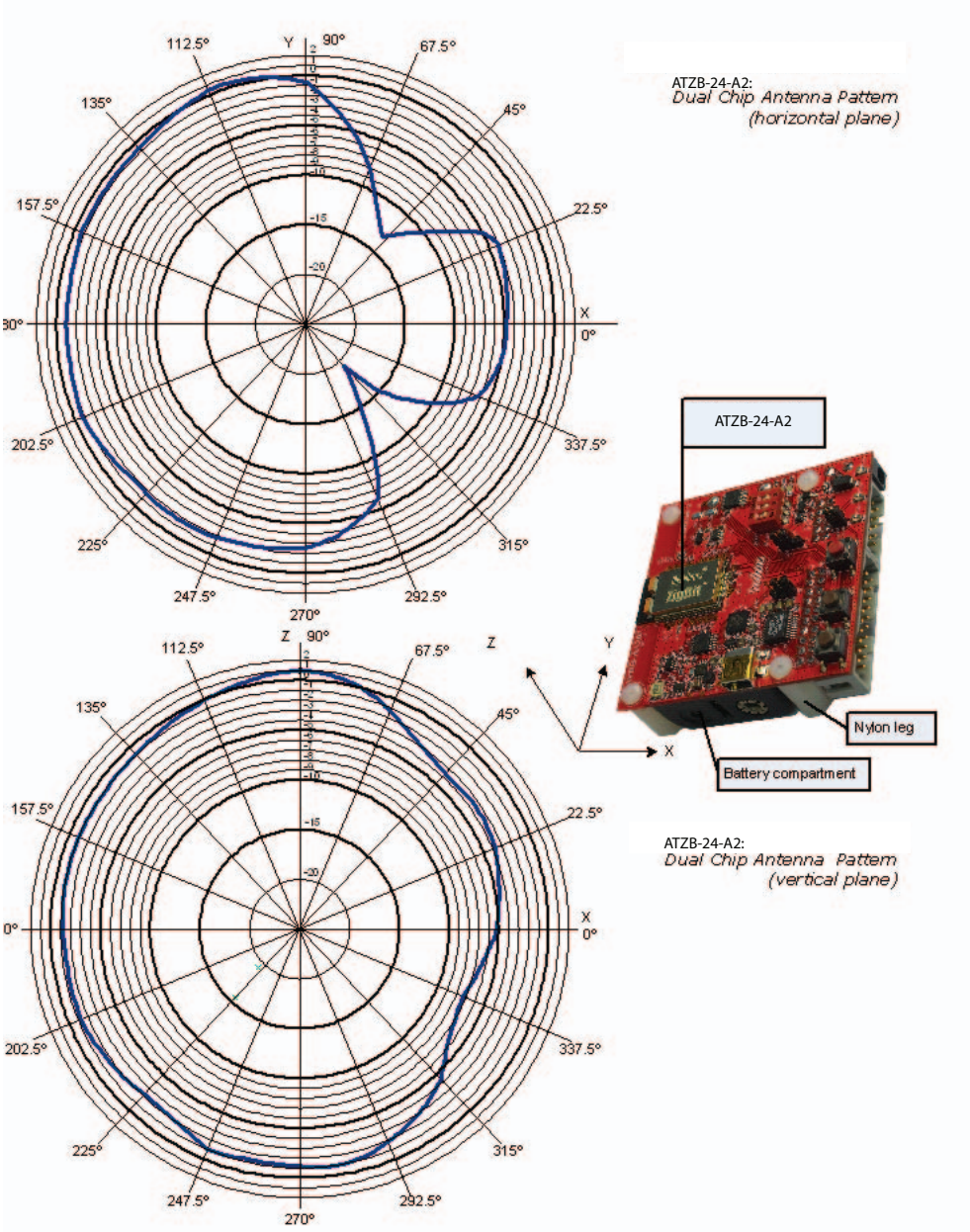
In case the external unbalanced 50 Ohm antenna is required, it can be easily interfaced to ATZB-24-B0 module by using 2:1 balun as shown above. The reference design in Figure 3-10 demonstrates how to use SMA connector.

Figure 3-10. SMA connectors



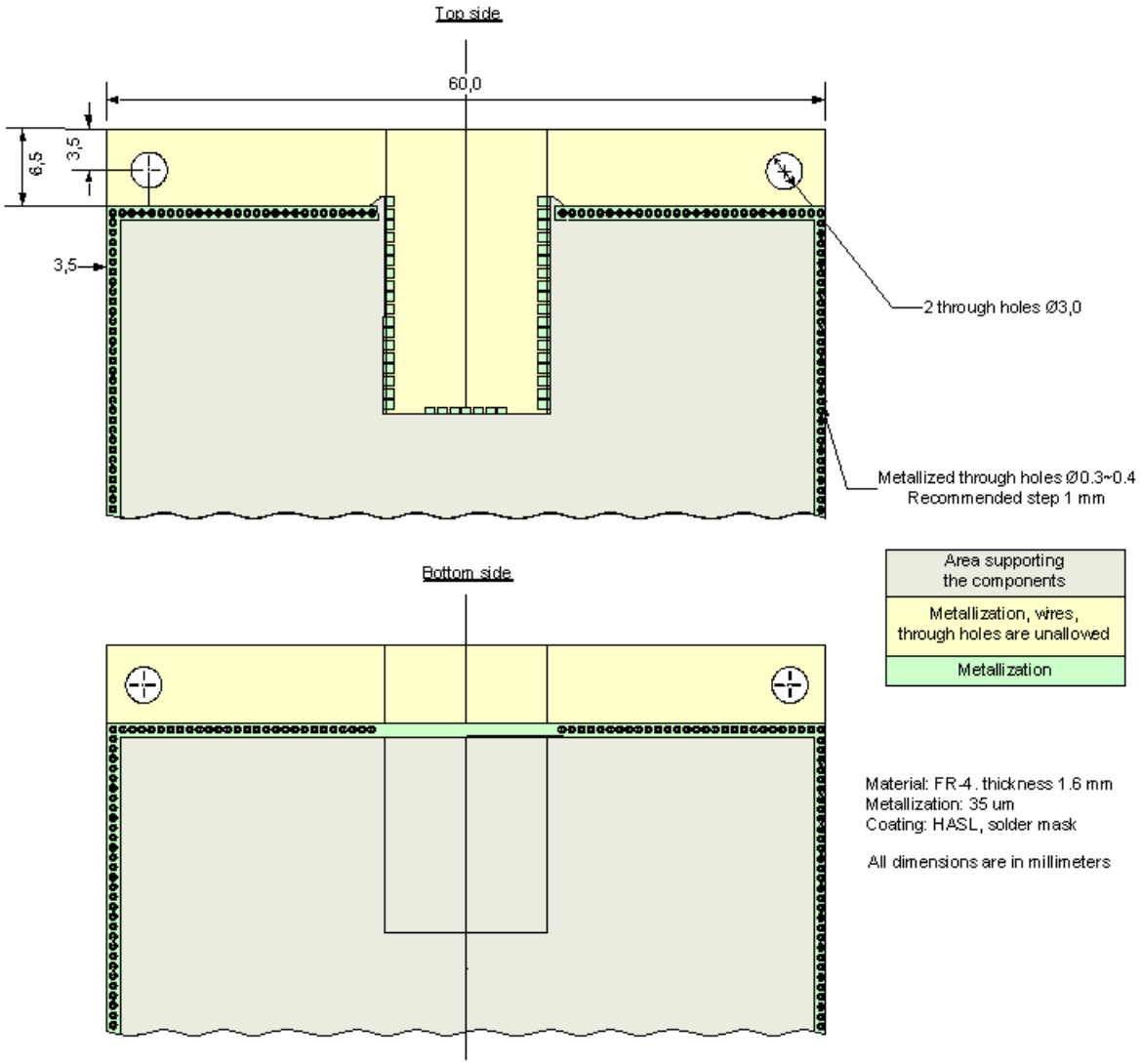
3.6.2 ATZB-24-A2

Figure 3-11. Symmetric Dipole Antenna Pattern (horizontal and vertical plane) for ATZB-24-A2



Note: The antenna patterns presented above were observed using PCB enhanced with legs made of original nylon.

Figure 3-12. PCB Layout with Dual Chip Antenna Module recommended for ATZB-24-A2



Normally, chip antennas are more tolerant of the board or enclosure materials in ZigBit's neighborhood as well. However, general recommendations given above for the PCB antenna design still apply.

The board design should prevent propagation of microwave field inside the board material. Electromagnetic waves of high frequency may penetrate the board thus making the edges of the board radiate, which may distort the antenna pattern. To eliminate this effect, metalized and grounded holes must be placed around the board's edges as shown.

Since the design of dual chip antenna is intended for installation on FR-4 board 1.6 mm thick, the antenna performance may only be guaranteed for the particular board type and thickness.

Section 4

Agency Certifications

4.1 UNITED STATES (FCC)

This equipment complies with Part 15 of the FCC rules and regulations.

To fulfill FCC Certification requirements, an OEM manufacturer must comply with the following regulations:

1. The modular transmitter must be labelled with its own FCC ID number, and, if the FCC ID is not visible when the module is installed inside another device, then the outside of the device into which the module is installed must also display a label referring to the enclosed module. This exterior label can use wording such as the following:

   Example of label required for OEM product containing ATZB-24-A2 module

Contains FCC ID: U6TZIGBIT-A2
The enclosed device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (i.) this device may not cause harmful interference and (ii.) this device must accept any interference received, including interference that may cause undesired operation.

   Example of label required for OEM product containing ATZB-24-B0 module

Contains FCC ID: U6TZIGBIT-B0
The enclosed device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (i.) this device may not cause harmful interference and (ii.) this device must accept any interference received, including interference that may cause undesired operation.

   Any similar wording that expresses the same meaning may be used.

2. To be used with the ATZB-24-B0 module, the external antennas have been tested and approved which are specified in here below. The ATZB-24-B0 Module may be integrated with other custom design antennas which OEM installer must authorize following the FCC 15.21 requirements.

WARNING: The Original Equipment Manufacturer (OEM) must ensure that the OEM modular transmitter must be labeled with its own FCC ID number. This includes a clearly visible label on the outside of the final product enclosure that displays the contents shown below. If the FCC ID is not visible when the equipment is installed inside another device, then the outside of the device into which the equipment is installed must also display a label referring to the enclosed equipment.

IMPORTANT: This equipment complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation (FCC 15.19).

The internal / external antenna(s) used for this mobile transmitter must provide a separation distance of at least 20 cm from all persons and must not be co-located or operating in conjunction with any other antenna or transmitter.

Installers must be provided with antenna installation instructions and transmitter operating conditions for satisfying RF exposure compliance. This device is approved as a mobile device with respect to RF exposure compliance, and may only be marketed to OEM installers. Use in portable exposure conditions (FCC 2.1093) requires separate equipment authorization.

IMPORTANT: Modifications not expressly approved by this company could void the user's authority to operate this equipment (FCC section 15.21).

IMPORTANT: This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense (FCC section 15.105).

4.2 CANADA (IC)

Equipment is subject to certification under the applicable RSSs, shall be permanently labelled on each item, or as an inseparable combination. The label must contain the following information for full compliance:

For ATZB-24-A2 module:

Certification Number:	**IC: 7036A-ZIGBITA2**
Manufacturer's Name, Trade Name or Brand Name:	**ZIGBIT**
Model Name:	**ATZB-24-A2**

For ATZB-24-B0 module:

Certification Number:	**IC: 7036A-ZIGBITB0**
Manufacturer's Name, Trade Name or Brand Name:	**ZIGBIT**
Model Name:	**ATZB-24-B0**

IMPORTANT: This equipment for which a certificate has been issued is not considered certified if it is not properly labelled. The information on the Canadian label can be combined with the manufacturer's other labelling requirements

IMPORTANT: Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

IMPORTANT: To reduce potential radio interference to other users, the antenna type and its gain should be so chosen that the equivalent isotropically radiated power (e.i.r.p.) is not more than that permitted for successful communication.

IMPORTANT: The installer of this radio equipment must ensure that the antenna is located or pointed such that it does not emit RF field in excess of Health Canada limits for the general population. Consult Safety Code 6, obtainable from Health Canada's website www.hc-sc.gc.ca/rpb.

4.3 EUROPEAN UNION (ETSI)

The ATZB-24-A2 and ATZB-24-B0 Modules has been certified for use in European Union countries.

If the ATZB-24-A2 and ATZB-24-B0 Modules are incorporated into a product, the manufacturer must ensure compliance of the final product to the European harmonized EMC and low-voltage/safety standards. A Declaration of Conformity must be issued for each of these standards and kept on file as described in Annex II of the R&TTE Directive.

Furthermore, the manufacturer must maintain a copy of the ATZB-24-A2 and ATZB-24-B0 Modules documentation and ensure the final product does not exceed the specified power ratings, antenna specifications, and/or installation requirements as specified in the user manual. If any of these specifications are exceeded in the final product, a submission must be made to a notified body for compliance testing to all required standards.

IMPORTANT: The 'CE' marking must be affixed to a visible location on the OEM product. The CE mark shall consist of the initials "CE" taking the following form:

- If the CE marking is reduced or enlarged, the proportions given in the above graduated drawing must be respected.
- The CE marking must have a height of at least 5mm except where this is not possible on account of the nature of the apparatus.
- The CE marking must be affixed visibly, legibly, and indelibly.

More detailed information about CE marking requirements you can find at "**DIRECTIVE 1999/5/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL**" on 9 March 1999 at section 12.

Certification Approved Antennas list is presented in below.

4.4 Approved Antenna List

ATZB-24-A2 Module works with integrated dual chip antenna. The design of the antenna is fully compliant with all the aforementioned regulation.

ATZB-24-B0 Module has been tested and approved for use with the antennas listed in the table below. ATZB-24-0B Module may be integrated with other custom design antennas which OEM installer must authorize with respective regulatory agencies.

Table 4-1. Approved Antenna specifications

Part Number	Manufacture and description	Gain [dBi]	Minimum separation [cm]
2010B48-01	Antenova Titanis, swivel antenna (1/4 wave antenna) with SMA connector, frequency range 2.4 - 2.5 GHz	2.2	20
17010.10	WiMo, swivel antenna (1/2 wave antenna) with SMA connector, frequency range 2.35 - 2.5 GHz	2.1	20

Section 5

Ordering Information

5.1 Ordering Information

Part Number	Description
ATZB-24-B0R	2.4 GHz IEEE802.15.4/ZigBee OEM Module w/ Balanced RF Port
ATZB-24-A2R	2.4 GHz IEEE802.15.4/ZigBee OEM Module with dual chip antenna

Note: Tape&Reel quantity: 200

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory segments
  - 4/8/16K Bytes of In-System Self-programmable Flash program memory
  - 256/512/512 Bytes EEPROM
  - 512/1K/1K Bytes Internal SRAM
  - Write/Erase cyles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C()
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
  - 6-channel 10-bit ADC in PDIP Package
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - DebugWIRE On-Chip Debug System
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Five Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, and Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V for ATmega48V/88V/168V
  - 2.7 - 5.5V for ATmega48/88/168
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - ATmega48V/88V/168V: 0 - 4 MHz @ 1.8 - 5.5V, 0 - 10 MHz @ 2.7 - 5.5V
  - ATmega48/88/168: 0 - 10 MHz @ 2.7 - 5.5V, 0 - 20 MHz @ 4.5 - 5.5V
- Low Power Consumption
  - Active Mode:
    - 250 µA at 1 MHz, 1.8V
    - 15 µA at 32 kHz, 1.8V (including Oscillator)
  - Power-down Mode:
    - 0.1µA at 1.8V

Note: 1. See “Data Retention” on page 7 for details.

8-bit AVR® Microcontroller with 8K Bytes In-System Programmable Flash

ATmega48/V
ATmega88/V
ATmega168/V

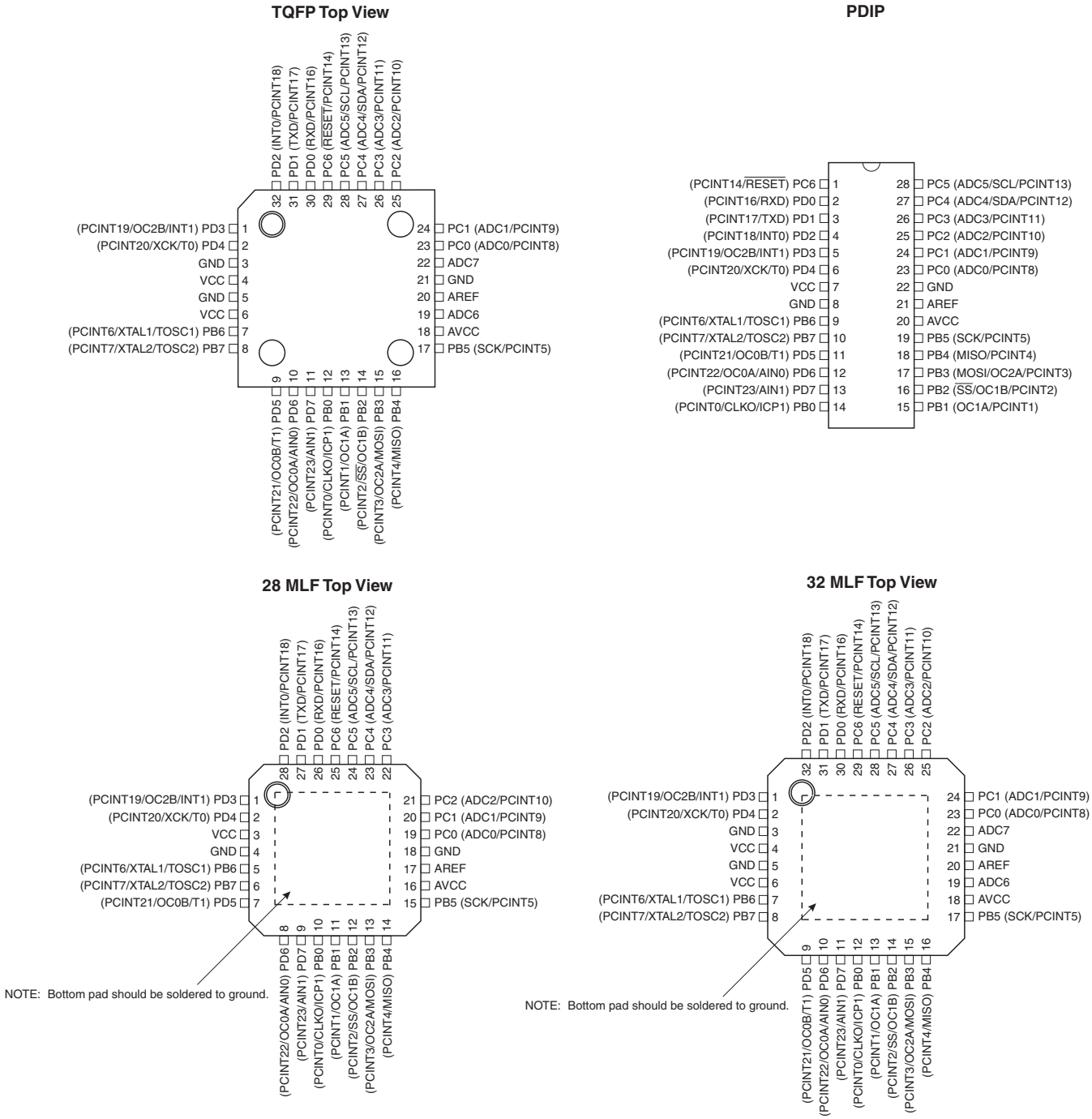
Summary

Note: Not recommended for new designs

Rev. 2545RS–AVR–07/09

1. Pin Configurations

Figure 1-1. Pinout ATmega48/88/1682545RS



1.1 Pin Descriptions

1.1.1 VCC

Digital supply voltage.

1.1.2 GND

Ground.

1.1.3 Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TOSC2..1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

The various special features of Port B are elaborated in “Alternate Functions of Port B” on page 77 and “System Clock and Clock Options” on page 26.

1.1.4 Port C (PC5:0)

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5..0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

1.1.5 PC6/$\overline{\text{RESET}}$

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. The minimum pulse length is given in Table 26-3 on page 306. Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in “Alternate Functions of Port C” on page 80.

1.1.6 Port D (PD7:0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up

resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

The various special features of Port D are elaborated in “Alternate Functions of Port D” on page 83.

1.1.7 AV_{CC}

AV_{CC} is the supply voltage pin for the A/D Converter, PC3:0, and ADC7:6. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter. Note that PC6..4 use digital supply voltage, V_{CC}.

1.1.8 AREF

AREF is the analog reference pin for the A/D Converter.

1.1.9 ADC7:6 (TQFP and QFN/MLF Package Only)

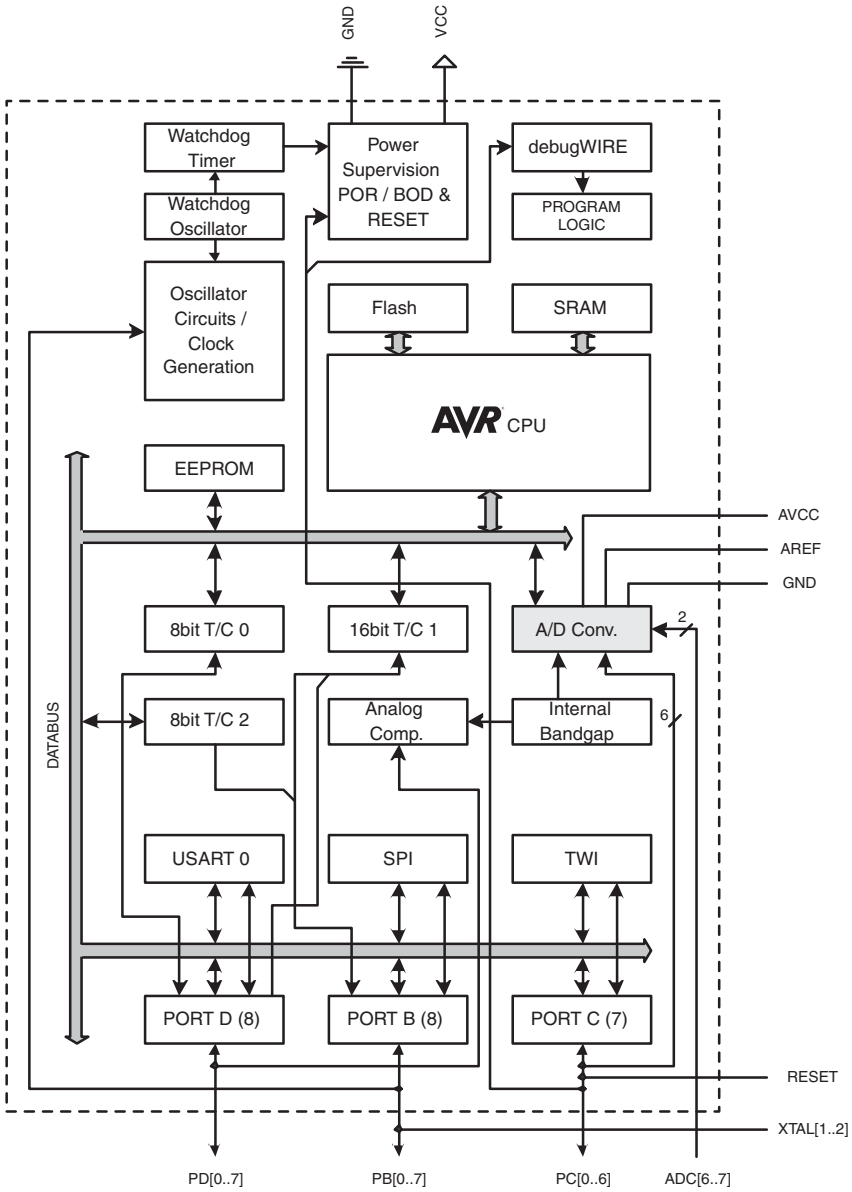
In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

2. Overview

The ATmega48/88/168 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega48/88/168 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

2.1 Block Diagram

Figure 2-1. Block Diagram



The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting

architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega48/88/168 provides the following features: 4K/8K/16K bytes of In-System Programmable Flash with Read-While-Write capabilities, 256/512/512 bytes EEPROM, 512/1K/1K bytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte-oriented 2-wire Serial Interface, an SPI serial port, a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages), a programmable Watchdog Timer with internal Oscillator, and five software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, USART, 2-wire Serial Interface, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption.

The device is manufactured using Atmel's high density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional non-volatile memory programmer, or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega48/88/168 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega48/88/168 AVR is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kits.

2.2 Comparison Between ATmega48, ATmega88, and ATmega168

The ATmega48, ATmega88 and ATmega168 differ only in memory sizes, boot loader support, and interrupt vector sizes. Table 2-1 summarizes the different memory and interrupt vector sizes for the three devices.

Table 2-1. Memory Size Summary

Device	Flash	EEPROM	RAM	Interrupt Vector Size
ATmega48	4K Bytes	256 Bytes	512 Bytes	1 instruction word/vector
ATmega88	8K Bytes	512 Bytes	1K Bytes	1 instruction word/vector
ATmega168	16K Bytes	512 Bytes	1K Bytes	2 instruction words/vector

ATmega88 and ATmega168 support a real Read-While-Write Self-Programming mechanism. There is a separate Boot Loader Section, and the SPM instruction can only execute from there. In ATmega48, there is no Read-While-Write support and no separate Boot Loader Section. The SPM instruction can execute from the entire Flash.

3. About

3.1 Resources

A comprehensive set of development tools, application notes and datasheets are available for download on http://www.atmel.com/avr.

3.2 Data Retention

Reliability Qualification results show that the projected data retention failure rate is much less than 1 PPM over 20 years at 85°C or 100 years at 25°C.

3.3 Code Examples

This documentation contains simple code examples that briefly show how to use various parts of the device. These code examples assume that the part specific header file is included before compilation. Be aware that not all C compiler vendors include bit definitions in the header files and interrupt handling in C is compiler dependent. Please confirm with the C compiler documentation for more details.

For I/O Registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

4. Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xFF)	Reserved	–	–	–	–	–	–	–	–	
(0xFE)	Reserved	–	–	–	–	–	–	–	–	
(0xFD)	Reserved	–	–	–	–	–	–	–	–	
(0xFC)	Reserved	–	–	–	–	–	–	–	–	
(0xFB)	Reserved	–	–	–	–	–	–	–	–	
(0xFA)	Reserved	–	–	–	–	–	–	–	–	
(0xF9)	Reserved	–	–	–	–	–	–	–	–	
(0xF8)	Reserved	–	–	–	–	–	–	–	–	
(0xF7)	Reserved	–	–	–	–	–	–	–	–	
(0xF6)	Reserved	–	–	–	–	–	–	–	–	
(0xF5)	Reserved	–	–	–	–	–	–	–	–	
(0xF4)	Reserved	–	–	–	–	–	–	–	–	
(0xF3)	Reserved	–	–	–	–	–	–	–	–	
(0xF2)	Reserved	–	–	–	–	–	–	–	–	
(0xF1)	Reserved	–	–	–	–	–	–	–	–	
(0xF0)	Reserved	–	–	–	–	–	–	–	–	
(0xEF)	Reserved	–	–	–	–	–	–	–	–	
(0xEE)	Reserved	–	–	–	–	–	–	–	–	
(0xED)	Reserved	–	–	–	–	–	–	–	–	
(0xEC)	Reserved	–	–	–	–	–	–	–	–	
(0xEB)	Reserved	–	–	–	–	–	–	–	–	
(0xEA)	Reserved	–	–	–	–	–	–	–	–	
(0xE9)	Reserved	–	–	–	–	–	–	–	–	
(0xE8)	Reserved	–	–	–	–	–	–	–	–	
(0xE7)	Reserved	–	–	–	–	–	–	–	–	
(0xE6)	Reserved	–	–	–	–	–	–	–	–	
(0xE5)	Reserved	–	–	–	–	–	–	–	–	
(0xE4)	Reserved	–	–	–	–	–	–	–	–	
(0xE3)	Reserved	–	–	–	–	–	–	–	–	
(0xE2)	Reserved	–	–	–	–	–	–	–	–	
(0xE1)	Reserved	–	–	–	–	–	–	–	–	
(0xE0)	Reserved	–	–	–	–	–	–	–	–	
(0xDF)	Reserved	–	–	–	–	–	–	–	–	
(0xDE)	Reserved	–	–	–	–	–	–	–	–	
(0xDD)	Reserved	–	–	–	–	–	–	–	–	
(0xDC)	Reserved	–	–	–	–	–	–	–	–	
(0xDB)	Reserved	–	–	–	–	–	–	–	–	
(0xDA)	Reserved	–	–	–	–	–	–	–	–	
(0xD9)	Reserved	–	–	–	–	–	–	–	–	
(0xD8)	Reserved	–	–	–	–	–	–	–	–	
(0xD7)	Reserved	–	–	–	–	–	–	–	–	
(0xD6)	Reserved	–	–	–	–	–	–	–	–	
(0xD5)	Reserved	–	–	–	–	–	–	–	–	
(0xD4)	Reserved	–	–	–	–	–	–	–	–	
(0xD3)	Reserved	–	–	–	–	–	–	–	–	
(0xD2)	Reserved	–	–	–	–	–	–	–	–	
(0xD1)	Reserved	–	–	–	–	–	–	–	–	
(0xD0)	Reserved	–	–	–	–	–	–	–	–	
(0xCF)	Reserved	–	–	–	–	–	–	–	–	
(0xCE)	Reserved	–	–	–	–	–	–	–	–	
(0xCD)	Reserved	–	–	–	–	–	–	–	–	
(0xCC)	Reserved	–	–	–	–	–	–	–	–	
(0xCB)	Reserved	–	–	–	–	–	–	–	–	
(0xCA)	Reserved	–	–	–	–	–	–	–	–	
(0xC9)	Reserved	–	–	–	–	–	–	–	–	
(0xC8)	Reserved	–	–	–	–	–	–	–	–	
(0xC7)	Reserved	–	–	–	–	–	–	–	–	
(0xC6)	UDR0	USART I/O Data Register								189
(0xC5)	UBRR0H					USART Baud Rate Register High				193
(0xC4)	UBRR0L	USART Baud Rate Register Low								193
(0xC3)	Reserved	–	–	–	–	–	–	–	–	
(0xC2)	UCSR0C	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01 /UDORD0	UCSZ00 / UCPHA0	UCPOL0	191/206
(0xC1)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80	190
(0xC0)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0	189

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xBF)	Reserved	–	–	–	–	–	–	–	–	
(0xBE)	Reserved	–	–	–	–	–	–	–	–	
(0xBD)	TWAMR	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2	TWAM1	TWAM0	–	238
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	235
(0xBB)	TWDR	2-wire Serial Interface Data Register								237
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	238
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	237
(0xB8)	TWBR	2-wire Serial Interface Bit Rate Register								235
(0xB7)	Reserved	–		–	–	–	–	–	–	
(0xB6)	ASSR	–	EXCLK	AS2	TCN2UB	OCR2AUB	OCR2BUB	TCR2AUB	TCR2BUB	158
(0xB5)	Reserved	–	–	–	–	–	–	–	–	
(0xB4)	OCR2B	Timer/Counter2 Output Compare Register B								157
(0xB3)	OCR2A	Timer/Counter2 Output Compare Register A								156
(0xB2)	TCNT2	Timer/Counter2 (8-bit)								156
(0xB1)	TCCR2B	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	155
(0xB0)	TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	152
(0xAF)	Reserved	–	–	–	–	–	–	–	–	
(0xAE)	Reserved	–	–	–	–	–	–	–	–	
(0xAD)	Reserved	–	–	–	–	–	–	–	–	
(0xAC)	Reserved	–	–	–	–	–	–	–	–	
(0xAB)	Reserved	–	–	–	–	–	–	–	–	
(0xAA)	Reserved	–	–	–	–	–	–	–	–	
(0xA9)	Reserved	–	–	–	–	–	–	–	–	
(0xA8)	Reserved	–	–	–	–	–	–	–	–	
(0xA7)	Reserved	–	–	–	–	–	–	–	–	
(0xA6)	Reserved	–	–	–	–	–	–	–	–	
(0xA5)	Reserved	–	–	–	–	–	–	–	–	
(0xA4)	Reserved	–	–	–	–	–	–	–	–	
(0xA3)	Reserved	–	–	–	–	–	–	–	–	
(0xA2)	Reserved	–	–	–	–	–	–	–	–	
(0xA1)	Reserved	–	–	–	–	–	–	–	–	
(0xA0)	Reserved	–	–	–	–	–	–	–	–	
(0x9F)	Reserved	–	–	–	–	–	–	–	–	
(0x9E)	Reserved	–	–	–	–	–	–	–	–	
(0x9D)	Reserved	–	–	–	–	–	–	–	–	
(0x9C)	Reserved	–	–	–	–	–	–	–	–	
(0x9B)	Reserved	–	–	–	–	–	–	–	–	
(0x9A)	Reserved	–	–	–	–	–	–	–	–	
(0x99)	Reserved	–	–	–	–	–	–	–	–	
(0x98)	Reserved	–	–	–	–	–	–	–	–	
(0x97)	Reserved	–	–	–	–	–	–	–	–	
(0x96)	Reserved	–	–	–	–	–	–	–	–	
(0x95)	Reserved	–	–	–	–	–	–	–	–	
(0x94)	Reserved	–	–	–	–	–	–	–	–	
(0x93)	Reserved	–	–	–	–	–	–	–	–	
(0x92)	Reserved	–	–	–	–	–	–	–	–	
(0x91)	Reserved	–	–	–	–	–	–	–	–	
(0x90)	Reserved	–	–	–	–	–	–	–	–	
(0x8F)	Reserved	–	–	–	–	–	–	–	–	
(0x8E)	Reserved	–	–	–	–	–	–	–	–	
(0x8D)	Reserved	–	–	–	–	–	–	–	–	
(0x8C)	Reserved	–	–	–	–	–	–	–	–	
(0x8B)	OCR1BH	Timer/Counter1 - Output Compare Register B High Byte								133
(0x8A)	OCR1BL	Timer/Counter1 - Output Compare Register B Low Byte								133
(0x89)	OCR1AH	Timer/Counter1 - Output Compare Register A High Byte								133
(0x88)	OCR1AL	Timer/Counter1 - Output Compare Register A Low Byte								133
(0x87)	ICR1H	Timer/Counter1 - Input Capture Register High Byte								134
(0x86)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte								134
(0x85)	TCNT1H	Timer/Counter1 - Counter Register High Byte								133
(0x84)	TCNT1L	Timer/Counter1 - Counter Register Low Byte								133
(0x83)	Reserved	–	–	–	–	–	–	–	–	
(0x82)	TCCR1C	FOC1A	FOC1B	–	–	–	–	–	–	132
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	131
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	129
(0x7F)	DIDR1	–	–	–	–	–	–	AIN1D	AIN0D	242
(0x7E)	DIDR0	–	–	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	258

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0x7D)	Reserved	–	–	–	–	–	–	–	–	
(0x7C)	ADMUX	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	254
(0x7B)	ADCSRB	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	257
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	255
(0x79)	ADCH	ADC Data Register High byte								257
(0x78)	ADCL	ADC Data Register Low byte								257
(0x77)	Reserved	–	–	–	–	–	–	–	–	
(0x76)	Reserved	–	–	–	–	–	–	–	–	
(0x75)	Reserved	–	–	–	–	–	–	–	–	
(0x74)	Reserved	–	–	–	–	–	–	–	–	
(0x73)	Reserved	–	–	–	–	–	–	–	–	
(0x72)	Reserved	–	–	–	–	–	–	–	–	
(0x71)	Reserved	–	–	–	–	–	–	–	–	
(0x70)	TIMSK2	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2	157
(0x6F)	TIMSK1	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	134
(0x6E)	TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	105
(0x6D)	PCMSK2	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	69
(0x6C)	PCMSK1	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	69
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	69
(0x6A)	Reserved	–	–	–	–	–	–	–	–	
(0x69)	EICRA	–	–	–	–	ISC11	ISC10	ISC01	ISC00	66
(0x68)	PCICR	–	–	–	–	–	PCIE2	PCIE1	PCIE0	
(0x67)	Reserved	–	–	–	–	–	–	–	–	
(0x66)	OSCCAL	Oscillator Calibration Register								36
(0x65)	Reserved	–	–	–	–	–	–	–	–	
(0x64)	PRR	PRTWI	PRTIM2	PRTIM0	–	PRTIM1	PRSPI	PRUSART0	PRADC	40
(0x63)	Reserved	–	–	–	–	–	–	–	–	
(0x62)	Reserved	–	–	–	–	–	–	–	–	
(0x61)	CLKPR	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	36
(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	52
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	10
0x3E (0x5E)	SPH	–	–	–	–	–	(SP10) [5.]	SP9	SP8	12
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	12
0x3C (0x5C)	Reserved	–	–	–	–	–	–	–	–	
0x3B (0x5B)	Reserved	–	–	–	–	–	–	–	–	
0x3A (0x5A)	Reserved	–	–	–	–	–	–	–	–	
0x39 (0x59)	Reserved	–	–	–	–	–	–	–	–	
0x38 (0x58)	Reserved	–	–	–	–	–	–	–	–	
0x37 (0x57)	SPMCSR	SPMIE	(RWWSB)[5.]	–	(RWWSRE)[5.]	BLBSET	PGWRT	PGERS	SELFPRGEN	282
0x36 (0x56)	Reserved	–	–	–	–	–	–	–	–	
0x35 (0x55)	MCUCR	–	–	–	PUD	–	–	IVSEL	IVCE	
0x34 (0x54)	MCUSR	–	–	–	–	WDRF	BORF	EXTRF	PORF	
0x33 (0x53)	SMCR	–	–	–	–	SM2	SM1	SM0	SE	38
0x32 (0x52)	Reserved	–	–	–	–	–	–	–	–	
0x31 (0x51)	Reserved	–	–	–	–	–	–	–	–	
0x30 (0x50)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	241
0x2F (0x4F)	Reserved	–	–	–	–	–	–	–	–	
0x2E (0x4E)	SPDR	SPI Data Register								169
0x2D (0x4D)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	168
0x2C (0x4C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	167
0x2B (0x4B)	GPIOR2	General Purpose I/O Register 2								25
0x2A (0x4A)	GPIOR1	General Purpose I/O Register 1								25
0x29 (0x49)	Reserved	–	–	–	–	–	–	–	–	
0x28 (0x48)	OCR0B	Timer/Counter0 Output Compare Register B								
0x27 (0x47)	OCR0A	Timer/Counter0 Output Compare Register A								
0x26 (0x46)	TCNT0	Timer/Counter0 (8-bit)								
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	
0x23 (0x43)	GTCCR	TSM	–	–	–	–	–	PSRASY	PSRSYNC	138/159
0x22 (0x42)	EEARH	(EEPROM Address Register High Byte) [5.]								21
0x21 (0x41)	EEARL	EEPROM Address Register Low Byte								21
0x20 (0x40)	EEDR	EEPROM Data Register								21
0x1F (0x3F)	EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	21
0x1E (0x3E)	GPIOR0	General Purpose I/O Register 0								25
0x1D (0x3D)	EIMSK	–	–	–	–	–	–	INT1	INT0	67
0x1C (0x3C)	EIFR	–	–	–	–	–	–	INTF1	INTF0	67

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x1B (0x3B)	PCIFR	–	–	–	–	–	PCIF2	PCIF1	PCIF0	
0x1A (0x3A)	Reserved	–	–	–	–	–	–	–	–	
0x19 (0x39)	Reserved	–	–	–	–	–	–	–	–	
0x18 (0x38)	Reserved	–	–	–	–	–	–	–	–	
0x17 (0x37)	TIFR2	–	–	–	–	–	OCF2B	OCF2A	TOV2	157
0x16 (0x36)	TIFR1	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	135
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0	
0x14 (0x34)	Reserved	–	–	–	–	–	–	–	–	
0x13 (0x33)	Reserved	–	–	–	–	–	–	–	–	
0x12 (0x32)	Reserved	–	–	–	–	–	–	–	–	
0x11 (0x31)	Reserved	–	–	–	–	–	–	–	–	
0x10 (0x30)	Reserved	–	–	–	–	–	–	–	–	
0x0F (0x2F)	Reserved	–	–	–	–	–	–	–	–	
0x0E (0x2E)	Reserved	–	–	–	–	–	–	–	–	
0x0D (0x2D)	Reserved	–	–	–	–	–	–	–	–	
0x0C (0x2C)	Reserved	–	–	–	–	–	–	–	–	
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	87
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	87
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	87
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	86
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	86
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	86
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	86
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	86
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	86
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–	
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–	
0x0 (0x20)	Reserved	–	–	–	–	–	–	–	–	

Note:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the Status Flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such Status Flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATmega48/88/168 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
5. Only valid for ATmega88/168

5. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF − Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 − Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd − 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP[1]	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL[1]	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd − Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd − Rr − C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd − K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

Note: 1. These instructions are only available in ATmega168.

6. Ordering Information

6.1 ATmega48

Speed (MHz)	Power Supply	Ordering Code	Package[1]	Operational Range
10[3]	1.8 - 5.5	ATmega48V-10AI ATmega48V-10MI ATmega48V-10PI ATmega48V-10AU[2] ATmega48V-10MMU[2] ATmega48V-10MU[2] ATmega48V-10PU[2]	32A 32M1-A 28P3 32A 28M1 32M1-A 28P3	Industrial (-40°C to 85°C)
20[3]	2.7 - 5.5	ATmega48-20AI ATmega48-20MI ATmega48-20PI ATmega48-20AU[2] ATmega48-20MMU[2] ATmega48-20MU[2] ATmega48-20PU[2]	32A 32M1-A 28P3 32A 28M1 32M1-A 28P3	Industrial (-40°C to 85°C)

Note:
1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging alternative, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive).Also Halide free and fully Green.
3. See Figure 26-1 on page 304 and Figure 26-2 on page 304.

Package Type	
32A	32-lead, Thin (1.0 mm) Plastic Quad Flat Package (TQFP)

Package Type	
28M1	28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45 mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50 mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
28P3	28-lead, 0.300” Wide, Plastic Dual Inline Package (PDIP)

6.2 ATmega88

Speed (MHz)	Power Supply	Ordering Code	Package[1]	Operational Range
10[3]	1.8 - 5.5	ATmega88V-10AI ATmega88V-10MI ATmega88V-10PI ATmega88V-10AU[2] ATmega88V-10MU[2] ATmega88V-10PU[2]	32A 32M1-A 28P3 32A 32M1-A 28P3	Industrial (-40°C to 85°C)
20[3]	2.7 - 5.5	ATmega88-20AI ATmega88-20MI ATmega88-20PI ATmega88-20AU[2] ATmega88-20MU[2] ATmega88-20PU[2]	32A 32M1-A 28P3 32A 32M1-A 28P3	Industrial (-40°C to 85°C)

Note:
1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging alternative, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive).Also Halide free and fully Green.
3. See Figure 26-1 on page 304 and Figure 26-2 on page 304.

Package Type	
32A	32-lead, Thin (1.0 mm) Plastic Quad Flat Package (TQFP)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50 mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
28P3	28-lead, 0.300” Wide, Plastic Dual Inline Package (PDIP)

6.3 ATmega168

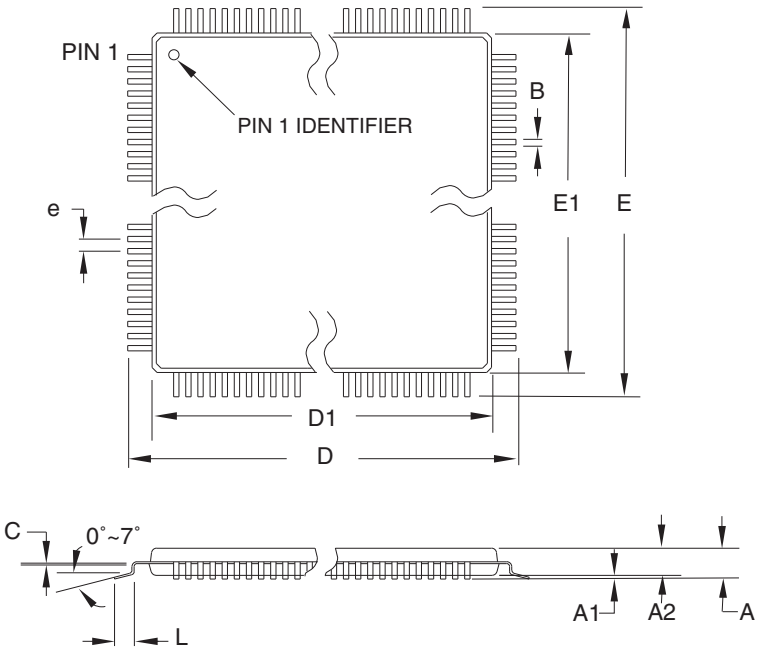
Speed (MHz)[3]	Power Supply	Ordering Code	Package[1]	Operational Range
10	1.8 - 5.5	ATmega168V-10AI ATmega168V-10MI ATmega168V-10PI ATmega168V-10AU[2] ATmega168V-10MU[2] ATmega168V-10PU[2]	32A 32M1-A 28P3 32A 32M1-A 28P3	Industrial (-40°C to 85°C)
20	2.7 - 5.5	ATmega168-20AI ATmega168-20MI ATmega168-20PI ATmega168-20AU[2] ATmega168-20MU[2] ATmega168-20PU[2]	32A 32M1-A 28P3 32A 32M1-A 28P3	Industrial (-40°C to 85°C)

Note:
1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging alternative, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive).Also Halide free and fully Green.
3. See Figure 26-1 on page 304 and Figure 26-2 on page 304.

Package Type	
32A	32-lead, Thin (1.0 mm) Plastic Quad Flat Package (TQFP)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50 mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
28P3	28-lead, 0.300” Wide, Plastic Dual Inline Package (PDIP)

7. Packaging Information

7.1 32A



COMMON DIMENSIONS
(Unit of Measure = mm)

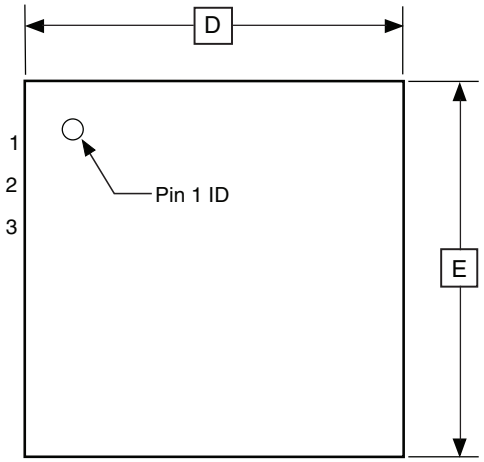
SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	1.20	
A1	0.05	–	0.15	
A2	0.95	1.00	1.05	
D	8.75	9.00	9.25	
D1	6.90	7.00	7.10	Note 2
E	8.75	9.00	9.25	
E1	6.90	7.00	7.10	Note 2
B	0.30	–	0.45	
C	0.09	–	0.20	
L	0.45	–	0.75	
e	0.80 TYP			

Notes:
1. This package conforms to JEDEC reference MS-026, Variation ABA.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.10 mm maximum.

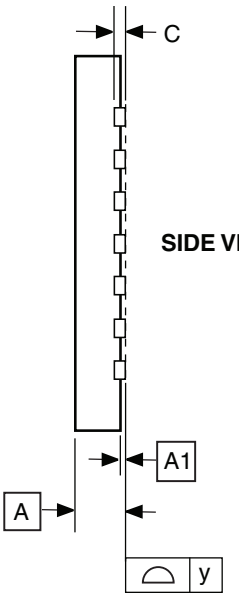
10/5/2001

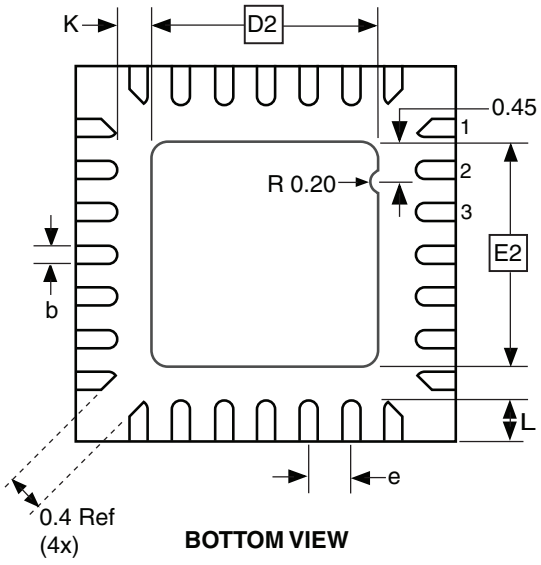
	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**32A,** 32-lead, 7 x 7 mm Body Size, 1.0 mm Body Thickness, 0.8 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP)	32A	B

7.2 28M1



TOP VIEW





BOTTOM VIEW

Note: The terminal #1 ID is a Laser-marked Feature.

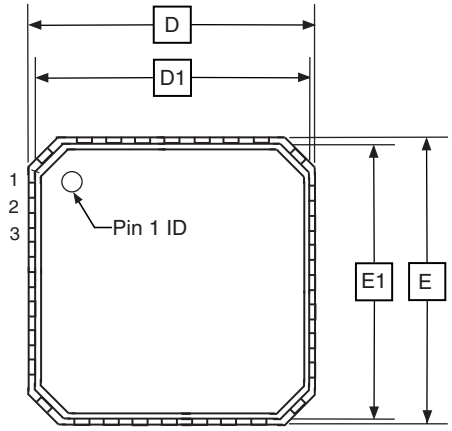
COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	0.80	0.90	1.00	
A1	0.00	0.02	0.05	
b	0.17	0.22	0.27	
C		0.20 REF		
D	3.95	4.00	4.05	
D2	2.35	2.40	2.45	
E	3.95	4.00	4.05	
E2	2.35	2.40	2.45	
e		0.45		
L	0.35	0.40	0.45	
y	0.00	–	0.08	
K	0.20	–	–	

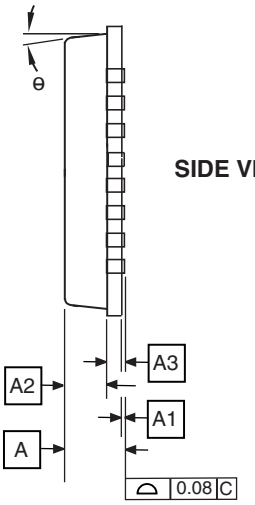
10/24/08

	TITLE	GPC	DRAWING NO.	REV.
Package Drawing Contact: packagedrawings@atmel.com	**28M1,** 28-pad, 4 x 4 x 1.0 mm Body, Lead Pitch 0.45 mm, 2.4 x 2.4 mm Exposed Pad, Thermally Enhanced Plastic Very Thin Quad Flat No Lead Package (VQFN)	ZBV	28M1	B

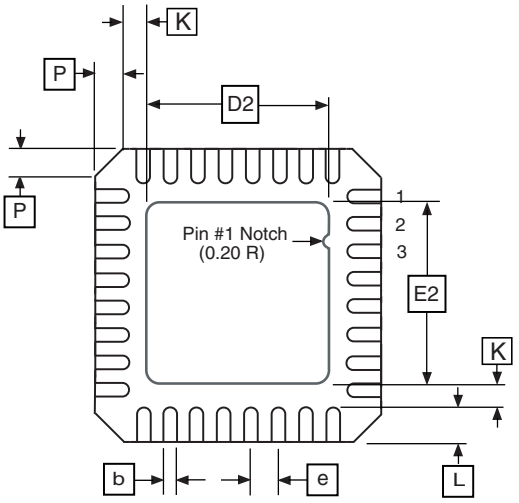
7.3 32M1-A



TOP VIEW



SIDE VIEW



BOTTOM VIEW

COMMON DIMENSIONS
(Unit of Measure = mm)

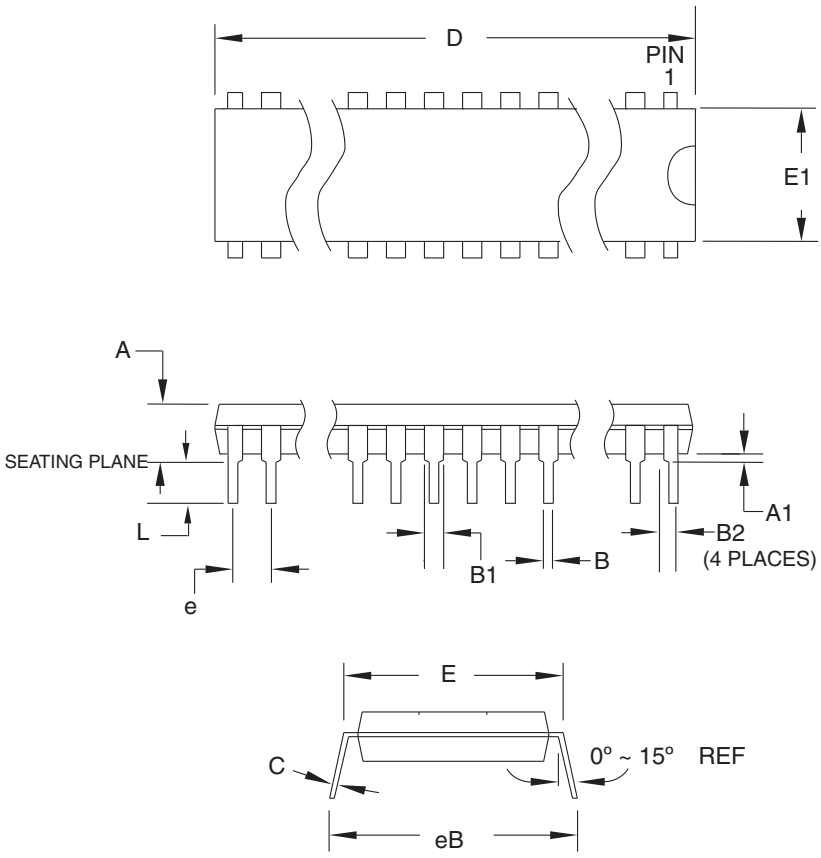
SYMBOL	MIN	NOM	MAX	NOTE
A	0.80	0.90	1.00	
A1	–	0.02	0.05	
A2	–	0.65	1.00	
A3		0.20 REF		
b	0.18	0.23	0.30	
D	4.90	5.00	5.10	
D1	4.70	4.75	4.80	
D2	2.95	3.10	3.25	
E	4.90	5.00	5.10	
E1	4.70	4.75	4.80	
E2	2.95	3.10	3.25	
e		0.50 BSC		
L	0.30	0.40	0.50	
P	–	–	0.60	
θ	–	–	12°	
K	0.20	–	–	

Note: JEDEC Standard MO-220, Fig. 2 (Anvil Singulation), VHHD-2.

5/25/06

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**32M1-A**, 32-pad, 5 x 5 x 1.0 mm Body, Lead Pitch 0.50 mm, 3.10 mm Exposed Pad, Micro Lead Frame Package (MLF)	32M1-A	E

7.4 28P3



COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	4.5724	
A1	0.508	–	–	
D	34.544	–	34.798	Note 1
E	7.620	–	8.255	
E1	7.112	–	7.493	Note 1
B	0.381	–	0.533	
B1	1.143	–	1.397	
B2	0.762	–	1.143	
L	3.175	–	3.429	
C	0.203	–	0.356	
eB	–	–	10.160	
e	2.540 TYP			

Note: 1. Dimensions D and E1 do not include mold Flash or Protrusion. Mold Flash or Protrusion shall not exceed 0.25 mm (0.010").

09/28/01

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**28P3**, 28-lead (0.300"/7.62 mm Wide) Plastic Dual Inline Package (PDIP)	28P3	B

8. Errata

8.1 Errata ATmega48

The revision letter in this section refers to the revision of the ATmega48 device.

8.1.1 Rev. D

- **Interrupts may be lost when writing the timer registers in the asynchronous timer**

1. **Interrupts may be lost when writing the timer registers in the asynchronous timer**
   The interrupt will be lost if a timer register that is synchronous timer clock is written when the asynchronous Timer/Counter register (TCNTx) is 0x00.

   Problem Fix/Workaround
   Always check that the asynchronous Timer/Counter register neither have the value 0xFF nor 0x00 before writing to the asynchronous Timer Control Register (TCCRx), asynchronous Timer Counter Register (TCNTx), or asynchronous Output Compare Register (OCRx).

8.1.2 Rev. C

- **Reading EEPROM when system clock frequency is below 900 kHz may not work**
- **Interrupts may be lost when writing the timer registers in the asynchronous timer**

1. **Reading EEPROM when system clock frequency is below 900 kHz may not work**
   Reading Data from the EEPROM at system clock frequency below 900 kHz may result in wrong data read.

   Problem Fix/Workaround
   Avoid using the EEPROM at clock frequency below 900 kHz.

2. **Interrupts may be lost when writing the timer registers in the asynchronous timer**
   The interrupt will be lost if a timer register that is synchronous timer clock is written when the asynchronous Timer/Counter register (TCNTx) is 0x00.

   Problem Fix/Workaround
   Always check that the asynchronous Timer/Counter register neither have the value 0xFF nor 0x00 before writing to the asynchronous Timer Control Register (TCCRx), asynchronous Timer Counter Register (TCNTx), or asynchronous Output Compare Register (OCRx).

8.1.3 Rev. B

- **Interrupts may be lost when writing the timer registers in the asynchronous timer**

1. **Interrupts may be lost when writing the timer registers in the asynchronous timer**
   The interrupt will be lost if a timer register that is synchronous timer clock is written when the asynchronous Timer/Counter register (TCNTx) is 0x00.

   Problem Fix/Workaround
   Always check that the asynchronous Timer/Counter register neither have the value 0xFF nor 0x00 before writing to the asynchronous Timer Control Register (TCCRx), asynchronous Timer Counter Register (TCNTx), or asynchronous Output Compare Register (OCRx).

8.1.4 Rev A

- **Part may hang in reset**
- **Wrong values read after Erase Only operation**
- **Watchdog Timer Interrupt disabled**
- **Start-up time with Crystal Oscillator is higher than expected**
- **High Power Consumption in Power-down with External Clock**
- **Asynchronous Oscillator does not stop in Power-down**
- **Interrupts may be lost when writing the timer registers in the asynchronous timer**

1. **Part may hang in reset**

   Some parts may get stuck in a reset state when a reset signal is applied when the internal reset state-machine is in a specific state. The internal reset state-machine is in this state for approximately 10 ns immediately before the part wakes up after a reset, and in a 10 ns window when altering the system clock prescaler. The problem is most often seen during In-System Programming of the device. There are theoretical possibilities of this happening also in run-mode. The following three cases can trigger the device to get stuck in a reset-state:

   - Two succeeding resets are applied where the second reset occurs in the 10ns window before the device is out of the reset-state caused by the first reset.

   - A reset is applied in a 10 ns window while the system clock prescaler value is updated by software.

   - Leaving SPI-programming mode generates an internal reset signal that can trigger this case.

   The two first cases can occur during normal operating mode, while the last case occurs only during programming of the device.

   Problem Fix/Workaround

   The first case can be avoided during run-mode by ensuring that only one reset source is active. If an external reset push button is used, the reset start-up time should be selected such that the reset line is fully debounced during the start-up time.

   The second case can be avoided by not using the system clock prescaler.

   The third case occurs during In-System programming only. It is most frequently seen when using the internal RC at maximum frequency.

   If the device gets stuck in the reset-state, turn power off, then on again to get the device out of this state.

2. **Wrong values read after Erase Only operation**

   At supply voltages below 2.7 V, an EEPROM location that is erased by the Erase Only operation may read as programmed (0x00).

   Problem Fix/Workaround

   If it is necessary to read an EEPROM location after Erase Only, use an Atomic Write operation with 0xFF as data in order to erase a location. In any case, the Write Only operation can be used as intended. Thus no special considerations are needed as long as the erased location is not read before it is programmed.

3. **Watchdog Timer Interrupt disabled**

If the watchdog timer interrupt flag is not cleared before a new timeout occurs, the watchdog will be disabled, and the interrupt flag will automatically be cleared. This is only applicable in interrupt only mode. If the Watchdog is configured to reset the device in the watchdog time-out following an interrupt, the device works correctly.

Problem fix / Workaround

Make sure there is enough time to always service the first timeout event before a new watchdog timeout occurs. This is done by selecting a long enough time-out period.

4. **Start-up time with Crystal Oscillator is higher than expected**

   The clock counting part of the start-up time is about 2 times higher than expected for all start-up periods when running on an external Crystal. This applies only when waking up by reset. Wake-up from power down is not affected. For most settings, the clock counting parts is a small fraction of the overall start-up time, and thus, the problem can be ignored. The exception is when using a very low frequency crystal like for instance a 32 kHz clock crystal.

   Problem fix / Workaround

   No known workaround.

5. **High Power Consumption in Power-down with External Clock**

   The power consumption in power down with an active external clock is about 10 times higher than when using internal RC or external oscillators.

   Problem fix / Workaround

   Stop the external clock when the device is in power down.

6. **Asynchronous Oscillator does not stop in Power-down**

   The Asynchronous oscillator does not stop when entering power down mode. This leads to higher power consumption than expected.

   Problem fix / Workaround

   Manually disable the asynchronous timer before entering power down.

7. **Interrupts may be lost when writing the timer registers in the asynchronous timer**

   The interrupt will be lost if a timer register that is synchronous timer clock is written when the asynchronous Timer/Counter register (TCNTx) is 0x00.

   Problem Fix/Workaround

   Always check that the asynchronous Timer/Counter register neither have the value 0xFF nor 0x00 before writing to the asynchronous Timer Control Register (TCCRx), asynchronous Timer Counter Register (TCNTx), or asynchronous Output Compare Register (OCRx).

8.2 Errata ATmega88

The revision letter in this section refers to the revision of the ATmega88 device.

8.2.1 Rev. D

- **Interrupts may be lost when writing the timer registers in the asynchronous timer**

1. **Interrupts may be lost when writing the timer registers in the asynchronous timer**
   The interrupt will be lost if a timer register that is synchronous timer clock is written when the asynchronous Timer/Counter register (TCNTx) is 0x00.

   Problem Fix/Workaround
   Always check that the asynchronous Timer/Counter register neither have the value 0xFF nor 0x00 before writing to the asynchronous Timer Control Register (TCCRx), asynchronous Timer Counter Register (TCNTx), or asynchronous Output Compare Register (OCRx).

8.2.2 Rev. B/C

Not sampled.

8.2.3 Rev. A

- **Writing to EEPROM does not work at low Operating Voltages**
- **Part may hang in reset**
- **Interrupts may be lost when writing the timer registers in the asynchronous timer**

1. **Writing to EEPROM does not work at low operating voltages**
   Writing to the EEPROM does not work at low voltages.

   Problem Fix/Workaround
   Do not write the EEPROM at voltages below 4.5 Volts.
   This will be corrected in rev. B.

2. **Part may hang in reset**
   Some parts may get stuck in a reset state when a reset signal is applied when the internal reset state-machine is in a specific state. The internal reset state-machine is in this state for approximately 10 ns immediately before the part wakes up after a reset, and in a 10 ns window when altering the system clock prescaler. The problem is most often seen during In-System Programming of the device. There are theoretical possibilities of this happening also in run-mode. The following three cases can trigger the device to get stuck in a reset-state:

   - Two succeeding resets are applied where the second reset occurs in the 10ns window before the device is out of the reset-state caused by the first reset.

   - A reset is applied in a 10 ns window while the system clock prescaler value is updated by software.

   - Leaving SPI-programming mode generates an internal reset signal that can trigger this case.

   The two first cases can occur during normal operating mode, while the last case occurs only during programming of the device.

Problem Fix/Workaround

The first case can be avoided during run-mode by ensuring that only one reset source is active. If an external reset push button is used, the reset start-up time should be selected such that the reset line is fully debounced during the start-up time.

The second case can be avoided by not using the system clock prescaler.

The third case occurs during In-System programming only. It is most frequently seen when using the internal RC at maximum frequency.

If the device gets stuck in the reset-state, turn power off, then on again to get the device out of this state.

3. **Interrupts may be lost when writing the timer registers in the asynchronous timer**

The interrupt will be lost if a timer register that is synchronous timer clock is written when the asynchronous Timer/Counter register (TCNTx) is 0x00.

Problem Fix/Workaround

Always check that the asynchronous Timer/Counter register neither have the value 0xFF nor 0x00 before writing to the asynchronous Timer Control Register (TCCRx), asynchronous Timer Counter Register (TCNTx), or asynchronous Output Compare Register (OCRx).

8.3 Errata ATmega168

The revision letter in this section refers to the revision of the ATmega168 device.

8.3.1 Rev C

- **Interrupts may be lost when writing the timer registers in the asynchronous timer**

1. **Interrupts may be lost when writing the timer registers in the asynchronous timer**

The interrupt will be lost if a timer register that is synchronous timer clock is written when the asynchronous Timer/Counter register (TCNTx) is 0x00.

Problem Fix/Workaround

Always check that the asynchronous Timer/Counter register neither have the value 0xFF nor 0x00 before writing to the asynchronous Timer Control Register (TCCRx), asynchronous Timer Counter Register (TCNTx), or asynchronous Output Compare Register (OCRx).

8.3.2 Rev B

- **Part may hang in reset**
- **Interrupts may be lost when writing the timer registers in the asynchronous timer**

1. **Part may hang in reset**

Some parts may get stuck in a reset state when a reset signal is applied when the internal reset state-machine is in a specific state. The internal reset state-machine is in this state for approximately 10 ns immediately before the part wakes up after a reset, and in a 10 ns window when altering the system clock prescaler. The problem is most often seen during In-System Programming of the device. There are theoretical possibilities of this happening also in run-mode. The following three cases can trigger the device to get stuck in a reset-state:

- Two succeeding resets are applied where the second reset occurs in the 10ns window before the device is out of the reset-state caused by the first reset.

- A reset is applied in a 10 ns window while the system clock prescaler value is updated by software.

- Leaving SPI-programming mode generates an internal reset signal that can trigger this case.

The two first cases can occur during normal operating mode, while the last case occurs only during programming of the device.

Problem Fix/Workaround

The first case can be avoided during run-mode by ensuring that only one reset source is active. If an external reset push button is used, the reset start-up time should be selected such that the reset line is fully debounced during the start-up time.

The second case can be avoided by not using the system clock prescaler.

The third case occurs during In-System programming only. It is most frequently seen when using the internal RC at maximum frequency.

If the device gets stuck in the reset-state, turn power off, then on again to get the device out of this state.

2. **Interrupts may be lost when writing the timer registers in the asynchronous timer**

The interrupt will be lost if a timer register that is synchronous timer clock is written when the asynchronous Timer/Counter register (TCNTx) is 0x00.

Problem Fix/Workaround

Always check that the asynchronous Timer/Counter register neither have the value 0xFF nor 0x00 before writing to the asynchronous Timer Control Register (TCCRx), asynchronous Timer Counter Register (TCNTx), or asynchronous Output Compare Register (OCRx).

8.3.3 Rev A

- **Wrong values read after Erase Only operation**
- **Part may hang in reset**
- **Interrupts may be lost when writing the timer registers in the asynchronous timer**

1. **Wrong values read after Erase Only operation**

At supply voltages below 2.7 V, an EEPROM location that is erased by the Erase Only operation may read as programmed (0x00).

Problem Fix/Workaround

If it is necessary to read an EEPROM location after Erase Only, use an Atomic Write operation with 0xFF as data in order to erase a location. In any case, the Write Only operation can be used as intended. Thus no special considerations are needed as long as the erased location is not read before it is programmed.

2. **Part may hang in reset**

Some parts may get stuck in a reset state when a reset signal is applied when the internal reset state-machine is in a specific state. The internal reset state-machine is in this state for approximately 10 ns immediately before the part wakes up after a reset, and in a 10 ns window when altering the system clock prescaler. The problem is most often seen during In-System Programming of the device. There are theoretical possibilities of this happening also in run-mode. The following three cases can trigger the device to get stuck in a reset-state:

- Two succeeding resets are applied where the second reset occurs in the 10ns window before the device is out of the reset-state caused by the first reset.

- A reset is applied in a 10 ns window while the system clock prescaler value is updated by software.

- Leaving SPI-programming mode generates an internal reset signal that can trigger this case.

The two first cases can occur during normal operating mode, while the last case occurs only during programming of the device.

Problem Fix/Workaround

The first case can be avoided during run-mode by ensuring that only one reset source is active. If an external reset push button is used, the reset start-up time should be selected such that the reset line is fully debounced during the start-up time.

The second case can be avoided by not using the system clock prescaler.

The third case occurs during In-System programming only. It is most frequently seen when using the internal RC at maximum frequency.

If the device gets stuck in the reset-state, turn power off, then on again to get the device out of this state.

2. **Interrupts may be lost when writing the timer registers in the asynchronous timer**

The interrupt will be lost if a timer register that is synchronous timer clock is written when the asynchronous Timer/Counter register (TCNTx) is 0x00.

Problem Fix/Workaround

Always check that the asynchronous Timer/Counter register neither have the value 0xFF nor 0x00 before writing to the asynchronous Timer Control Register (TCCRx), asynchronous Timer Counter Register (TCNTx), or asynchronous Output Compare Register (OCRx).

9. Datasheet Revision History

Please note that the referring page numbers in this section are referred to this document. The referring revision in this section are referring to the document revision.

9.1 Rev. 2545R-07/09

1. Updated “Errata” on page 357.
2. Updated the last page with Atmel’s new addresses.

9.2 Rev. 2545Q-06/09

1. Removed the heading “About”. The subsections of this sectionis now separate sections, “Resources”, “Data Retention” and “About Code Examples”
2. Updated “Ordering Information” on page 349.

9.3 Rev. 2545P-02/09

1. Removed Power-off slope rate from Table 28-3 on page 306.

9.4 Rev. 2545O-02/09

1. Changed minimum Power-on Reset Threshold Voltage (falling) to 0.05V in Table 28-3 on page 306.
2. Removed section “Power-on slope rate” from “System and Reset Characteristics” on page 306.

9.5 Rev. 2545N-01/09

1. Updated “Features” on page 1 and added the note “Not recommended for new designs”.
2. Merged the sections Resources, Data Retention and About Code Examples under one common section, “Resources” on page 7.
3. Updated Figure 8-4 on page 34.
4. Updated “System Clock Prescaler” on page 35.
5. Updated “Alternate Functions of Port B” on page 77.
6. Added section “” on page 306.
7. Updated “Pin Thresholds and Hysteresis” on page 329.

9.6 Rev. 2545M-09/07

1. Added “Data Retention” on page 7.
2. Updated “ADC Characteristics” on page 310.
3. “Preliminary“ removed through the datasheet.

9.7 Rev. 2545L-08/07

1. Updated “Features” on page 1.
2. Updated code example in “MCUCR – MCU Control Register” on page 63.
3. Updated “System and Reset Characteristics” on page 306.
4. Updated Note in Table 8-3 on page 29, Table 8-5 on page 30, Table 8-8 on page 33, Table 8-10 on page 33.

9.8 Rev. 2545K-04/07

1. Updated “Interrupts” on page 55.
2. Updated“Errata ATmega48” on page 357 .
3. Changed description in “Analog-to-Digital Converter” on page 243.

9.9 Rev. 2545J-12/06

1. Updated “Features” on page 1.
2. Updated Table 1-1 on page 2.
3. Updated “Ordering Information” on page 349.
4. Updated “Packaging Information” on page 353.

9.10 Rev. 2545I-11/06

1. Updated “Features” on page 1.
2. Updated Features in “2-wire Serial Interface” on page 208.
3. Fixed typos in Table 28-3 on page 306.

9.11 Rev. 2545H-10/06

1. Updated typos.
2. Updated “Features” on page 1.
3. Updated “Calibrated Internal RC Oscillator” on page 32.
4. Updated “System Control and Reset” on page 44.
5. Updated “Brown-out Detection” on page 46.
6. Updated “Fast PWM Mode” on page 120.
7. Updated bit description in “TCCR1C – Timer/Counter1 Control Register C” on page 132.

8. Updated code example in “SPI – Serial Peripheral Interface” on page 160.
9. Updated Table 14-3 on page 100, Table 14-6 on page 101, Table 14-8 on page 102, Table 15-2 on page 129, Table 15-3 on page 130, Table 15-4 on page 131, Table 17-3 on page 153, Table 17-6 on page 154, Table 17-8 on page 155, and Table 27-5 on page 286.
10. Added Note to Table 25-1 on page 264, Table 26-5 on page 278, and Table 27-17 on page 299.
11. Updated “Setting the Boot Loader Lock Bits by SPM” on page 276.
12. Updated “Signature Bytes” on page 287
13. Updated “Electrical Characteristics” on page 302.
14. Updated “Errata” on page 357.

9.12 Rev. 2545G-06/06

1. Added Addresses in Registers.
2. Updated “Calibrated Internal RC Oscillator” on page 32.
3. Updated Table 8-12 on page 34, Table 9-1 on page 38, Table 10-1 on page 53, Table 13-3 on page 77.
4. Updated “ADC Noise Reduction Mode” on page 39.
5. Updated note for Table 9-2 on page 42.
6. Updatad “Bit 2 - PRSPI: Power Reduction Serial Peripheral Interface” on page 43.
7. Updated “TCCR0B – Timer/Counter Control Register B” on page 103.
8. Updated “Fast PWM Mode” on page 120.
9. Updated “Asynchronous Operation of Timer/Counter2” on page 150.
10. Updated “SPI – Serial Peripheral Interface” on page 160.
11. Updated “UCSRnA – USART MSPIM Control and Status Register n A” on page 205.
12. Updated note in “Bit Rate Generator Unit” on page 215.
13. Updated “Bit 6 – ACBG: Analog Comparator Bandgap Select” on page 241.
14. Updated Features in “Analog-to-Digital Converter” on page 243.
15. Updated “Prescaling and Conversion Timing” on page 246.
16. Updated “Limitations of debugWIRE” on page 260.
17 Added Table 28-1 on page 305.
18. Updated Figure 15-7 on page 121, Figure 29-45 on page 338.
19. Updated rev. A in “Errata ATmega48” on page 357.
20. Added rev. C and D in “Errata ATmega48” on page 357.

9.13 Rev. 2545F-05/05

1. Added Section 3. “Resources” on page 7
2. Update Section 8.6 “Calibrated Internal RC Oscillator” on page 32.
3. Updated Section 27.8.3 “Serial Programming Instruction set” on page 299.
4. Table notes in Section 28.2 “DC Characteristics” on page 302 updated.
5. Updated Section 34. “Errata” on page 357.

9.14 Rev. 2545E-02/05

1. MLF-package alternative changed to “Quad Flat No-Lead/Micro Lead Frame Package QFN/MLF”.
2. Updated “EECR – The EEPROM Control Register” on page 21.
3. Updated “Calibrated Internal RC Oscillator” on page 32.
4. Updated “External Clock” on page 34.
5. Updated Table 28-3 on page 306, Table 28-6 on page 308, Table 28-2 on page 305and Table 27-16 on page 299
6. Added “Pin Change Interrupt Timing” on page 65
7. Updated “8-bit Timer/Counter Block Diagram” on page 89.
8. Updated “SPMCSR – Store Program Memory Control and Status Register” on page 266.
9. Updated “Enter Programming Mode” on page 290.
10. Updated “DC Characteristics” on page 302.
11. Updated “Ordering Information” on page 349.
12. Updated “Errata ATmega88” on page 360 and “Errata ATmega168” on page 361.

9.15 Rev. 2545D-07/04

1. Updated instructions used with WDTCSR in relevant code examples.
2. Updated Table 8-5 on page 30, Table 28-4 on page 306, Table 26-9 on page 281, and Table 26-11 on page 282.
3. Updated “System Clock Prescaler” on page 35.
4. Moved “TIMSK2 – Timer/Counter2 Interrupt Mask Register” on page17.11.6 and “TIFR2 – Timer/Counter2 Interrupt Flag Register” on page17.11.7 to “Register Description” on page 152.
5. Updated cross-reference in “Electrical Interconnection” on page 209.
6. Updated equation in “Bit Rate Generator Unit” on page 215.
7. Added “Page Size” on page 288.
8. Updated “Serial Programming Algorithm” on page 298.
9. Updated Ordering Information for “ATmega168” on page 351.
10. Updated “Errata ATmega88” on page 360 and “Errata ATmega168” on page 361.
11. Updated equation in “Bit Rate Generator Unit” on page 215.

9.16 Rev. 2545C-04/04

1. Speed Grades changed: 12MHz to 10MHz and 24MHz to 20MHz
2. Updated “Speed Grades” on page 304.
3. Updated “Ordering Information” on page 349.
4. Updated “Errata ATmega88” on page 360.

9.17 Rev. 2545B-01/04

1. Added PDIP to “I/O and Packages”, updated “Speed Grade” and Power Consumption Estimates in 35.“Features” on page 1.
2. Updated “Stack Pointer” on page 12 with RAMEND as recommended Stack Pointer value.
3. Added section “Power Reduction Register” on page 40 and a note regarding the use of the PRR bits to 2-wire, Timer/Counters, USART, Analog Comparator and ADC sections.
4. Updated “Watchdog Timer” on page 48.
5. Updated Figure 15-2 on page 129 and Table 15-3 on page 130.
6. Extra Compare Match Interrupt OCF2B added to features in section “8-bit Timer/Counter2 with PWM and Asynchronous Operation” on page 139
7. Updated Table 9-1 on page 38, Table 23-5 on page 258, Table 27-4 to Table 27-7 on page 285 to 287 and Table 23-1 on page 248. Added note 2 to Table 27-1 on page 284. Fixed typo in Table 12-1 on page 66.
8. Updated whole “Typical Characteristics” on page 314.
9. Added item 2 to 5 in “Errata ATmega48” on page 357.
10. Renamed the following bits:
    - SPMEN to SELFPRGEN
    - PSR2 to PSRASY
    - PSR10 to PSRSYNC
    - Watchdog Reset to Watchdog System Reset
11. Updated C code examples containing old IAR syntax.
12. Updated BLBSET description in “SPMCSR – Store Program Memory Control and Status Register” on page 282.

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

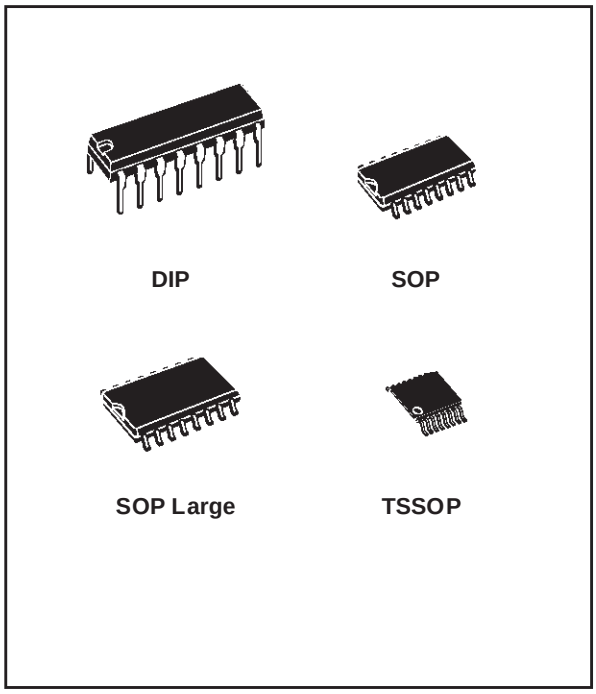
Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature

ST3232

3 TO 5.5V, LOW POWER, UP TO 400KBPS, RS-232 DRIVERS AND RECEIVERS

- 300µA SUPPLY CURRENT
- 300Kbps MINIMUM GUARENTEED DATA RATE
- 6V/µs MINIMUM GUARANTEED SLEW RATE
- MEET EIA/TIA-232 SPECIFICATIONS DOWN TO 3V
- AVAILABLE IN DIP-16, SO-16, SO-16 LARGE AND TSSOP16



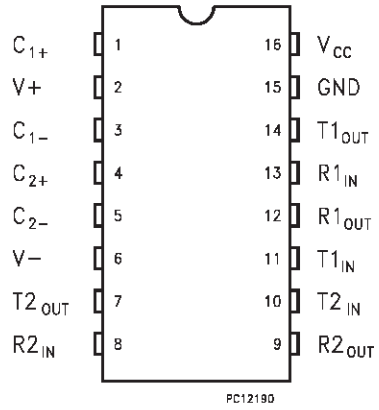
DESCRIPTION

The ST3232 is a 3V powered EIA/TIA-232 and V.28/V.24 communication interface with low power requirements, high data-rate capabilities. ST3232 has a proprietary low dropout transmitter output stage providing true RS-232 performance from 3 to 5.5V supplies. The device requires only four small 0.1µF standard external capacitors for operations from 3V supply.
The ST3232 has two receivers and two drivers.
The device is guaranteed to run at data rates of 250Kbps while maintaining RS-232 output levels.
Typical applications are Notebook, Subnotebook and Palmtop Computers, Battery Powered Equipment, Hand-Held Equipment, Peripherals and Printers.

ORDERING CODES

Type	Temperature Range	Package	Comments
ST3232CN	0 to 70 °C	DIP-16	25parts per tube / 40tube per box
ST3232BN	-40 to 85 °C	DIP-16	25parts per tube / 40tube per box
ST3232CD	0 to 70 °C	SO-16 (Tube)	50parts per tube / 20tube per box
ST3232BD	-40 to 85 °C	SO-16 (Tube)	50parts per tube / 20tube per box
ST3232CDR	0 to 70 °C	SO-16 (Tape & Reel)	2500 parts per reel
ST3232BDR	-40 to 85 °C	SO-16 (Tape & Reel)	2500 parts per reel
ST3232CW	0 to 70 °C	SO-16 Large (Tube)	49parts per tube / 25tube per box
ST3232BW	-40 to 85 °C	SO-16 Large (Tube)	49parts per tube / 25tube per box
ST3232CWR	0 to 70 °C	SO-16 Large (Tape & Reel)	1000 parts per reel
ST3232BWR	-40 to 85 °C	SO-16 Large (Tape & Reel)	1000 parts per reel
ST3232CTR	0 to 70 °C	TSSOP16 (Tape & Reel)	2500 parts per reel
ST3232BTR	-40 to 85 °C	TSSOP16 (Tape & Reel)	2500 parts per reel

PIN CONFIGURATION



PIN DESCRIPTION

PIN N°	SYMBOL	NAME AND FUNCTION
1	C_1+	Positive Terminal for the first Charge Pump Capacitor
2	V+	Doubled Voltage Terminal
3	C_1-	Negative Terminal for the first Charge Pump Capacitor
4	C_2+	Positive Terminal for the second Charge Pump Capacitor
5	C_2-	Negative Terminal for the second Charge Pump Capacitor
6	V-	Inverted Voltage Terminal
7	$T2_{OUT}$	Second Transmitter Output Voltage
8	$R2_{IN}$	Second Receiver Input Voltage
9	$R2_{OUT}$	Second Receiver Output Voltage
10	$T2_{IN}$	Second Transmitter Input Voltage
11	$T1_{IN}$	First Transmitter Input Voltage
12	$R1_{OUT}$	First Receiver Output Voltage
13	$R1_{IN}$	First Receiver Input Voltage
14	$T1_{OUT}$	First Transmitter Output Voltage
15	GND	Ground
16	V_{CC}	Supply Voltage

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_{CC}	Supply Voltage	-0.3 to 6	V
V+	Doubled Voltage Terminal	(V_{CC} - 0.3) to 7	V
V-	Inverted Voltage Terminal	0.3 to -7	V
V+ +|V-|		13	V
T_{IN}	Transmitter Input Voltage Range	-0.3 to 6	V
R_{IN}	Receiver Input Voltage Range	± 25	V
T_{OUT}	Transmitter Output Voltage Range	± 13.2	V
R_{OUT}	Receiver Output Voltage Range	-0.3 to (V_{CC} + 0.3)	V
t_{SHORT}	Transmitter Output Short to GND Time	Continuous	

Absolute Maximum Ratings are those values beyond which damage to the device may occur. Functional operation under these condition is not implied. V+ and V- can have a maximum magnitude of +7V, but their absolute addition can not exceed 13 V.

ELECTRICAL CHARACTERISTICS

(C_1 - C_4 = 0.1µF, V_{CC} = 3V to 5.5V, T_A = -40 to 85°C, unless otherwise specified.
Typical values are referred to T_A = 25°C)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
I_{SUPPLY}	V_{CC} Power Supply Current	No Load V_{CC} = 3V ±10% T_A = 25°C		0.3	1	mA
		No Load V_{CC} = 5V ±10% T_A = 25°C		1	2	mA

LOGIC INPUT ELECTRICAL CHARACTERISTICS

(C_1 - C_4 = 0.1µF, V_{CC} = 3V to 5.5V, T_A = -40 to 85°C, unless otherwise specified.
Typical values are referred to T_A = 25°C)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_{TIL}	Input Logic Threshold Low	T-IN (Note 1)			0.8	V
V_{TIH}	Input Logic Threshold High	V_{CC} = 3.3V V_{CC} = 5V	2 2.4			V V
I_{IL}	Input Leakage Current	T-IN		± 0.01	± 1	µA

Note 1: Transmitter input hysteresis is typically 250mV

TRANSMITTER ELECTRICAL CHARACTERISTICS

(C_1 - C_4 = 0.1µF tested at V_{CC} = 3V to 5.5V, T_A = -40 to 85°C, unless otherwise specified.
Typical values are referred to T_A = 25°C)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_{TOUT}	Output Voltage Swing	All Transmitter outputs are loaded with 3KΩ to GND	± 5	± 5.4		V
R_{TOUT}	Transmitter Output Resistance	V_{CC} = V+ = V- = 0V V_{OUT} = ± 2V	300	10M		Ω
I_{TSC}	Output Short Circuit Current	V_{CC} = 3V to 5V V_{OUT} = ± 12V			± 60	mA

RECEIVER ELECTRICAL CHARACTERISTICS

(C_1 - C_4 = 0.1µF tested at V_{CC} = 3V to 5.5V, T_A = -40 to 85°C, unless otherwise specified.
Typical values are referred to T_A = 25°C)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_{RIN}	Receiver Input Voltage Operating Range		-25		25	V
V_{RIL}	RS-232 Input Threshold Low	T_A = 25°C V_{CC} = 3.3V T_A = 25°C V_{CC} = 5V	0.6 0.8	1.2 1.5		V V
V_{RIH}	RS-232 Input Threshold High	T_A = 25°C V_{CC} = 3.3V T_A = 25°C V_{CC} = 5V		1.5 1.8	2.4 2.4	V V
V_{RIHYS}	Input Hysteresis			0.3		V
R_{RIN}	Input Resistance	T_A = 25°C	3	5	7	KΩ
V_{ROL}	TTL/CMOS Output Voltage Low	I_{OUT} = 1.6mA			0.4	V
V_{ROH}	TTL/CMOS Output Voltage High	I_{OUT} = -1mA	V_{CC}-0.6	V_{CC}-0.1		V

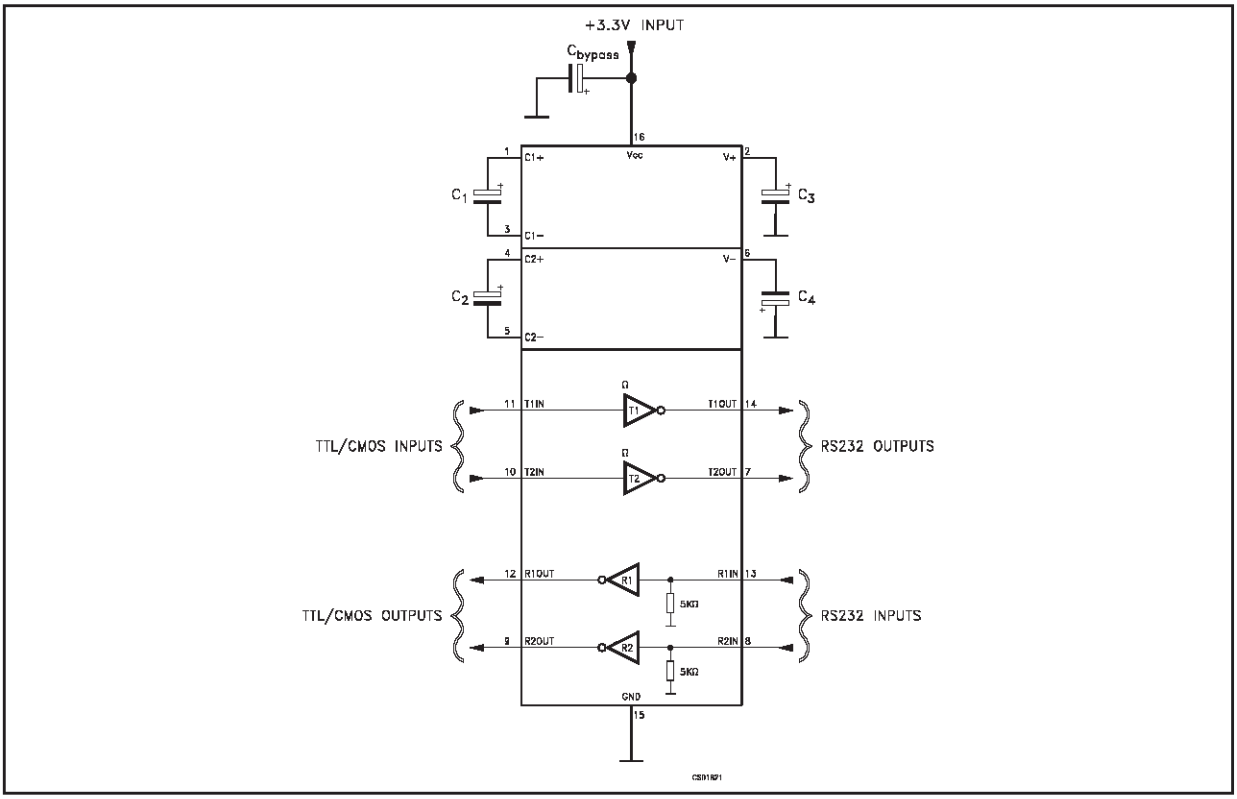
TIMING CHARACTERISTICS

(C_1 - C_4 = 0.1µF, V_{CC} = 3V to 5.5V, T_A = -40 to 85°C, unless otherwise specified.
Typical values are referred to T_A = 25°C)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
D_R	Data Transfer Rate	R_L = 3KΩ C_{L2}= 1000pF one trasmitter switching	300	400		Kbps
t_{PHLR} t_{PLHR}	Propagation Delay Input to Output	R_{XIN} = R_{XOUT} C_L = 150pF		0.2		µs
$\|t_{PHLT} - t_{THL}\|$	Transmitter Propagation Delay Difference	(Note 1)		100		ns
$\|t_{PHLR} - t_{THR}\|$	Receiver Propagation Delay Difference			50		ns
S_{RT}	Trnasition Slew Rate	T_A = 25°C R_L = 3KΩ to 7KΩ V_{CC} = 3.3V measured from +3V to -3V or -3V to +3V C_L = 150pF to 1000pF C_L = 150pF to 2500pF	 6 4		 30 30	 V/µs V/µs

Transmitter Skew is measured at the transmitter zero cross points

APPLICATION CIRCUITS



CAPACITANCE VALUE (µF)

V_{CC}	C1	C2	C3	C4	Cbypass
3.0 to 3.6	0.1	0.1	0.1	0.1	0.1
4.5 to 5.5	0.047	0.33	0.33	0.33	0.33

TYPICAL PERFORMANCE CHARACTERISTICS (unless otherwise specified T_j = 25°C)

Figure 1 : Driver Voltage Transfer Characteristics for Trasmitter Inputs

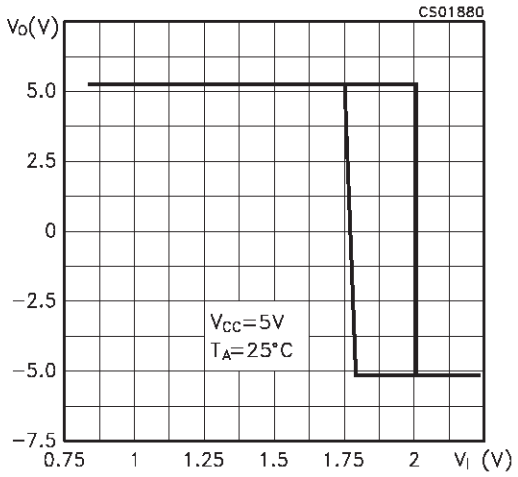


Figure 2 : Driver Voltage Transfer Characteristics for Receiver Inputs

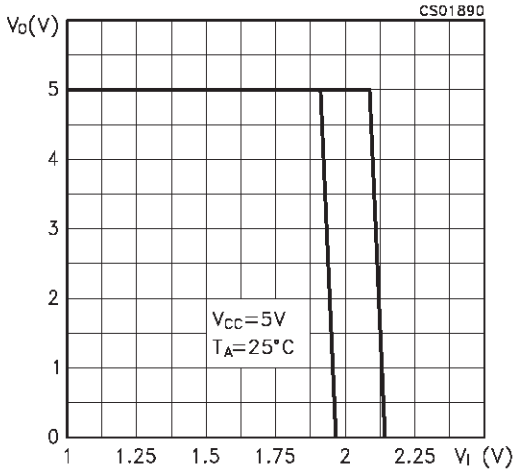


Figure 3 : Output Current vs Output Low Voltage

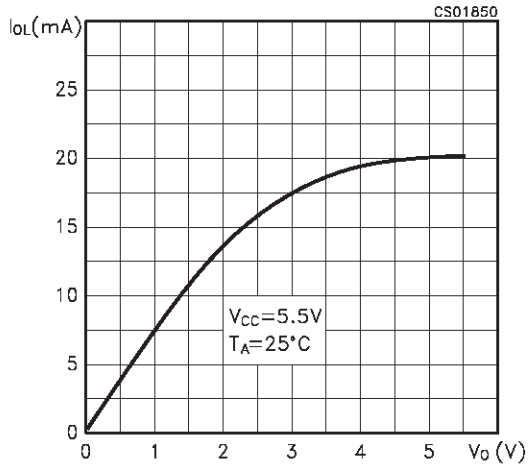


Figure 4 : Output Current vs Output Low Voltage

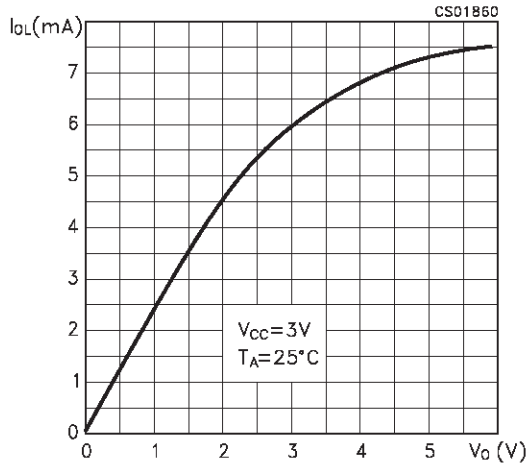


Figure 5 : Output Current vs Output High Voltage

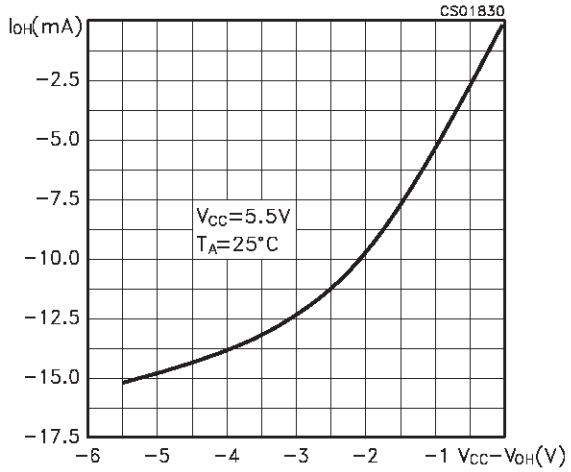


Figure 6 : Output Current vs Output High Voltage

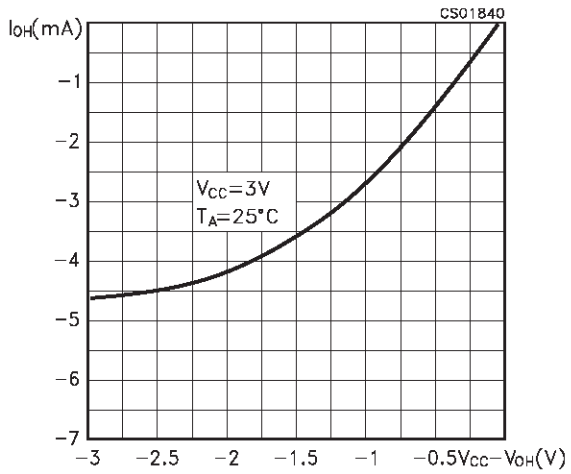
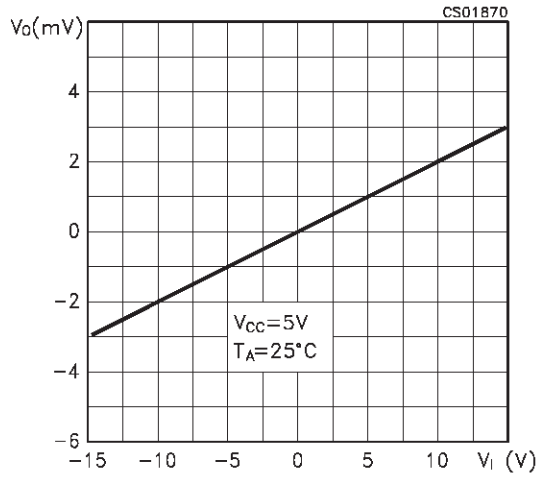
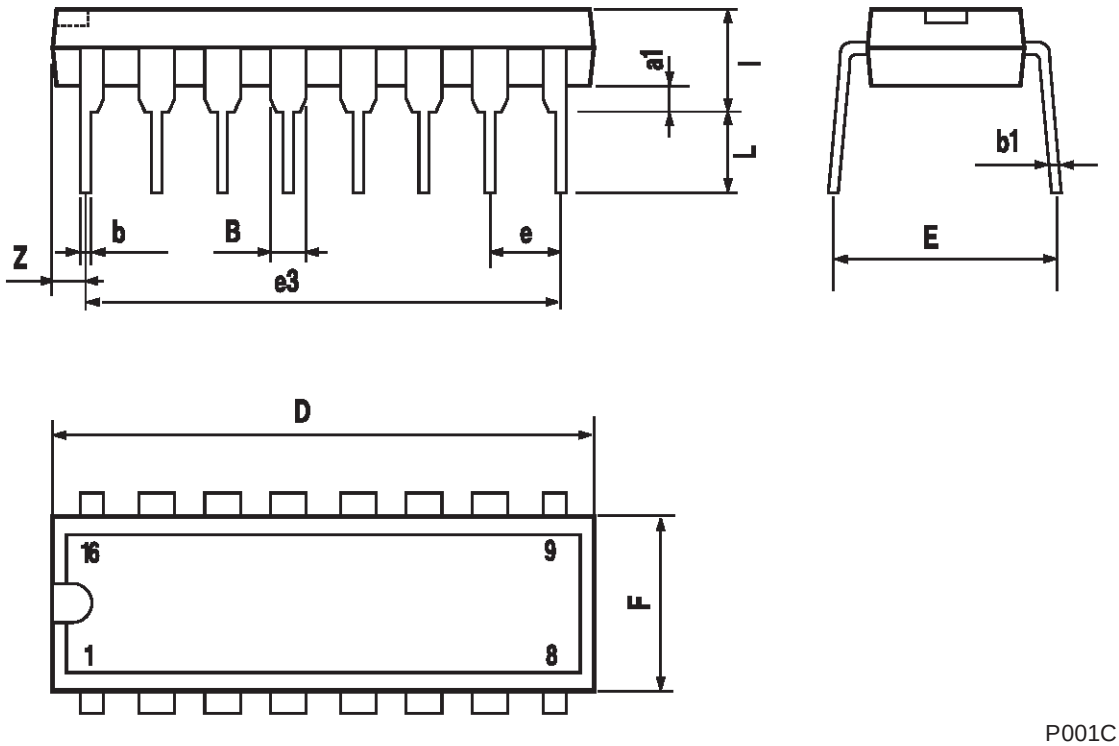


Figure 7 : Receiver Input Resistance



Plastic DIP-16 (0.25) MECHANICAL DATA

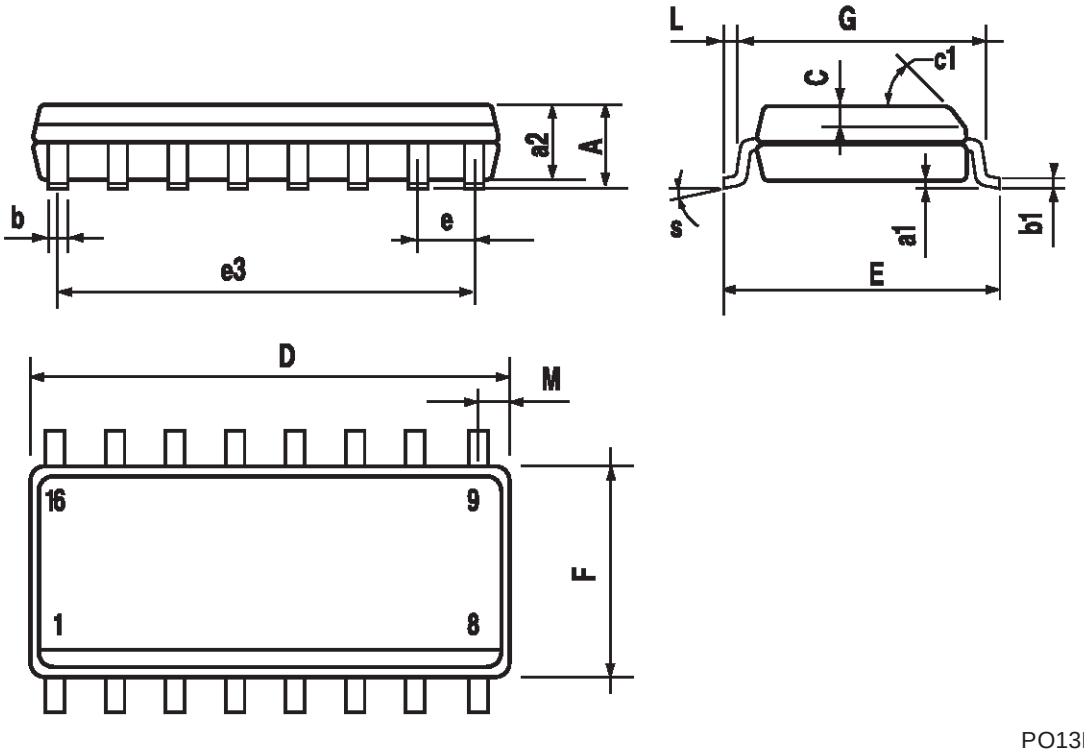
DIM.	mm.			inch		
	MIN.	TYP	MAX.	MIN.	TYP.	MAX.
a1	0.51			0.020		
B	0.77		1.65	0.030		0.065
b		0.5			0.020	
b1		0.25			0.010	
D			20			0.787
E		8.5			0.335	
e		2.54			0.100	
e3		17.78			0.700	
F			7.1			0.280
I			5.1			0.201
L		3.3			0.130	
Z			1.27			0.050



P001C

SO-16 MECHANICAL DATA

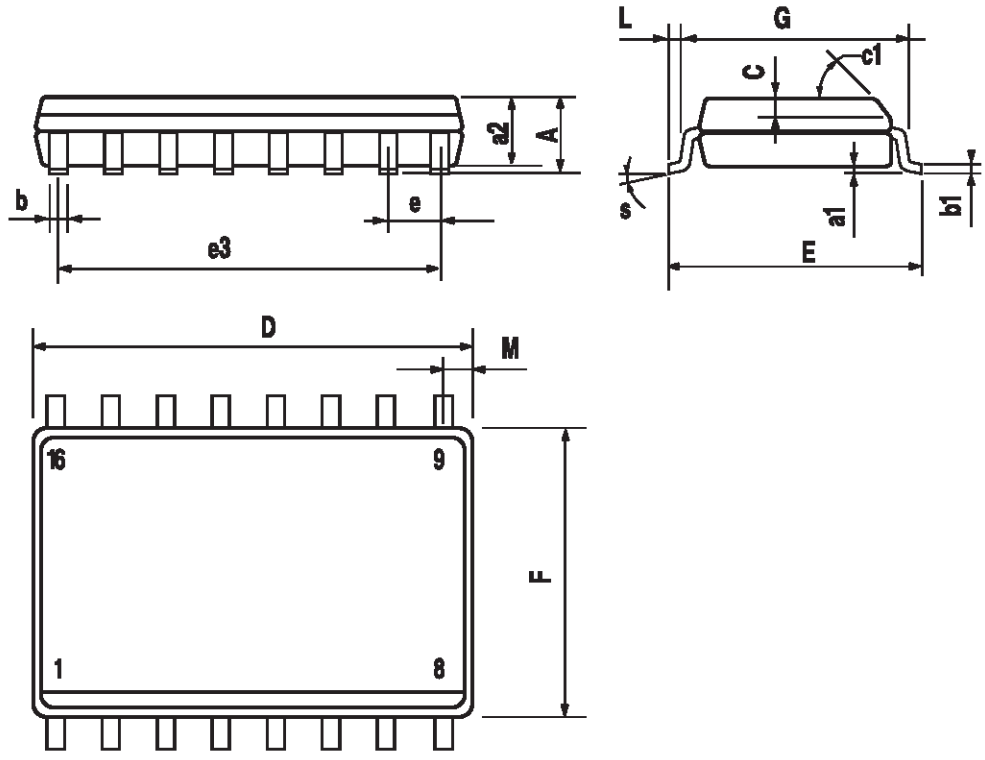
DIM.	mm.			inch		
	MIN.	TYP	MAX.	MIN.	TYP.	MAX.
A			1.75			0.068
a1	0.1		0.2	0.003		0.007
a2			1.65			0.064
b	0.35		0.46	0.013		0.018
b1	0.19		0.25	0.007		0.010
C		0.5			0.019	
c1	45° (typ.)					
D	9.8		10	0.385		0.393
E	5.8		6.2	0.228		0.244
e		1.27			0.050	
e3		8.89			0.350	
F	3.8		4.0	0.149		0.157
G	4.6		5.3	0.181		0.208
L	0.5		1.27	0.019		0.050
M			0.62			0.024
S	8° (max.)					



PO13H

SO-16L MECHANICAL DATA

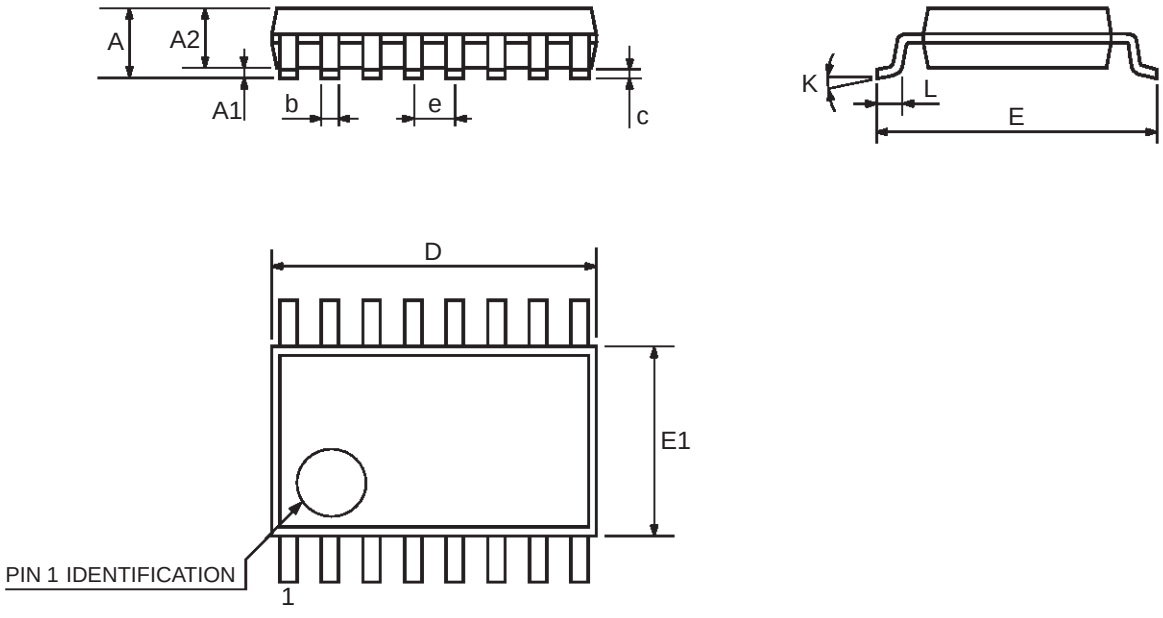
DIM.	mm.			inch		
	MIN.	TYP	MAX.	MIN.	TYP.	MAX.
A			2.65			0.104
a1	0.1		0.2	0.004		0.008
a2			2.45			0.096
b	0.35		0.49	0.014		0.019
b1	0.23		0.32	0.009		0.012
C		0.5			0.020	
c1	45° (typ.)					
D	10.1		10.5	0.397		0.413
E	10.0		10.65	0.393		0.419
e		1.27			0.050	
e3		8.89			0.350	
F	7.4		7.6	0.291		0.300
G						
L	0.5		1.27	0.020		0.050
M			0.75			0.029
S	8° (max.)					



PO13I

TSSOP16 MECHANICAL DATA

DIM.	mm.			inch		
	MIN.	TYP	MAX.	MIN.	TYP.	MAX.
A			1.2			0.047
A1	0.05		0.15	0.002	0.004	0.006
A2	0.8	1	1.05	0.031	0.039	0.041
b	0.19		0.30	0.007		0.012
c	0.09		0.20	0.004		0.0089
D	4.9	5	5.1	0.193	0.197	0.201
E	6.2	6.4	6.6	0.244	0.252	0.260
E1	4.3	4.4	4.48	0.169	0.173	0.176
e		0.65 BSC			0.0256 BSC	
K	0°		8°	0°		8°
L	0.45	0.60	0.75	0.018	0.024	0.030



BitCloud™ SerialNet™

User Guide

8269A–MCU Wireless–11/09

Table of Contents

Section 1

Introduction 1-1

Section 2

References 2-1

2.1 Related Documents and References 2-1

2.2 Abbreviations and Acronyms 2-1

Section 3

Overview 3-1

3.1 Supported Platforms 3-1

3.2 Conventions 3-1

3.3 Architecture Overview 3-2

3.3.1 Protocol Principles 3-2

Section 4

Getting Started 4-1

4.1 Connection with Board 4-1

Section 5

Command Summary 5-1

5.1 AT Commands 5-1

5.1.1 Parameter Persistence 5-3

5.2 Result Codes 5-4

5.3 S-registers 5-5

5.4 Examples 5-6

5.4.1 Prepare nodes for networking 5-6

5.4.2 Checking network status and basic data transmission 5-7

5.4.3 Remote Extension 5-8

5.4.4 End Device Power Control 5-8

5.4.5 Control of LED and DIP switches 5-9

Section 6

Command Description 6-1

6.1 Protocol General Description 6-1

6.1.1 Character Formatting and Data Rates 6-1

6.1.2 Alphabet 6-1

6.1.3 Basic Command-Line Operations 6-1

6.1.4 Parameter Values 6-2
6.1.5 Command Types 6-3
6.1.6 Action Command Syntax 6-4
6.1.7 Parameter Set Command Syntax 6-4
6.1.8 Parameter Read Command Syntax 6-4
6.1.9 Parameter Test Command Syntax 6-5
6.1.10 S-registers 6-6
6.1.11 Device Responses 6-7
6.1.12 Information Text Formats 6-7
6.2 Networking Parameters 6-7
6.2.1 "+WPANID" - Set/Get extended PAN ID 6-8
6.2.2 "+WCHAN"- Get active channel 6-9
6.2.3 "+WCHMASK" - Set/Get channel mask 6-9
6.2.4 +WCHPAGE" - Set/Get channel page 6-10
6.2.5 +WAUTONET" - Enable/Disable automatic networking 6-11
6.2.6 "+WROLE" - Set/Get node role (Coordinator / Router / End device) 6-11
6.2.7 “+GSN” – Set/Get extended (MAC) address 6-12
6.2.8 "+WSRC" - Set/Get short (NWK) address 6-13
6.3 Network Management Functions 6-13
6.3.1 "+WJOIN" - Start/Join to the network 6-14
6.3.2 "+WLEAVE" - Leave the network 6-14
6.3.3 “+WNWK” – Get networking status 6-14
6.3.4 +WPARENT" - Get parent address 6-15
6.3.5 “+WCHILDREN” – Get children addresses 6-15
6.3.6 "+WNBSIZE" - Get number of neighbors 6-15
6.3.7 "+WNB" - Get neighbor information 6-16
6.3.8 "S30" - Set node addressing mode 6-17
6.3.9 "+WLQI" - Get LQI value 6-18
6.3.10 "+WRSSI" - Get RSSI value 6-18
6.4 Data Transmission 6-19
6.4.1 Parent polling mechanism 6-19
6.4.2 "D" - Send data to a specific node 6-20
6.4.3 "DU" - Send broadcast data 6-20
6.4.4 "DS" - Send S-register value to a specific node 6-21
6.4.5 "+WPING" - Ping the node 6-21
6.4.6 "+WSYNCPRD" - Poll rate for requesting indirect transactions from the parent 6-22
6.4.7 "+WTIMEOUT" - Data delivery time-out 6-22
6.4.8 "+WRETRY" - Repetition count 6-23
6.4.9 "+WWAIT" - Data transmission waiting time-out 6-23
6.5 Power Management 6-24

6.5.1 "+WPWR" - Configuration of sleep/active intervals 6-24
6.5.2 "+WSLEEP" - Force node to sleep 6-25
6.5.3 "+WTXPWR" - TX power level 6-25
6.6 Generic Control 6-26
6.6.1 "Z" - Warm reset 6-26
6.6.2 "&H" - Command Help 6-27
6.6.3 "%H" - Display parameters and S-register values 6-28
6.6.4 "I" - Display product identification information 6-29
6.6.5 "+GMI" - Get manufacturer identifier 6-29
6.6.6 "+GMM" - Request for the model identifier 6-30
6.6.7 "+GMR" - Request for the hardware/software revision identifier 6-30
6.6.8 “&F” – Set to factory-default configuration 6-30
6.6.9 "+WACALIBRATE" - Configure periodic internal clock calibration 6-31
6.6.10 "+WCALIBRATE" - Calibrate internal clock 6-31
6.7 Hot Interface Commands 6-32
6.7.1 "S3" - Termination character 6-32
6.7.2 "S4" - Response formatting character 6-33
6.7.3 "S5" - Command editing character 6-33
6.7.4 "E" - Command echo 6-34
6.7.5 "Q" - Result code suppression 6-35
6.7.6 "V" - Response format 6-35
6.7.7 "X" - Result code selection 6-36
6.7.8 "+IPR" - Serial port communication rate 6-37
6.7.9 "+IFC" - Serial port flow control 6-38
6.7.10 "&D" - DTR behavior 6-39
6.7.11 S0 - Request for the latest result code 6-39
6.8 Hardware Control 6-40
6.8.1 GPIO configuration 6-40
6.8.2 GPIO 6-41
6.8.3 A/D configuration 6-42
6.8.4 A/D 6-43
6.8.5 PWM configuration 6-44
6.8.6 PWM frequency control 6-45
6.8.7 PWM duty cycle control 6-45
6.9 Remote Management 6-46
6.9.1 "+WPASSWORD" - Set a password 6-46
6.9.2 "R"-Remote execution of AT command 6-47

Section 7

User Guide Revision History 7-1
7.1 Rev.8021A – 11/09 7-1

Section 1

Introduction

SerialNet is a manufacturer-specific profile developed on top of BitCloud C API. It offers control of embedded BitCloud stack through a serial interface using standardized AT-command set and requires no embedded API programming. Node's parameters can be easily accessed over-the-air without specifically dedicated protocol thus opening a way to network management and remote node control

The document presents the description of the SerialNet AT-command language

Section 2

References

2.1 Related Documents and References

1. RZUSBSTICK. Chapter 4, AVR2016: RZRAVEN Hardware User's Guide. www.atmel.com
2. ZigBit™ 2.4 GHz wireless modules. ATZB-24-A2/B0 datasheet. www.atmel.com/zigbit
3. ZigBit™ 2.4 GHz wireless modules. ATZB-A24-UFL/U0 datasheet. www.atmel.com/zigbit
4. ZigBit™ 700/800/900 MHz wireless modules. ATZB-900-B0 datasheet. www.atmel.com/zigbit
5. ZigBee™ Specification, Document 053474r17, October 2007.
6. Serial asynchronous automatic dialing and control. ITU-T Recommendation V.250, 05/99
7. International Reference Alphabet (IRA) (Formerly International Alphabet No. 5 or IA5). Information Technology – 7-Bit Coded Character Set for Information Interchange, CCIT Recommendation T.50, 09/92.
8. General Structure of Signals of International Alphabet No. 5. Code for Character Oriented Data Transmission over Public Telephone Networks. ITU-T Recommendation V.4
9. IEEE Std. 802.15.4-2006 IEEE Standard for Information technology – Part 15.4 Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)
10. BitCloud ™ IEEE 802.15.4/ZigBee Software. www.atmel.com/bitcloud
11. AVR2051: BitCloud Stack Documentation. Part of BitCloud SDK.
12. 8-bit AVR Microcontroller with 64K/128K/256K Bytes In-System Programmable Flash ATmega 640/V, ATmega 1280/V, ATmega 1281/V, ATmega 2560/V, ATmega 2561/V. www.atmel.com

2.2 Abbreviations and Acronyms

Table 2-1. Abbreviations and Acronyms

ARQ	Automatic Repeat-reQuest
ASCII	American Standard Code for Information Interchange
BS	Backspace character
CCITT	Consultative Committee on International Telephony and Telegraphy.
CR	Carriage Return
CRE	Coordinator / Router / End device (meaning any of those)
CTS	Clear To Send
DCE	Data Communication Equipment,
DTR	Data Terminal Ready
EEPROM	Electrically Erasable Programmable Read Only Memory

Table 2-1. Abbreviations and Acronyms

GPIO	General Purpose Input/Output
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
ITU	International Telecommunications Union
LED	Light Emitting Diode
LF	Line Feed character
LQI	Link Quality Indicator
LSB	Least Significant Bit
MAC	Medium Access Control (Sublayer)
MCU	MultiController Unit/Multi-Chip Unit
NWK	Network layer
OEM	Original Equipment Manufacturer
PAN	Personal Area Network
PHY	PHYsical Layer
PWM	Pulse Width Modulation
R	Read-only parameter
RSSI	Received Signal Strength Indicator
RTS	Request To Send
RW	Read-write parameter
RX	Receiver
TBD	To Be Defined
TX	Transmitter
UART	Universal Asynchronous Receiver Transmitter
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
ZDO	ZigBee Device Object

Section 3

Overview

SerilalNet is based on the AT-command protocol which is widely used in embedded networking systems due to its simplicity, textual parameter representation and inherent flexibility. This Chapter gives a brief introduction into the concept of SerialNet protocol, lists HW platforms SerialNet is available for and describes conventions used throughout the document.

3.1 Supported Platforms

The following hardware platforms are supported by SerialNet:

Table 3-1. Supported hardware platforms

Name in This Document	Platform (MCU + RF)	ZigBit Modules	Appropriate SDK
RZUSBSTICK	AT90USB1287 + AT86RF230 See [1.] on page 2-1	N/A	BitCloud for ATAVRRZRAVEN
ZigBit™	ATmega1281 + AT86RF230	ATZB-24-B0 (ZigBit B0); ATZB-24-A2 (ZigBit A2). See [2.] on page 2-1	BitCloud for ZDK
ZigBit™ Amp	ATmega1281 + AT86RF230	ATZB-A24-UFL (ZigBit Amp) [3.] on page 2-1	BitCloud for ZDK Amp
ZigBit™ 900	ATmega1281 + AT86RF212	ATZB-900-B0 (ZigBit 900) [4.] on page page 2-1	BitCloud for ZDK 900

Most of the SerialNet commands are HW-independent and can be executed on all supported platforms. However,a few commands either exhibit platform-specific behavior or are supported on particular HW platforms only. For such cases, command descriptions given in Chapter provide corresponding differences in the command functionality for various platforms. If no reference to platform is given in command description, then platform-independence is implied.

3.2 Conventions

The term *module* will be used throughout the document implying a supported platform (MCU + RF chip) controlled by a *host* equipment using AT-commands.

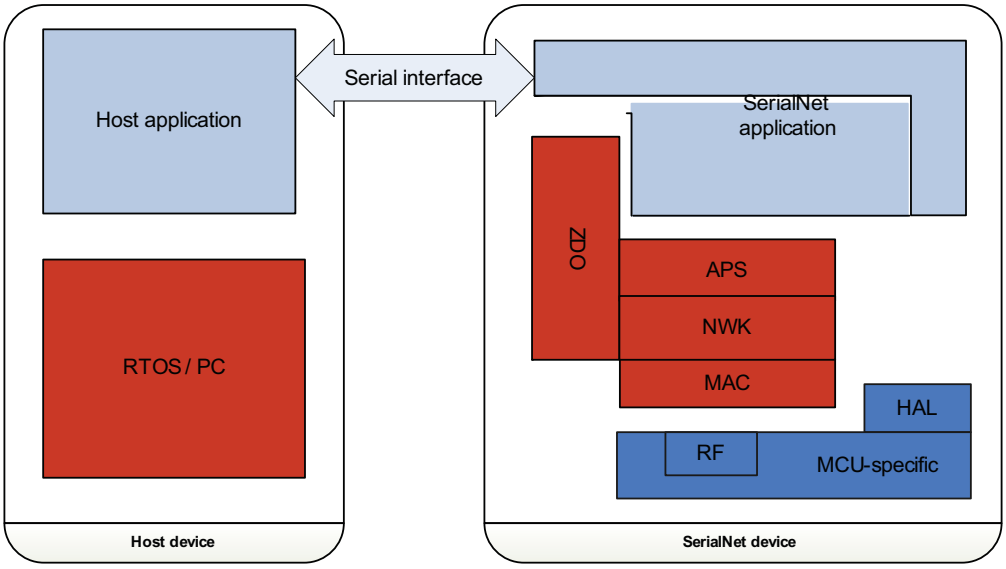
The term *node* will be used in reference to the device's role in the network (End device, Router or Coordinator).

To be distinguished from the rest, the definitions of commands directed to the module are denoted in Courier while the module responses are given in **Bold Courier** font. Angle brackets enclose mandatory parameters. Square brackets contain optional parameters.

3.3 Architecture Overview

SerialNet application is developed on top of Atmel's BitCloud ZigBee PRO-certified stack, see - Step 10.on page 2-1. It provides an easy-to-use control over ZigBee PRO networking functionality that is accessible for the host device through serial connection using an extensive set of AT-commands in ASCII format. SerialNet device executes received requests and responds to the host. Table 3-1 illustrates the basic architecture.

Figure 3-1. SerialNet usage scheme

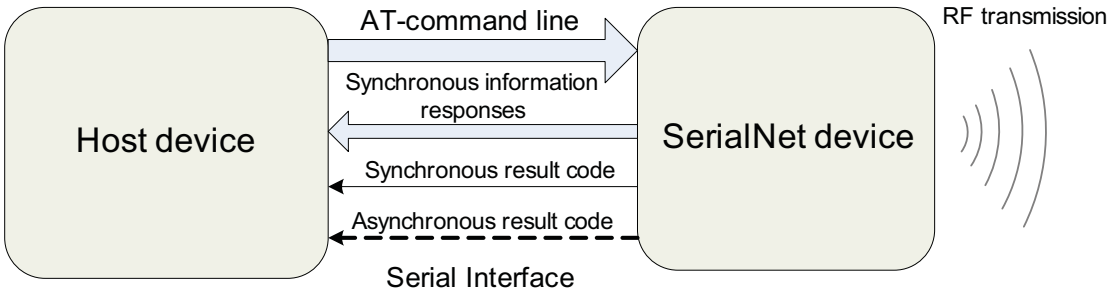


An important feature of SerialNet is the capability to request execution of particular function over the air via ATR command (see Table 5-8 on page 5-8). It allows transferring the AT-command to the remote node in the network, executing it there and redirecting the execution output to the originator. Thus, the remote node can be monitored, commissioned and the corresponding parameters can be set.

3.3.1 Protocol Principles

SerialNet supports an extensive set of AT-commands that provide full control over different functionality of the module. Read/write commands to S-registers can be used to access device and network parameters. In many cases AT-command functionality can be duplicated by certain S-register to reduce overhead of the serial protocol. The basic principle of SerialNet protocol is illustrated in Table 3-2.

Figure 3-2. SerialNet command executions



The host device shall transmit a command line prefixed by the "AT" string followed by the chained SerialNet commands to be executed consecutively. Upon successful execution of each command in the sequence corresponding information response is returned to the host device in an easily recognizable string format. The final result of the command line execution is indicated by the result code. In case of any command executed incorrectly, the command sequence is interrupted and the `ERROR` result code is returned. Result code is OK if all commands in the sequence were executed successfully.

Each command in a sequence may have different syntax, depending on whether it is used to execute an action, to read or to write parameter(s) or to test valid parameter range. An example illustrating different command and response types is provided in Table 3-2.

Table 3-2. At command string execution

	Command/Response	Comment
Command to device	ATE1V1+WTXPWR=-4+WLQI2+WRSSI2S22?	Turn echo on (E1), enable verbose response, set Tx power level to -4 dBm, request for LQI and RSSI for link with node 2, request for active channel
Information responses	**+WLQI:254**	LQI value is 254
	+WRSSI:-80	RSSI is -80 dBm
	B	Node is operating on channel 0x0B
Result code	**OK**	Execution is completed successfully

More complex examples are provided in the section “Examples” on page 5-6.

In addition to synchronous result codes indicating command execution status, SerialNet device upon specific event s can send to the host device asynchronous result codes. The full list of both verbose and numeric forms of the result codes can be found in “Parameter Persistence” on page 5-4.

“AT Commands” on page 5-1 summarizes the basic specifications of AT-commands grouped into functional categories while detailed definition for each command is given in Chapter 6.

“S-registers” on page 5-6 is a functional representation of S-registers with the corresponding AT-commands.

Section 4

Getting Started

4.1 Connection with Board

The supported platform (see"Supported Platforms" on page 3-1) shall be first programmed (via JTAG, USB or RS-232) with the SerialNet firmware version for the corresponding platform. After that it shall be connected to a host device (a PC, MCU, etc.) using USB or RS-232 interface. To start communication the host device shall configure its serial port with default SerialNet parameters:

Table 4-1. Default Serial Net Parameters

Baud rate	38400
Data bits:	8
Parity:	None
Stop bits:	1
Flow control:	None

Note that these parameters can be modified for SerialNet device and saved in persistent memory using corresponding commands described in .

If a PC is the host, then HyperTerminal software from the standard Windows package can be used to communicate with SerialNet device. To check the connection, AT should be entered on the terminal window followed by <Enter>. If the board responds with OK, then communication between host and SerialNet devices is established successfully.

The section "Examples" on page 5-6 includes examples showing how a SerialNet device can be configured for networking operations, data exchange and remote control.

Section 5

Command Summary

5.1 AT Commands

The AT-commands implemented in SerialNet fall into the following categories:

- Network configuration and management
- Data transmission
- Power management
- Generic control
- Host interface control
- Hardware control
- Remote management.

Table 5 1 provides a full list of SerialNet commands with information about supporting node roles, syntaxes, corresponding S-registers (if any), persistence and references to the detailed command description in Chapter 5.4.5.

Table 5-1. Command Summary

Function	Node type (C/R/E)	S-register	Action syntax	Parameter set syntax	Parameter read syntax	Parameter test syntax	Command	Persistence	Reference
Networking parameters									
Extended PAN ID	CRE	20, 21		x	x	x	+WPANID	x	6.2.1
Active channel	CRE	22			x		+WCHAN		6.2.2
Channel mask	CRE	23		x	x	x	+WCHMASK	x	6.2.3
Channel page	CRE	25		x	x	x	+WCHPAGE	x	6.2.4
Automatic networking	CRE	24		x	x	x	+WAUTONET	x	6.2.5
Node role	CRE	33		x	x	x	+WROLE	x	6.2.6
Device extended address	CRE			x	x		**+GSN or I4**		6.2.7
Node short address	CRE	55		x	x	x	**+WSRC**	**x**	6.2.8
Network management									
Start/Join to network	CRE		x				**+WJOIN**		6.3.1

Table 5-1. Command Summary

Leave the network	CRE		x				**+WLEAVE**		6.3.2
Request for networking status	CRE		x				+WNWK		6.3.3
Request for parent address	E				x		+WPARENT		6.3.4
Request for children addresses	CR				x		+WCHILDREN		6.3.5
Request for a number of neighbor nodes	CRE				x		+WNBSIZE		6.3.6
Request for neighbors' information	CRE				x		+WNB		6.3.7
Network addressing mode	CRE	30		x	x		S30		6.3.8
Request for LQI	CRE		x				+WLQI		6.3.9
Request for RSSI	CRE		x				+WRSSI		6.3.10
Power management									
End device sleep parameters	CRE	31, 32		x	x	x	+WPWR	x	6.5.1
Force to sleep	E		x				+WSLEEP		6.5.2
Tx power level	CRE	34		x	x	x	+WTXPWR	x	6.5.3
Data transmission									
Send data to specific node	CRE		x				D		6.4.2
Send broadcast data	CRE		x				DU		6.4.3
Send S-register value to specific node	CRE		x				DS		6.4.4
Ping the node	CRE		x				+WPING		6.4.5
Indirect poll rate	CRE	37		x	x	x	+WSYNCPRD		6.4.6
Data delivery time-out	CRE	51			x		+WTIMEOUT		6.4.7
Repetition count	CRE	52			x		+WRETRY		6.4.8
Data transmission waiting time-out	CRE	53		x	x	x	+WWAIT	x	6.4.9
Generic control									
Warm reset	CRE		x				Z		6.6.1
Help	CRE		x				&H		6.6.2
Display parameters and S-register values	CRE		x				%H		6.6.3
Display product identification information	CRE		x				I, I0		6.6.4
Request for Manufacturer Identification	CRE		x				`+GMI or I1`		6.6.5
Request for Model Identification	CRE		x				`+GMM or I2`		6.6.6
Request for hardware/software revision Identification	CRE		x				`+GMR or I3`		6.6.7
Set to factory-defined configuration	CRE		x				&F		6.6.8
Host interface commands									
Termination character	CRE	3		x	x		S3	x	6.7.1

Table 5-1. Command Summary

Response formatting character	CRE	4		x	x		S4	x	6.7.2
Command editing character	CRE	5		x	x		S5	x	6.7.3
Command echo	CRE		x				E	x	6.7.4
Result code suppression	CRE		x				Q	x	6.7.5
Response format	CRE		x				V	x	6.7.6
Result code selection	CRE		x				X	x	6.7.7
Serial port communication rate	CRE			x	x	x	+IPR	x	6.7.8
Serial port flow control	CRE			x	x	x	+IFC	x	6.7.9
DTR behavior	CRE	50	x				&D	x	6.7.10
Request for the latest result code	CRE	0			x		S0		6.7.11
Hardware control									
GPIO configuration	CRE	120 … 128		x	x		S120…S128	x	6.8.1
GPIO	CRE	130 … 138		x	x		S130…S138		6.8.2
A/D configuration	CRE	100		x	x		S100	x	6.8.3
A/D	CRE	101 … 104			x		S101…S104		6.8.4
PWM configuration	CRE	140, 141, 142		x	x		S140, S141, S142		6.8.5
PWM frequency control	CRE	143, 144, 145		x	x		S143, S144, S145		6.8.6
PWM duty cycle control	CRE	146, 147 148		x	x		S146, S147, S148		6.8.7
Remote management									
Set a password	CRE		x				+WPASSWORD	x	6.9.1
Remote execution of AT command	CRE		x				R		6.9.2

Note: ***1.*** The second column contains roles of nodes to which a given command is applicable. C stands for Coordinator, R for Router, and E for End device.

5.1.1 Parameter Persistence

In Table 5-1 many parameters associated with AT-commands are indicated as persistent. This means that their values are stored in persistent memory of MCU and in contrast to non-persistent parameters they will not be set to default configuration upon device reset.

However, value assigned to a persistent parameter by corresponding AT command is not written to the persistent memory right away. Instead it is applied to SerialNet operation but is kept in RAM. SerialNet periodically (with 5 minutes interval) verifies whether values of persistent parameters in EEPROM match their actual values in RAM. If differences are detected, then corresponding values in EEPROM are updated. For platforms with warm reset command support (see Table 6-41 on page 6-26) persistent

parameters in EEPROM are updated to actual values (if necessary) automatically upon ATZ command execution.

Upon device reset SerialNet assigns persistent parameters to their values stored in EEPROM. If a parameter value has not been transferred from RAM to EEPROM then the old EEPROM value will be used.

5.2 Result Codes

Result codes appear either synchronously in response to a command or, asynchronously, due to the specific events in the network or on a SerialNet device. See detailed description of result codes in "Device Responses" on page 6-7. Table 5-2 provides both verbose and numeric forms for available result codes.

Table 5-2. Result codes

Verbose Code	Numeric Code	Parameters	Description
OK	**0**	None	Command is executed successfully
ERROR	**4**	None	Error occurred during command execution
DATA	**8**	**<addr>,<bcast>,<length>:<data>**	Indicates data reception from a remote node. **addr** is a short (network) address of a source node data is originating from **bcast** is set to 1 if data is sent by broadcast transmission, otherwise it is set to 0 **length** is a length of the **<data>** field **data** is byte sequence of received data Note: +WPING command (see Table 6-33 on page 6-21) results in the following code on the destination node: **DATA <addr>,0,0:**
EVENT	7	**:<text>**	text is a text specifying an event.
		:JOINED	Indicates that the node has joined to the network Note: Event is returned in auto network mode only and not after +WJOIN command.
		:LOST	Indicates that the node has lost connection to the network (i.e. to its current parent) Note: Event can occur on end device nodes only and is not returned after +WLEAVE.

Table 5-2. Result codes

		:CHILD_JOINED <addr>	Indicates to the node that device with extended address **<addr>** has just joined to it as a child
		:CHILD_LOST <addr>	Indicates to the node that its child end device with extended address **<addr>** has disconnected from the node. Note: Event occurs when child end device switches to a new parent, when it leaves the network using +WLEAVE command or when it is not accessible (powered off, no link, etc.) for *3*(sleep_interval + sync_period)* as configured on parent device by +WPWR and +WSYNCPRD commands.
		:CALIBR	Indicates that the device has successfully calibrated its internal clock after encountering errors on serial interface.

5.3 S-registers

An extensive set of S-registers available in SerialNet provides easy read/write access to device and networking parameters. In many cases AT-command functionality can be duplicated by certain S-register to reduce overhead of the serial ASCII protocol.

Table 5-3. S-Registers

Parameter	Acceptable Operations (R/RW)	S-register	Command Reference
The latest result code	R	S0	6.7.11
Termination character	RW	S3	6.7.1
Response formatting character	RW	S4	6.7.2
Command editing character	RW	S5	6.7.3
PAN ID	RW	S21, S20	6.2.1
Active channel	R	S22	6.2.2
Channel mask	RW	S23	6.2.3
Automatic networking	RW	S24	6.2.5
Channel page	RW	S25	6.2.4
Network addressing mode	RW	S30	6.3.8
Power management	RW	S31, S32	6.5.1
Node role	RW	S33	6.2.6
Tx power level	RW	S34	6.5.3
Indirect poll rate	RW	S37	6.4.6
DTR behavior	RW	S50	6.7.10
Data delivery time-out	R	S51	6.4.7
Repetition count	R	S52	6.4.8

Table 5-3. S-Registers

Data transmission waiting time-out	RW	S53	6.4.9
Own network address	RW	S55	6.2.8
A/D configuration	RW	S100	6.8.3
A/D	R	S101…S104	6.8.4
GPIO configuration	RW	S120…S128	6.8.1
GPIO	RW	S130…S138	6.8.2
PWM configuration	RW	S140, S141, S142	6.8.5
PWM frequency control	RW	S143, S144, S145	6.8.6
PWM duty cycle control	RW	S146, S147, S148	6.8.7

5.4 Examples

The examples given below show usage of AT-commands to control the SerialNet devices and are valid for all supported platforms listed in "Supported Platforms" on page 3-1.

5.4.1 Prepare nodes for networking

The following examples require at least 2 nodes. The first step is configuring network parameters. One of the nodes should function as a coordinator and others could be routers or end devices. It is important that all nodes have different extended (MAC) and short (NWK) addresses. Coordinator node shall have short address 0, and all other nodes shall have non-zero addresses.

Note: Selection of particular addresses is application dependent. It should be done only the first time during the manufacturing process of initial installation.

Table 5-4. Network coordinator

Command/Response	Comment
`ATX`	set a node to transmit **EVENT** and **DATA** to a host
OK	
`AT+GSN=1`	set extended address for the node
OK	
`AT+WPANID=1620`	set node's extended PAN ID
OK	
`AT+WCHMASK=100000`	set node's channel mask (this one enables channel 0x14 only)
OK	
`AT+WROLE=0 +WSRC=0`	set coordinator role and short address to 0x0000
OK	
`AT+WJOIN`	perform network start
`OK`	result code for successful network start

If the node indicates ERROR, that means the embedded software does not support coordinator function and cannot be configured in such a way. In this case, try checking the coordinator support on other nodes using AT+WROLE? command, as described in Table 6-17 on page 6-11.

Then set configure another device to be a router node:

Table 5-5. Network router

Command/Response	Comment
ATX	set a node to transmit **EVENT** and **DATA** to a host
OK	
AT+GSN=2	set extended address for the node
OK	
AT+WPANID=1620	set node's extended PAN ID
OK	
AT+WCHMASK=100000	Set node's channel mask (this one enables channel 0x14 only)
OK	
AT+WROLE=1 +WSRC=55	set router role, short address equal to 0x0055
OK	
AT+WJOIN	perform network join
OK	indication for router having joined the network

5.4.2 Checking network status and basic data transmission

Now we can easily verify networking status on both devices by AT+WNWK command and perform data exchange between them. For example on coordinator:

Table 5-6. Verify networking status on coordinator

Command/Response	Comment
AT+WNWK	request networking status
OK	means that the node is in the network
AT+WWAIT=3000 **OK** ATD55 HELLO **OK**	set 3 sec time-out to wait for input and send HELLO word to the node with short address 55

Simultaneously, HELLO word will appear on the terminal connected to the router in form of DATA event:

Table 5-7. Verify networking status on router terminal

Command/Response	Comment
DATA 0000,0,5:HELLO	data (5 bytes) came from device with address 0 by unicast request

5.4.3 Remote Extension

ATR command provides mechanism for AT-command execution on a remote node with command response redirection to the originator. Thus it allows remote monitoring and configuration over the air.

The example below demonstrates how to execute AT-commands on the router device remotely using ATR command on the coordinator:

Table 5-8. Remote execution of AT-commands on the router

Command/Response	Comment
ATR55,0,+WROLE?+GSN? **+WROLE:1** **+GSN:0000000000000055** **OK**	get node role and extended address from the router
ATR55,0,+GMI? **+GMI:ATMEL** **OK**	get model number from the router
ATR55,0,+WAUTONET=1S30=1 **OK**	set autonet mode and command addressing mode

5.4.4 End Device Power Control

This example demonstrates how to configure an end device node with certain duty cycle, perform network join and deliver data to an end device:

Table 5-9. Configure end device node with duty cycle

Command/Response	Comment
ATX **OK** AT+GSN=3 **OK** AT+WROLE=2 +WSRC=56 **OK** AT+WPANID=1620+WCHMASK=100000 **OK** AT+IFC=2,2 **OK**	set a node to transmit **EVENT** and **DATA** to a host set extended (MAC) address for the node set the board as end device with short address 0x0056 set extended PAN ID and channel mask (channel 0x14) for this node configure RTS and CTS line modes for end device flow control. Reconfigure flow control on the host accordingly. (E.g. select Hardware mode for Flow Control in Hyper Terminal)
AT+WPWR=100,100 **OK** AT+WPWR? **+WPWR:100,100** **OK**	set duty cycle 10 sec sleep / 1 sec active verify that the duty cycle is accepted successfully
AT+WJOIN **OK**	perform network join result code indicating successful network join for the end device

Now, the data intended for the end device can be sent from the coordinator:

Table 5-10. Test data from the coordinator

Command/Response	Comment

`ATD56,0,4`
`test`
`OK`
	send test data from coordinator for the end device staying in a sleep mode

In active state end device periodically polls its parent for buffered data with interval configured by `+WSYNCPRD` parameter. In the given example it retrieves the test frame:

Table 5-11. Polling of buffered data from parent

Command/Response	Comment
`DATA 0000,0,4:test`	the `test` word is received by end device after wake up

5.4.5 Control of LED and DIP switches

The example below is valid only for MeshBean2 development boards. Mapping of I/O pins of the ZigBit module and their functions on the MeshBean2 boards is summarized in the table below

Table 5-12. GPIO Pins Summary

Component	I/O pin	Description
LED1	GPIO0	output, 1 means LED on
LED2	GPIO1	output, 1 means LED on
LED3	GPIO2	output, 1 means LED on
SW4:1	GPIO3	input (no pull-up on the board), ON – logical zero
SW4:2	GPIO4	input (no pull-up on the board), ON – logical zero
SW4:3	GPIO5	input (no pull-up on the board), ON – logical zero
	GPIO6	reserved for MeshBean2 sensor interfaces
	GPIO7	reserved for MeshBean2 sensor interfaces
	GPIO8	reserved for MeshBean2 sensor interfaces

Initially, set DIP-switches physically as SW4:1 to OFF, SW4:2 and SW4:3 to ON, and, next, configure I/O pins via command:

Table 5-13. Configure I/O pins

Command/Response	Comment
`ATS120=3 S121=3 S122=3`	configure GPIO0, GPIO1, GPIO2 for output
`OK`	
`ATS123=1 S124=1 S125=1`	configure GPIO3, GPIO4, GPIO5 for input and turn on internal pull-up
`OK`	

Afterwards, it is possible to control LEDs and to obtain status of DIP-switches using corresponding S-registers:

Table 5-14. Control LEDs and check DIP-switches

Command/Response	Comment
`ATS130=1 S131=0 S132=1`	turn on LED1 and LED3
OK	
`ATS133? S134? S135?`	
1	SW4:1 is in the OFF state
0	SW4:2 is in the ON state
0	SW4:3 is in the ON state
OK	

Section 6

Command Description

6.1 Protocol General Description

6.1.1 Character Formatting and Data Rates

Data transmitted between the host and the module over serial interface conforms to the requirements for start-stop data transmission specified in the ITU-T Recommendation V.4 Step 8. in page 2-1. Parity is even, odd or not used. Each character has at least one complete stop bit. The module accepts commands using any combination of parity and stop bits supported. These include, at least, the following combinations, each of which consists of up to ten bits (including the start bit):

- 7 data bits, even parity, 1 stop bit
- 7 data bits, odd parity, 1 stop bit
- 8 data bits, no parity, 1 stop bit.

Both the host and the module are able to accept commands at 1200 bit/s at least. Particular character formatting and the data rate can be changed using appropriate AT-commands - see Table 6-59 on page 6-37), Table 6-60 on page 6-38 and Table 6.7.6 on page 6-35. The host has the means to select explicitly data rate and character formatting according to the specifications above.

6.1.2 Alphabet

For any information exchange between the module and the host the T.50 International Alphabet 5 (IA5) is used - see Step 7.in "Related Documents and References" on page 2-1. Only the seven low-order bits of each character are significant, any of eighth or higher-order bit(s), if present, are ignored for the purpose of identifying commands and parameters. Lower-case characters (hex codes `0x61` through `0x7A`) are considered identical to their upper-case equivalents (hex codes `0x41` through `0x5A`) when received by the module from the host. Result codes from the module, which are particularly defined, are specified in upper case.

6.1.3 Basic Command-Line Operations

Command line editing, echoing and repeating are done in accordance with the Clauses 5.2.2, 5.2.3 and 5.2.4 of the Recommendation V.250. The description below follows the statements introduced in Step 6. in "Related Documents and References" on page 2-1.

The module may echo the characters received from the host back to the host, depending on the setting of the E command (see Table 6.7.4 on page 6-34). If so enabled, the characters received from the host are echoed at the same rate, parity, and format as those received.

The module checks on the characters coming from the host first, to see if they match the termination character S3 (see Table 6-51 on page 6-32). Next, it checks the editing character (S5, see Table 6-53 on page 6-33), before considering any other character. That insures the characters will be properly recognized even though they were set to values which the module uses for other purposes. If S3 and S5 are

set to the same value, the character checked will be treated as a character matching S3 (as S3 is checked before S5).

The character defined by S5 parameter (by default, it is backspace character - BS [hex code 0x08], see Table 6-53 on page 6-33) is intended to be interpreted as a request from the host to the module to delete the previous character. Any control characters (hex codes 0x00 through 0x1F, inclusive) that remain in command line after receiving the termination character will be ignored by the module.

Once the module finds the termination character, it starts processing the command line. Command line starts with AT (characters 0x41, 0x54) and should contain a sequence of commands in the following syntax formats:

Table 6-1. Command Syntax Formats

Command	Syntax
Action command	<command>[<value>]
Parameter set command	<command>=<value>
Parameter read command	<command>?
Testing a range of valid values	<command>=?

Where <command> is one of the following:

- a single character
- '&' character (0x26) followed by a single character
- '%' character (0x25) followed by a single character
- '+' character followed by a string of characters.

The characters allowed to be used in <command> should be taken from the T.50 International Alphabet 5. The first three of the command cases above are referred to as basic commands; they may be of the action command syntax only. Commands beginning with the plus sign are known as the extended syntax commands and can fit all the syntax rules depending on their type. Typically, a command that supports the parameter set syntax also supports the testing syntax.

A command (with associated parameters, if any) may be followed by additional commands in the same command line without using any delimiting character. Some commands may cause the remainder of the command line being ignored (the D command, see Table 6-30 on page 6-20, for instance).

If command line is started with the 'A/' or 'a/' prefix (hex codes 0x41, 0x2F or 0x61,0x2F), the module repeats immediately the execution of the preceding command line. No editing is possible, and no termination character is required. With this mechanism, a command line may be repeated as much as desired.

6.1.4 Parameter Values

Parameters may take either a single value, or multiple (compound) values. A compound value consists of any combination of numeric values (as defined in the description of the action or parameter command). The comma character (hex code 0x2C) is included as a separator, before the second and all subsequent values in the compound value. If a value is not specified as missed (i.e. defaults assumed), the required comma separator should be specified; however, trailing comma characters may be omitted if all the associated values are also omitted.

Note: When any of optional parameters is misused in a command, the command would be performed as if the parameter was be omitted. That parameter would be further treated as if the

other subsequent command were input, probably causing an ERROR message. To avoid confusions follow the command syntax.

Actions may have more than one of associated sub-parameters, and parameters may have more than one value. These are known as "compound values", and their treatment is the same in both the action command syntax and the parameter command syntax.

Each value may be either decimal or hexadecimal number. The choice depends on a particular command and hexadecimal numbers if they are not preceded with '0x'. Hexadecimal numbers can represent 16-bit, 32-bit, 64-bit and 128-bit values.

Decimal numeric constants consist of a sequence of one or more of the characters '0' (hex code 0x30) through '9' (hex code 0x39), inclusive, and can be preceded by minus "-". The most significant digit is specified first. The leading '0' characters will be ignored.

Hexadecimal numbers consist of characters "0" through "9" and "A" through "F", inclusive. Minus sign is not allowed. The leading '0' characters will be ignored. To prevent misinterpretation of hexadecimal numbers in cases when the command containing them is not the last in the AT string, it is strongly recommended to add the leading zeroes. So, if a parameter is 32-bit long, it would be 8 characters long, if it is a 64-bit number, it would contain 16 characters and so on.

As a special case, string constant appears in R command (see Table 6-71 on page 6-47) only. Then, it is just a sequence of displayable IA5 characters, each in the range of 0x20 to 0x7F, inclusive.

6.1.5 Command Types

A command type may be one of the following:

- An action command
- A parameter command
- An S-registers command.

Parameters may be defined as "Read-only" (R) or "Read/Write" (RW). "Read-only" parameters are used to provide the host with the status or identifying information, but are not set by the host. Attempting to set such a parameter will result in an error. In some cases (depending on the particular parameter), the module may ignore any attempt to set the value for such parameter rather than respond with the ERROR result code. "Read-only" parameters may be read and tested.

"Read/Write" parameters may be set by the host in order to store a value or values for later use. "Read/Write" parameters may be set, read, and tested.

If <command> is not recognized, the module generates the ERROR result code and stops processing of the command line. The ERROR result code is also generated if: a sub-parameter is specified for an action that does not imply using sub-parameters; too many sub-parameters are specified; a mandatory sub-parameter is not specified; a value is specified of the wrong type; or if a value is specified that is not within the supported range.

Some commands allow omitting a value. If a command does omit one, then it should be immediately followed by another command (or the termination character) in the command line. The '0' value is assumed unless otherwise specified in the <command> description. If the <command> does not expect a value but the value is present, the ERROR code is generated.

6.1.6 Action Command Syntax

The format of the action commands, except for the D, DU and S commands, is as follows:

Table 6-2. Action command syntax

Command	AT Syntax
Action command with no parameters used	<command>
Action command with one or more sub-parameters used	<command>[<value>]

The value may be either a single value parameter or a compound value parameter as described in 6.1.4. Some commands may have no parameters at all. Expected value is noted in the description of a particular command.

Table 6-3. Example of action command

Command/Response	Comment
AT+WLEAVE	Leave the network
OK	Result code
ATX2	2 - Disables events and data indications
OK	Result code

6.1.7 Parameter Set Command Syntax

The following syntax is used for a parameter set command:

Table 6-4. Parameter set command syntax

Command	AT Syntax
Parameter set command	<command>=[<value>]

If the named parameter is implemented in the module, all the mandatory values are specified, and all values are valid according to the definition of the parameter, the specified values should be stored. If <command> is not recognized, one or more of mandatory values are omitted, or one or more values are of wrong type or beyond the valid range, the module generates the ERROR result code and terminates processing of the command line. ERROR is also generated if too many values are specified. In case of error, the previous values of the parameter are unaffected:

Table 6-5. Example of parameter set command

Command/Response	Comment
AT+WWAIT=4000	Set parameter +WWAIT
OK	Result code

6.1.8 Parameter Read Command Syntax

The host may determine current value or values stored in a parameter by using the following syntax:

The following syntaxes are used

Table 6-6. Parameter read command syntax

Command	AT Syntax
Parameter read command	<command>?

If the named parameter is implemented, its current values are sent to the host in an information text response. The format of this response is described in definition of the parameter. Generally, the response string is beginning with <command> followed by ':' character and the values represented in the same form, in which they would be generated by the host in a parameter set command. If multiple values are supported, they will generally be separated by commas, as in a parameter set command. For example:

Table 6-7. Example of parameter read command syntax

Command/Response	Comment
AT+WRETRY?	Request for parameter +WRETRY
+WRETRY:3	Returned value
OK	Result code

6.1.9 Parameter Test Command Syntax

Table 6-8. Parameter test command syntax

Command	AT Syntax
Parameter test command	<command>=?

If the module does not recognize the indicated <command>, it returns the ERROR result code and terminates processing of the command line. If the module does recognize the parameter name, it returns an information text response to the host, followed by the OK result code. The information text response will indicate the values supported by the module for each of sub-parameters, and, possibly, additional information. The format of this information text response is defined for each parameter. See"Information Text Formats" on page 6-7 for the general formats for specification of sets and ranges of numeric values. Generally, an information text response is started with a <command> followed by ':'.

When an action/parameter accepts a single numeric sub-parameter, or the parameter accepts only one numeric value, the set of supported values may be presented in an information text as an ordered list of values. The list should be preceded by left parenthesis '(', (hex code 0x28), and closed by right parenthesis ')', (hex code 0x29). If that very single value is supported, it should appear in parentheses. If more than one value is supported, then the values may be listed individually, separated by comma characters (hex code 0x2C). When a continuous range of values is supported, the values appear in form of the first value in the range, and the last value in the range, both separated by a hyphen character (hex code 0x2D). The specification of single values and value ranges may be alternated within a single information text. Nevertheless, the supported values should be indicated in an ascending order. For example, the following are some examples of value range indications:

Table 6-9. Value range indications

Value	Comment
(0)	Only the 0 value is supported.
(1,2,3)	The values 1, 2, and 3 are supported.
(1-3)	The values 1 through 3 are supported.
(0,4,5,6,9,11,12)	The several listed values are supported.
(0,4-6,9,11-12)	Alternative expression of the previous list.

The value may be either a single value parameter or a compound value parameter as described in 6.1.4. Some commands may have no parameters at all. Expected value is noted in the description of a particular command.

Table 6-10. Example of parameter test command syntax

Command/Response	Comment
`AT+WSRC=?`	Request for valid range of the short address
`+WSRC:(0000-FFF7)`	Returned value
`OK`	Result code

When an action/parameter accepts more than one sub-parameter, or the parameter accepts more than one value, the set of supported values may be presented as a list of the parenthetically-enclosed value range strings, separated by commas. For example, the information text in response to testing an action that accepts three sub-parameters, and supports various ranges for each of them, could appear as follows:

```
(0),(1-3),(0,4-6,9,11-12)
```

This indicates that the first sub-parameter accepts only the 0 value, the second accepts any value from `1` through `3`, inclusively, and the third sub-parameter accepts any of the values `0, 4, 5, 6, 9, 11` or `12`.

6.1.10 S-registers

S-registers represent a group of numerical parameters that can be addressed in a special syntax. Each S-register has its own address and value. Some S-registers are standardized by the V.250 recommendations and are used in the module. Some of the S-registers are non-standard defined specifically by the SerialNet software.

AT-commands that begin with the 'S' character are allowed for S-register access. These differ from other AT-commands in some respects. The number following the 'S' character indicates the referenced "register number". If the number is not recognized as a valid register number (register is omitted), the `ERROR` result code is generated.

Immediately following that number, either a `'?'` or `'='` character (hex codes `0x3F` or `0x3D`, respectively) should appear. `'?'` is used to read the current value of the indicated S-parameter. `'='` is used to set the S-parameter to a new value.

Table 6-11. S-Registers

Command	AT Syntax
Reading the S-register	`S<parameter_number>?`
Setting the S-register	`S<parameter_number>=[<value>]`

If the '=' character is used, the new value to be stored in the S-parameter is specified in decimal form following the '=' character. If no value is given (i.e. the end of the command line occurs or the next command follows immediately), the corresponding S-parameter will be set to 0. The ranges of acceptable values are given in description of each S-register.

"S-registers" on page 6-6 gives functional representation of S-registers associated to the commands.

6.1.11 Device Responses

There are two types of responses that may be generated by the module:

- information responses
- result codes.

Basically, any information response consists of three parts: header, text, and trailer. The characters generated in header are determined by user's setting (see V command, Table 6-56 on page 6-35). Trailer consists of two characters, namely the ordinal value of parameter S3 followed by the ordinal value of parameter S4. Information text may contain multiple lines, and the text may include any formatting characters to improve readability.

A result code consists of three parts: header, the result text, and trailer. The characters to be generated in header and trailer are determined by user's setting (see the V command, Table 6-56 on page 6-35). The result text may be generated as a number or a string, depending on the user-selected setting (see the V command, Table 6-56 on page 6-35).

There are two general types of result codes: final and unsolicited.

Final result codes (**OK/ERROR**) indicate completion of the module action and readiness to accept new commands from the host. Unsolicited result codes (such as **DATA**) may not be directly associated with the issuance of a command from the host. They indicate the occurrence of another **EVENT** causing them.

Command X (see Table 6-58 on page 6-36) controls the generation of result codes, while command Q (see Table 6-55 on page 6-35) results in their total suppression.

"Parameter Persistence" on page 5-3 summarizes representations the result codes are in both verbose and numeric forms with the corresponding parameter(s), if any, and their brief description. Each command description itself refers to the specific result codes that may be generated in relation to the command and the circumstances, under which they may be issued.

6.1.12 Information Text Formats

In general, the particular format of information text returned by extended syntax commands will be specified in the command definition.

Note that the module may insert intermediate <CR> characters in very long information text responses, in order to avoid overflow in the host receive buffers. If intermediate <CR> characters are included, the module does not include the character sequences "**0** <CR>" (0x30, 0x0D) or "**OK**<CR>" (0x4F, 0x4B, 0x0D), so that the host can avoid false detection of the end of these information text responses.

6.2 Networking Parameters

This section describes SerialNet commands associated with networking parameters. Most of the parameters shall be set on each device according to desired network characteristics prior to executing network start/join procedure. Not that if default setting or persistent value from the EEPROM (see Section 5.1.1 on page 5-3) already has desired value for a network parameter, there is no need to assign it explicitly again prior to network start/join.

There is also a number of hard-coded parameters that cannot be changed by AT-commands but which have direct impact on possible network topology and performance:

```
CS_NEIB_TABLE_SIZE = 10
CS_MAX_CHILDREN_AMOUNT = 8
```

```
CS_MAX_CHILDREN_ROUTER_AMOUNT = 3
CS_MAX_NETWORK_DEPTH = 5
CS_ROUTE_TABLE_SIZE = 10
CS_ROUTE_DISCOVERY_TABLE_SIZE = 7
CS_ADDRESS_MAP_TABLE_SIZE = 10
CS_BTT_SIZE = 16
```

Their values shall be taken into account during network establishment and operation. Details about each parameter can be found in BitCloud Stack Documentation, see Step 11. in "Related Documents and References" on page 2-1.

6.2.1 "+WPANID" - Set/Get extended PAN ID

Table 6-12. "+WPANID" - Set/Get extended PAN ID

Syntax/Descriptor	Explanation
`+WPANID=<value>`	The command sets extended PAN ID for the device. `value` is extended PAN ID in form of a hexadecimal 64-bit number that uniquely identifies target network. If PAN ID is set to `0`, coordinator will form a network with extended PAN ID equal to its extended (MAC) address. Router and end device nodes in such case will join the first available network irrespectively to its extended PAN ID. Notes: 1. Setting the extended PAN ID is possible only when the device is not in the network. 2. Several networks with different PANIDs can be operated in parallel on the same frequency channel.
`+WPANID?`	The command returns extended PAN ID that is specified on the device for network operation.
`+WPANID=?`	The command requests valid range for extended PAN ID value.
`S-register`	`S21` (RW). This register is just keeping a copy of the parameter accessible through `+WPANID` command. `S20` (R). This register contains actual extended PAN ID that is used for networking. If `S21` register is set to `0`, and device is in the network, this register will keep extended PAN ID of the selected network. If device has not been connected, this register contains `0`.
`Result codes`	The set command is executed if device is not in the network and extended PAN ID is in the valid range. In such case device returns **`OK`** upon completion. Otherwise extended PAN ID is ignored and device responds with **`ERROR`**.

Example	`AT+WPANID=10`
	`OK`
	`AT+WPANID?`
	`+WPANID:0000000000000010`
	`OK`
	`AT+WPANID=?`
	`+WPANID:(0000000000000000-FFFFFFFFFFFFFFFE)`
	`OK`

Default value	`0000000000000000`
Persistence	`value` is stored in EEPROM
Node types	Coordinator / Router / End device

6.2.2 "+WCHAN"- Get active channel

Table 6-13. "+WCHAN"- Get active channel

Syntax/Descriptor	Explanation
+WCHAN?	The command requests the channel number (in hexadecimal form) the device is currently operating on. If the node is not in the network, **FF** is returned.
S-register	S22 (R)
Result codes	**OK**
Example	AT+WCHAN? **+WCHAN:0B** **OK**
Node types	Coordinator / Router / End device

6.2.3 "+WCHMASK" - Set/Get channel mask

Table 6-14. "+WCHMASK" - Set/Get channel mask

Syntax	Explanation
+WCHMASK=<value>-	The command sets channel mask to be used for network operation. value is a 32-bit field which specifies the channel numbers supported by the node. The 5 most significant bits of channel mask (b31,...,b27) shall be set to 0. The rest 27 bits (b26, b25,...b0) indicate availability status for each of the 27 valid channels (1 = supported, 0 = unsupported). Channels are distributed across frequency bands as follows: - 780 MHz: channel numbers 0 – 3 - 868 MHz: channel number 0 - 915 MHz: channel numbers 1 – 10 - 2.4 GHz: channel numbers 11 – 26 For sub-GHz bands corresponding channel page shall be configured by +WCHPAGE command (see Table 6-15). Detailed description of channel mask parameter can be found in the clause 6.1.2 of the 802.15.4-2006 standard. Notes: 1. Only channels from frequency bands supported by the platform's RF chip can be selected in the channel mask. 2. The command is not accessible when the node is joined to a network.
+WCHMASK?	The command returns actual channel mask. Returned channel mask can be different from the channel mask set by +WCHMASK=<value> command and depends on the hardware capabilities. The cleared bits mark unsupported channels.
+WCHMASK=?	The command returns channel capability mask in form of two 32-bit unsigned hexadecimal numbers. It returns 00000800-07FFF800 for 2.4 GHz chipset and 00000001-000007FF for Sub-GHz. Note: Strictly speaking, these two numbers do not represent "range" in its direct sense, but rather are the maximum and minimum values achievable by the composition of corresponding bits.
S-register	S23 (RW).
Result codes	The set command is executed if the node is not in the network and channel mask is set according to hardware capabilities really available. In such case device returns **OK**. Otherwise, channel mask is ignored and device responds with **ERROR**.

Table 6-14. "+WCHMASK" - Set/Get channel mask

Example	AT+WCHMASK=40000 **OK** AT+WCHMASK? **+WCHMASK:00040000** **OK** AT+WCHMASK=? **+WCHMASK(00000800-07FFF800)** **OK**
Default value	00000800 for 2.4 GHz chipset or 00000001 for Sub-GHz one.
Persistence	value is stored in the EEPROM.
Node types	Coordinator / Router / End device

6.2.4 +WCHPAGE" - Set/Get channel page

The command is available only for platforms with AT86RF212 radio part.

Table 6-15. "+WCHPAGE" - Set/Get channel page

Syntax	Explanation
+WCHPAGE=<value>	The command sets channel page that will be used for networking. Values 0 and 2 correspond respectively to BPSK and O-QPSK modulations on 868/915 MHz channels. Value 5 means that 780 MHz frequency band with O-QPSK modulation shall be used. Detailed description of channel page parameter can be found in the clause 6.1.2 of the 802.15.4-2006 standard Note: The command is not accessible when the node is joined to a network.
+WCHPAGE?	The command returns actual channel page.
+WCHPAGE=?	The command returns possible channel pages: **0,2,5**.
S-register	S25 (RW).
Result codes	**OK** if the device contains RF 212 radio chip and is not in the network; otherwise **ERROR** is returned.
Example	AT+WCHPAGE=0 **OK** AT+WCHPAGE? **+WCHPAGE:0** **OK** AT+WCHPAGE=? **+WCHPAGE:(0,2,5)** **OK**
Default value	0
Persistence	value is stored in the EEPROM.
Node types	Coordinator / Router / End device

6.2.5 +WAUTONET" - Enable/Disable automatic networking

Table 6-16. "+WAUTONET" - Enable/Disable automatic networking

Syntax	Explanation
+WAUTONET=<value>	The command controls the node activity behavior at power-up, reset or when a connection loss is detected. value has a Boolean type. 1 implies automatic joining to the network, 0 means that automatic joining is disabled and +WJOIN command shall be used for network start procedure.
+WAUTONET?	The command requests current automatic networking configuration.
+WAUTONET=?	The command requests the range of supported values.
S-register	S24 (RW).
Result codes	OK
Example	AT+WAUTONET=1 **OK** AT+WAUTONET? **+WAUTONET:1** **OK** AT+WAUTONET=? **+WAUTONET:(0,1)** **OK**
Default value	0 - automatic networking is disabled.
Persistence	value is stored in the EEPROM.
Node types	Coordinator / Router / End device

6.2.6 "+WROLE" - Set/Get node role (Coordinator / Router / End device)

Table 6-17. "+WROLE" - Set/Get node role (Coordinator / Router / End device

Syntax	Explanation
+WROLE=<value>	The command sets the node role to value as follows: 0 – Coordinator 1 – Router 2 – End device. Note: The command is not accessible when the node is joined to a network.
+WROLE?	The command requests the actual node role.
+WROLE=?	The command requests the node roles available for the device. Actual capabilities depend on the particular firmware version loaded on the device.
S-register	S33 (RW).
Result codes	**OK** is returned if value is in the valid range, otherwise **ERROR.**

Table 6-17. "+WROLE" - Set/Get node role (Coordinator / Router / End device

Example	AT+WLEAVE **OK** AT+WROLE=? **+WROLE:(0,1,2)** **OK** AT+WROLE=2 **OK** AT+WROLE? **+WROLE:2** **OK**	Leave the network Switch to the End device role
Default value	Depends on the firmware version. Typically 1 – Router.	
Persistence	value is stored in the EEPROM.	
Node types	Coordinator / Router / End device	

6.2.7 "+GSN" – Set/Get extended (MAC) address

Table 6-18. "+GSN" – Set/Get extended (MAC) address

Syntax	Explanation	
+GSN=<value>	The command assigns device extended (MAC) address. value is a 64-bit hexadecimal number that uniquely identifies the device. Note: The command is not accessible when the node is joined to a network.	
+GSN? I4	The command returns device extended (MAC) address in form of a 64-bit hexadecimal number.	
Result codes	**OK** is always returned	
Example	AT+GSN=FEDCBA0987654321 **OK** AT+GSN? **+GSN:FEDCBA0987654321** **OK** ATI4 **FEDCBA0987654321** **OK**	Just an alias to I4
Default value	0000000000000000 Notes: 1. If extended address is equal to zero then upon power up or reset SerialNet searches for the MAC address on 1-wire interface and applies it if detected. 2. User-defined MAC address shall be a non-zero values less than 0xFFFFFFFFFFFFFFFF (these values are reserved).	
Persistence	value is stored in EEPROM	
Node types	Coordinator / Router / End device	

6.2.8 "+WSRC" - Set/Get short (NWK) address

Table 6-19. "+WSRC" - Set/Get short (NWK) address

Syntax	Explanation
`+WSRC=<value>`	The command assigns device short (network) address. `value` is a 16-bit hexadecimal number which will be used by the device for communication in the network. It shall be unique within the network. Notes: 1. The command is not accessible when the node is joined to a network. 2. Coordinator node shall always have short address set as 0000. Nodes of other roles shall have non-zero short addresses.
`+WSRC?`	The command returns device short address in form of 16-bit hexadecimal number.
`+WSRC=?`	The command requests the range of valid addresses.
S-register	S55 (RW).
Result codes	**OK** is returned if `value` is in range, otherwise **ERROR** is returned.

Example	`AT+WSRC=2ABC`
	OK
	`AT+WSRC?`
	+WSRC:2ABC
	OK
	`AT+WSRC=?`
	+WSRC:(0000-FFF7)
	OK

Default value	FFFF Note: The default value is outside the allowed range, which means that the device will not join the network unless provided with the user-defined short (network) address.
Persistence	`addr` value is stored in the EEPROM.
Node types	Coordinator / Router / End device

6.3 Network Management Functions

SerialNet commands described in this section execute various network management functionality including network join and leave operations, obtaining network topology-related information, getting link quality data, etc.

When exploring network topology it is important to take into account the fact that due to mesh networking only an end device node can be a child and has a dedicated parent node (coordinator or router) during its lifetime in the network. Router nodes use coordinator or other routers only as network entry points and are not associated as direct children after network join. However, if there is enough space in node's neighbor table it will contain information about neighbor coordinator/router nodes.

6.3.1 "+WJOIN" - Start/Join to the network

Table 6-20. "+WJOIN" - Start/Join to the network

Syntax	Explanation
`+WJOIN`	The command forces device to form a network (for Coordinator node) or to join an existing network (for Router or End device nodes). Desired network and device characteristics shall be set prior to `+WJOIN` request using if necessary SerialNet commands from "Networking Parameters" on page 6-7.
Result codes	**OK** is returned if network formation/join is completed successfully; **ERROR** is returned if failed. If the node is in the network already, it returns **OK** immediately.
Example	`AT+WJOIN` **OK**
Node types	Coordinator / Router / End device

6.3.2 "+WLEAVE" - Leave the network

Table 6-21. "+WLEAVE" - Leave the network

Syntax	Explanation
`+WLEAVE`	The command forces the node to leave the network. If node has any children it will automatically force them to leave the network as well. Note: Parameters stored in EEPROM persist even after the node leaves the network.
Result codes	**OK** is returned on the process completion. If the device was not connected before starting the process, it returns **ERROR** immediately.
Example	`AT+WLEAVE` **OK**
Node types	Coordinator / Router / End device

6.3.3 "+WNWK" – Get networking status

Table 6-22. "+WNWK" – Get networking status

Syntax	Explanation	
`+WNWK`	The command requests current networking status of the device.	
Result codes	**OK** is returned if the device is joined to a network, otherwise it returns **ERROR**.	

Example	`AT+WLEAVE`
	OK
	`AT+WNWK`
	ERROR
		Leave the network first
		
		Device is not in a network now

Node types	Coordinator / Router / End device	

6.3.4 +WPARENT" - Get parent address

Table 6-23. "+WPARENT" - Get parent address

Syntax	Explanation
+WPARENT?	The command requests parent node address the device is associated to. Extended (MAC) address of the parent node is returned as a 64-bit hexadecimal number if S30 register is set to 0. Short (NWK) parent address is returned if S30 register is set to 1. See Table 6-27) for details. Note: This command does not cause network operations and just returns a copy of the parent address assigned during the joining process.
Result codes	**OK** is returned if the module is in the network and has a parent. **ERROR** will be returned if the device is not in the connected state or has node role Coordinator or Router.
Example	AT+WPARENT? **+WPARENT:0123456789ABCDEF** **OK**
Node types	End devices

6.3.5 “+WCHILDREN” – Get children addresses

Table 6-24. “+WCHILDREN” – Get children addresses

Syntax	Explanation
+WCHILDREN?	The command requests addresses of children end devices associated to the node. Extended (MAC) addresses of children nodes are returned as 64-bit hexadecimal numbers if S30 register is set to 0. Short (NWK) addresses of children nodes are returned if S30 register is set to 1. See Table 6-27for details. Children addresses are returned delimited by commas. Notes: 1. An end device is removed from the children list if the parent node receives no poll requests from the child during *3*(sleep_interval + sync_period)* time interval as configured on parent device by +WPWR and +WSYNCPRD commands. 2. This command does not cause network operations and just returns copies of the children addresses stored in the parent memory.
Result codes	**OK** is returned if the module is in the network even though there is no child connected yet. **ERROR** will be returned if the device is not in the connected state or has End device node role.
Example	AT+WCHILDREN? **+WCHILDREN:0123456789ABCDEF,123456789ABCDEF0** **OK**
Node types	Coordinator and Routers

6.3.6 "+WNBSIZE" - Get number of neighbors

Table 6-25. "+WNBSIZE" - Get number of neighbors

Syntax	Explanation
+WNBSIZE?	The command requests a number of entries in node’s neighbor table. Returned result consists of two values: the first one is the current number of occupied entries in node’s neighbor table; the second value is the maximum possible number of entries (size of the neighbor table).

Table 6-25. "+WNBSIZE" - Get number of neighbors

Result codes	**OK** is returned if the node is in the network. If device is not in the connected state **ERROR** will be returned.
Example	AT+WNBSIZE? **+WNBSIZE:2,5** **OK**
Node types	Coordinator / Router / End device

6.3.7 "+WNB" - Get neighbor information

Table 6-26. "+WNB" - Get neighbor information

Syntax	Explanation
+WNB <node_role> [,<device_addr>]	The command requests content of node's neighbor table. node_role parameter specifies node role of neighboring nodes to be extracted from the neighbor table. Following values are accepted: 0 – coordinator 1 – router 2 – end device 3 – all device types Optional parameter device_addr specifies the address of the neighboring node to be extracted. If S30 register is set to 0. device_addr is accepted as short (NWK) address if S30 register is set to 1, device_addr is expected to be an extended (MAC) address. See Table 6-27 for details. The command's information response has the following format: seqNr ¦ nodeRole ¦ extAddr ¦ nwkAddr ¦ relationship ¦ depth where seqNr – is the sequence number in the neighbor table nodeRole – is the node role of the neighbor extAddr – is neighbor's extended address nwkAddr – is neighbor's network address relationship – is neighbor's relationship to current node (0-parent, 1 – child, 3 – no relationship) depth – is neighbor's network depth Notes: 1. A neighbor entry is removed from the table if the node during certain interval doesn't receive any periodic management frames expected from the neighbor. If neighbor is a router/coordinator this interval is 45 seconds (management frames are sent once per 15 sec). If neighbor is an end device then interval equals *3*(sleep_interval +sync._period)* as configured on the node by +WPWR and +WSYNCPRD commands. 2. Although right after network join an end device node can have information about several nodes in its neighbor table, only actual parent node persists in the table while information about other nodes is removed shortly after end device join. Same is valid for information about an end device neighbor – in long term period it is present only in the neighbor table of its parent and is not directly "visible" for other routers in its neighborhood. 3. This command does not cause network operations and just returns information from node's current neighbor table.

Table 6-26. "+WNB" - Get neighbor information

Result codes	**OK** is returned if the node is in the network. If the node is not in the connected state **ERROR** will be returned.
Example	AT+WNB 3 **1 | 0 | 0000000000000001 | 0000 |3 | 2** **2 | 1 | 0000000000000002 | 0002 |0 | 1** **OK** AT+WNB 1,2 **1 | 1 | 0000000000000002 | 0002 |0 | 1** **OK**
Node types	Coordinator / Router / End device

6.3.8 "S30" - Set node addressing mode

Table 6-27. "S30" - Set node addressing mode

Syntax	Explanation
S30=<value>	The command sets the node addressing scheme to be used by some SerialNet commands. value: specifies addressing mode 0 -extended (64-bit) addressing 1- short (16-bit) addressing
S30?	The command requests current addressing mode.
Result codes	The command returns **OK** if value is in range, otherwise **ERROR**.
S-register	S30 (RW)
Example	ATS30=0 **OK** AT+WPARENT? **+WPARENT:000100000A3B98CC** **OK** ATS30=1 **OK** AT+WPARENT? **+WPARENT:0000** **OK**
Node types	Coordinator / Router / End device
Default value	0
Persistence	value is NOT stored in EEPROM

Note: Setting the addressing mode, the S30 command affects the performance of the following commands: +WPARENT? (see Section Table 6-23), +WCHILDREN? (see Table 6-24), and +WNB (see Table 6-26). These commands use extended (MAC) address if S30 is set to 0, but will switch to using short (NWK) addressing if S30 is set to 1.

6.3.9 "+WLQI" - Get LQI value

Table 6-28. "+WLQI" - Get LQI value

Syntax	Explanation	
`+WLQI <addr>`	The command requests LQI for the link to the node with short (NWK) address equal to `addr` specified in 16-bit hexadecimal format. The command returns the actual LQI value in the range of `0…255`. Notes: 1. LQI information can be retrieved for links within one-hop radius only. 2. An end device can obtain LQI only to its current parent node and vice versa: LQI to an end device can be obtained only from its current parent node. 3. LQI value is measured during data transmission initiated by `ATD` command. If `ATD` has not been performed yet, `+WLQI` may return irrelevant value.	
Result codes	The node returns **`OK`** if device is in the network and LQI value for this particular link exists, otherwise **`ERROR`** will be returned.	
Example	`AT+WLQI 1` **`+WLQI:254`** **`OK`**	request LQI for the link to the node with short address 0x0001
Node types	Coordinator / Router / End device	

6.3.10 "+WRSSI" - Get RSSI value

Table 6-29. "+WRSSI" - Get RSSI value

Syntax	Explanation	
`+WRSSI <addr>`	The command requests RSSI value for the link to the node with short (NWK) address equal to `addr` specified in 16-bit hexadecimal format. The command returns the actual RSSI value expressed in dBm. If RSSI is not available, then **-91** value is returned. Notes: 1. RSSI information can be retrieved for links within one-hop radius only. 2. An end device can obtain RSSI only to its current parent and vice versa: RSSI to an end device can be obtained only from its current parent node. 3. RSSI value is measured during data transmission initiated by `ATD` command. If `ATD` has not been performed yet, `+WRSSI` may return irrelevant value.	
Result codes	The node returns **`OK`** if device is in the network and RSSI value for this particular link exists, otherwise **`ERROR`** will be returned.	
Example	`AT+WRSSI 0001` **`+WRSSI:-80`** **`OK`**	request RSSI for the link to the node with short address 0x0001 -80 dBm
Node types	Coordinator / Router / End device	

6.4 Data Transmission

In SerialNet data can be transmitted in two ways:

- "Unicast transmission to a particular node using the D, DS or +WPING commands;
- "Broadcast transmission to all nodes using the DU command or D command with broadcast address.

It is important that extended (MAC) addresses are not used for data transmission directly; instead, they are substituted by short (network) addresses which are convenient for node replacement in network installation and maintenance.

Route establishment procedure to the target node is implemented inside the stack. It is executed automatically upon data transmission request and then if a route exists, data delivery (one-hop or multi-hop) is performed to the destination node.

Following application identifiers are used in SerialNet for all data exchange operations:

- "Profile ID: 0xC31A
- "Endpoint ID: 0x01
- "Cluster ID: 0x00

Note: **To ensure data transmission safely over serial interface between a host and an MCU, it is strongly recommended setting hardware flow control (see** Table 6-60 on page 6-38**for details). When running terminal software to control the node, the chosen COM port should be set with the Hardware flow control option selected.**

6.4.1 Parent polling mechanism

Data delivery to an end device over the last hop (i.e. from the parent node to the child) is performed using polling mechanism described below.

Upon frame reception destined for its child node or broadcast frame with non-exhausted transmission radius and destination address equal 0xFFFF, parent node buffers the frame and waits for poll request from the child. The maximum waiting time is (sleep_interval+3*sync._period) as configured on the parent by +WPWR and +WSYNCPRD commands.

In awake state an end device polls its parent node periodically every +WSYNCPRD ms (as configured on end device). Parent node can transmit a data frame to a child only after receiving corresponding data poll from it. After data frame reception is completed the end device issues another data poll request to verify whether there are any frames buffered at the parent.

6.4.2 "D" - Send data to a specific node

Table 6-30. "D" - Send data to a specific node

Syntax	Explanation	
`D <addr>[,[<arq>] [,<length>]] <data>`	The command sends data to a specific node. `addr` is a 16-bit hexadecimal short (network) address of the destination node. Optional `arq` parameter (equals to 1 or 0) controls ARQ/nonARQ data delivery mode, meaning 1 (i.e. ARQ) as default if omitted. `length` means the length in bytes of the data portion to be sent. The data portion length shall not exceed the maximum allowable number (84 characters). If length parameter is omitted, the maximum allowable number is implied by default. Data transmission starts either when the specified number of data bytes is received over serial interface or when the time interval between two consecutive data symbols exceeds the time-out preset (`+WWAIT` command - seeTable 6-37 on page 6-23). Notes: 1. `data` should be preceded by `<CR>` (S3 character, see Table 6-51 on page 6-32). This symbol is not transmitted over the air and it is not counted in `length`. 2. If the destination address is a broadcast address (FFFF for all nodes or FFFE for router/coordinator nodes), the broadcast transmission is performed.	
Result codes	If acknowledgement is requested (`arq` is set to 1), the node responds with **OK** upon receiving an acknowledgement in several attempts (see parameter `+WRETRY` in Table 6-36 on page 6-23)), otherwise it returns **ERROR**. If the destination node or the sending node itself is not in the network **ERROR** is returned.	

Example	`ATD 12,1,5`
	`HELLO`
	OK
	`ATD 12`
	`HELLO`
	OK
		Send `HELLO` to the node with address 12 using ARQ.
		
		The same as above, but the node will be waiting for the time-out expiration before going to the air.

Node types	Coordinator / Router / End device	

6.4.3 "DU" - Send broadcast data

Table 6-31. "DU" - Send broadcast data

Syntax	Explanation
`DU [<length>] <data>`	The command sends `data` using broadcast transmission. `length` means the length in bytes of the `data` portion to be sent. The `data` portion may not exceed the maximum allowable number (84 characters). If `length` parameter is omitted, the maximum allowable number is implied by default. Data transmission starts either when the specified number of data bytes is received over serial interface or when the time interval between two consecutive data symbols exceeds the time-out preset (`+WWAIT` command,Table 6-37 on page 6-23). Notes: 1. `ATDU` is, in fact, shorthand for `ATD` command with broadcast address (FFFF) as destination. 2. Data should be preceded by `<CR>` (S3 character, see Table 6-51 on page 6-32). This symbol is not transmitted over the air and it is not counted in length. 3. Data is broadcasted to the whole network (radius 0).

Table 6-31. "DU" - Send broadcast data

Result codes	The node responds with **OK** immediately after the transmission if the node itself is in the network. Otherwise, **ERROR** is returned.	
Example	ATDU HELLO **OK**	Send **HELLO** to all nodes in the network
Node types	Coordinator / Router / End device	

6.4.4 "DS" - Send S-register value to a specific node

Table 6-32. "DS" - Send S-register value to a specific node

Syntax	Explanation	
DS <S-reg>,<addr> [,[<arq>]	The command sends S-register value to a specific node. Default arq parameter (is set to 1 or 0) specifies whether the ARQ or non-ARQ data delivery mode is used. 1 is implied if arq is omitted. Destination node address addr should be a 16-bit hexadecimal short (network) address. S-register data is sent in the form readable by ATS command without the line termination characters. Note: S-registers defined by user extensions are also accessible by this command.	
Result codes	If acknowledgement is requested (arq is set to 1), the node responds with **OK** upon receiving acknowledgement in several attempts (see parameter +WRETRY, Table 6-36 on page 6-23), otherwise it returns **ERROR**. If the destination node or the sending node itself is not in the network **ERROR** is returned. Also, if the specified S-register can not be read, the command returns **ERROR** and the node does not send anything to the air.	
Example	ATDS130,2,0 **OK**	Send GPIO0 value to the node with address 2 without using ARQ.
Node types	Coordinator / Router / End device	

6.4.5 "+WPING" - Ping the node

Table 6-33. "+WPING" - Ping the node

Syntax	Explanation
+WPING <addr>	The command pings the targeted node. addr specifies destination address as 16-bit hexadecimal short (network) address. This command is equivalent to D command with ARQ and zero data length: ATD <addr>,1,0.
Result codes	The node responds with **OK** upon receiving acknowledgement in several attempts (see parameter +WRETRY, Table 6-36 on page 6-23) , otherwise it returns **ERROR**. If the destination node or the sending node itself is not in the network **ERROR** is returned.
Example	AT+WPING 1 **OK**
Node types	Coordinator / Router / End device

6.4.6 "+WSYNCPRD" - Poll rate for requesting indirect transactions from the parent

Table 6-34. "+WSYNCPRD" - Poll rate for requesting indirect transactions from the parent

Syntax	Explanation	
+WSYNCPRD=<rate	The command sets poll interval to the <rate> value measured in milliseconds. This value is used by the End device as the poll rate for requesting indirect transmission messages from the parent. Coordinator and router use this rate to verify children presence. Notes: 1. On End devices, the <rate> value must not be increased by this command. Otherwise, BitCloud behavior is unpredictable. 2. On routers and coordinators this parameter must be set to the largest <rate> value among all children. Otherwise, child presence status may be detected incorrectly. 3. This value should be at least 2 times smaller than the value of +WTIMEOUT (see Table 6-35 on page 6-22). 4. The command is not accessible when the node is joined to a network.	
+WSYNCPRD?	The command requests the actual poll rate.	
+WSYNCPRD=?	The command requests allowable range of poll rate values.	
S-registers	S37 (RW).	
Result codes	**OK** is always returned.	
Example	AT+WSYNCPRD=1000 **OK** ATS37? **300** **OK** AT+WSYNCPRD=? **+WSYNCPRD:(10-30000)** **OK**	Set poll rate to 1 sec
Default values	1400	
Node types	Coordinator / Router / End device	
Persistence	rate is NOT stored in EEPROM	

6.4.7 "+WTIMEOUT" - Data delivery time-out

Table 6-35. "+WTIMEOUT" - Data delivery time-out

Syntax	Explanation
+WTIMEOUT?	The command returns the time-out value in milliseconds. The returned value corresponds to the apscAckWaitDuration variable introduced by ZigBee recommendation Step 2. in "Related Documents and References" on page 2-1.
S-register	S51 (R).
Result codes	**OK** is always returned
Example	AT+WTIMEOUT? **+WTIMEOUT:2800** **OK**
Node types	Coordinator / Router / End device

6.4.8 "+WRETRY" - Repetition count

Table 6-36. "+WRETRY" - Repetition count

Syntax	Explanation
+WRETRY?	The command returns actual number of retransmission. The returned value corresponds to the apscMaxFrameRetries variable introduced by ZigBee recommendation .
S-register	S52 (R).
Result codes	**OK** is always returned
Example	AT+WRETRY? **+WRETRY:3** **OK**
Node types	Coordinator / Router / End device

6.4.9 "+WWAIT" - Data transmission waiting time-out

Table 6-37. "+WWAIT" - Data transmission waiting time-out

Syntax	Explanation
+WWAIT=<value>	The value parameter sets the time-out (in milliseconds) for the module to wait for entering the D (see Table 6-30 on page 6-20) or the DU (see Table 6-31 on page 6-20) command. If a pause between two consecutive characters coming from serial interface exceeds the specified time-out, the node will start data transmission even though the data length encountered has not yet reached the number specified by the length argument of the D/DU command. In such case, the length is replaced with its actual value according to the data transmitted.
+WWAIT?	The command returns actual time-out value.
+WWAIT=?	The command requests for the range of valid time-outs.
S-register	S53 (RW).
Result codes	**OK** is returned if the value is in range, otherwise **ERROR** is returned.
Example	AT+WWAIT=500 **OK** AT+WWAIT? **+WWAIT:500** **OK** AT+WWAIT=? **+WWAIT:(100-500)** **OK**
Default value	5000
Persistence	value is stored in the EEPROM.
Node types	Coordinator / Router / End device

6.5 Power Management

Because power consumption is a major concern in applications with battery-powered devices, SerialNet provides AT-commands that allow switching between awake and sleep modes as well as setting transmit power level.

Note that sleep mode is supported on end device nodes designed on ZigBit modules only and is not available for nodes using RZUSBSTICK platform. To avoid issues in network stability coordinator and router nodes are always kept in active mode and hence require continuous power supply.

In addition to power management of ZigBit module SerialNet simplifies power management of external peripherals or the host device via CTS line. If hardware flow control is enabled by +IFC command (see Table 6-60 on page 6-38), the line becomes high when the ZigBit module is in the sleep state.

6.5.1 "+WPWR" - Configuration of sleep/active intervals

Table 6-38. "+WPWR" - Configuration of sleep/active intervals

Syntax	Explanation	
+WPWR=<sleep>, <active>	The command sets duration of sleep and active intervals for end device node. sleep duration is specified in 100 msec units but active duration – in 10 msec units. Zero active period means that the node can be put asleep only explicitly by +WSLEEP command (in which case it will stay asleep for given sleep duration). On a coordinator/router node sleep interval is used for children tracking and should be not less than on its children nodes. It is also used as maximum time interval the data destined for the child can be buffered for. See "Parent polling mechanism" on page 6-19 for more details. Note: 1. Actual sleep/active periods will be slightly different and their values depend on multiple circumstances such as the network activity, external interfaces to the sensors, and so on. They can not be used for absolute timing. 2. The command is not accessible when the node is joined to a network.	
+WPWR?	The command requests current sleep/active intervals.	
+WPWR=?	The command requests valid ranges of sleep/active intervals.	
S-registers	S31, S32 (RW).	
Result codes	**OK** is returned if parameters are within their valid ranges. Otherwise **ERROR** is returned.	
Example	AT+WPWR=600,10 **OK** AT+WPWR? **+WPWR:600,10** **OK** ATS31? **600** **OK** AT+WPWR=? **+WPWR:(2-30000),(0-30000)** **OK**	Set duty cycle 1 min. sleep / 100 msec active Verify setting is applied Get sleep interval via S-register Get valid ranges for sleep/active intervals
Default values	100,0 (the node sleeps for 10 seconds if put asleep by +WSLEEP command)	
Persistence	The sleep, active values are stored in the EEPROM.	
Node types	Coordinator / Router / End device	

6.5.2 "+WSLEEP" - Force node to sleep

Table 6-39. "+WSLEEP" - Force node to sleep

Syntax	Explanation
+WSLEEP	The command forces the node to fall into the sleep mode. The command is supported on ZigBit modules only and is not available on RZUSBSTICK. Important: Take in mind that the node in sleep mode can respond to the subsequent commands with a delay, depending on the sleeping interval specified (see Table 6-38 on page 6-24), the node version and DTR configuration (see Table 6-61 on page 6-39). The command is accessible only when the node is joined to a network.
Result codes	**OK** is returned for End devices, otherwise **ERROR**. Note: The command is executed as follows: the node returns the result code first, and then it disables any of subsequent commands, completes pending operations and finally falls into the sleep mode. Wake-up occurs as scheduled by +WPWR command or DTR interrupt if enabled.
Example	AT+WSLEEP **OK**
Node types	End devices

6.5.3 "+WTXPWR" - TX power level

Table 6-40. "+WTXPWR" - TX power level

Syntax	Explanation	
+WTXPWR=<value>	The command sets transmit power level for the device. The value represents TX power level measured in dBm. Note: This setting will be applied to the radio circuitry during the warm reset procedure only. Thus, the accurate setting of TX power requires warm reboot of the node in using Z command, see Table 6-41 on page 6-26.	
+WTXPWR?	The command requests actual Tx power level. Notes: 1. Power level resolution is hardware dependent and may be coarser than 1 dB, so that some power values (say, -4, -6, -8…) may be forbidden, even despite being within the allowed range. On input, such values are rounded to the nearest allowed value. 2. This command just returns the number set by the +WTXPWR= command, but does not indicate real power level, which can vary due to the temperature, supply voltage and another factors.	
+WTXPWR=?	The command requests the allowable range of TX power level.	
S-register	S34 (RW).	
Result codes	**OK** is returned if value is in the valid range, otherwise **ERROR**.	
Example	AT+WTXPWR=-5 **OK** AT+WTXPWR? **+WTXPWR:-5** **OK** AT+WTXPWR=? **+WTXPWR:(-17-3)** **OK**	set -5dBm Tx power level

Table 6-40. "+WTXPWR" - TX power level

Default value	Hardware dependent, typically 3
Persistence	`value` is stored in the EEPROM.
Node types	Coordinator / Router / End device

6.6 Generic Control

6.6.1 "Z" - Warm reset

Table 6-41. "Z" - Warm reset

Syntax	Explanation
`Z`	The command instructs the device to execute warm (software) reset. This command resets the hardware, restores all persistent variables from EEPROM and restarts the firmware. The command is supported on ZigBit modules only and is not available on RZUSBSTICK. Important: The command should be used with precautions since it does not send 'leaving the network' signals to other nodes and hence can affect PAN's integrity. Therefore, the node should better be put out of the network by the `+WLEAVE` command prior to reset. If automatic networking is disabled then the node will not join the network automatically after reset. Note that the parameters stored in EEPROM persist after software reset; to erase them, use the `AT&F` command (see Table 6-48 on page 6-30). If `Z` is put in a line together with some other commands, the processing of those placed after `Z` is disabled. Result code is sent upon the reset process is completed. During the reset process some transients can be observed on the module pins (including GPIO) because of the nature of the MCU used. It is strongly recommended to wait until **`OK`** result code is received (or an equivalent numerical code 0 if verbose result codes are disabled by `V0` command, see Table 6-56 on page 6-35) before sending any new command to the module.
Result codes	**`OK`** is returned if command is supported for the device's platform. **`ERROR`** is returned otherwise.
Example	`ATZ` **`OK`**
Node types	Coordinator / Router / End device

6.6.2 "&H" - Command Help

Table 6-42. "&H" - Command Help

Syntax	Explanation
`&H`	The command outputs a list of valid AT-commands. The listing order may change. It depends on firmware version.
Result codes	**OK** is always returned

Example	`AT&H`
	E
	V
	Q
	Z
	&F
	+IPR
	+IFC
	&D
	&H
	%H
	I
	+GMI
	+GMM
	+GMR
	+GSN
	`(skipped…)`
	S146
	S147
	S148
	OK

Node types	Coordinator / Router / End device

6.6.3 "%H" - Display parameters and S-register values

Table 6-43. "%H" - Display parameters and S-register value

Syntax	Explanation
%H	The command outputs the values of parameters and S-registers. The listing order may change. It depends on firmware version.
Result codes	**OK** is always returned
Example	AT%H **+WPANID: 0000000000000000** **+WCHAN: FF** **+WCHMASK: 00000800** **+WAUTONET: 0** **+WPWR: 100,1000** **+WROLE: 2** **+WSRC: 0001** **+WSYNCPRD: 1400** **+WTXPWR: 0** **+WTIMEOUT: 2800** **+WRETRY: 3** **+WWAIT: 5000** **E: 1** **Q: 0** **V: 1** **X: 0** **+IPR: 38400** **+IFC: 0,0** **+GMI: ATMEL** **+GMM: ZIGBIT** **+GMR: BitCloud v.1.5.0; SerialNet v.2.2.0** **+GSN: 0001000011672CFC** *(skipped…)* **S146:0** **S147:0** **S148:0** **OK**
Node types	Coordinator / Router / End device

6.6.4 "I" - Display product identification information

Table 6-44. "I" - Display product identification information

Syntax	Explanation		
`I[<value>]`	The command instructs the node to return information text identifying the device. Information text depends on the `value` as follows:		
	value	Information text	Reference
	0 1 2 3 4	All the identifiers below Manufacturer identifier Model identifier Hardware/software revision identifier Product serial number identifier	 Section 6.6.5 Section 6.6.6 Section 6.6.7 Section 6.2.7
	If `value` is omitted, 0 is implied by default.		
Result codes	**OK** for any of the aforementioned values, **ERROR** otherwise.		
Example	`ATI0` **ATMEL** **ZIGBIT** **BitCloud v.1.5.0; SerialNet v.2.2.0** **000100001090C3F9** **OK**		
Node types	Coordinator / Router / End device		

6.6.5 "+GMI" - Get manufacturer identifier

Table 6-45. "+GMI" - Get manufacturer identifier

Syntax	Explanation	
`+GMI?` `I1`	The command instructs the node to output information text i identifying the manufacturer.	
Result codes	**OK** is always returned	

Example	`AT+GMI?`
	+GMI:ATMEL
	OK
	`ATI1`
	ATMEL
	OK
		Just an alias to `+GMI`

Node types	Coordinator / Router / End device	

6.6.6 "+GMM" - Request for the model identifier

Table 6-46. "+GMM" - Request for the model identifier

Syntax	Explanation	
+GMM? I2	The command instructs the node to transmit information text identifying the particular model of the device.	
Result codes	**OK** is always returned	
Example	AT+GMM? **+GMM:ZIGBIT** **OK** **ATI2** **ZIGBIT** **OK**	Just an alias to +GMM
Node types	Coordinator / Router / End device	

6.6.7 "+GMR" - Request for the hardware/software revision identifier

Table 6-47. "+GMR" - Request for the hardware/software revision identifier

Syntax	Explanation	
+GMR? I3	This command instructs the node to transmit an information text intended to identify the actual revision of hardware or software product burned into the device.	
Result codes	**OK** is always returned	
Example	AT+GMR? **+GMR: BitCloud v. 1.5.0; SerialNet v.2.2.0** **OK** ATI3 **+GMR: BitCloud v. 1.2 5.0; SerialNet v.2.2.0** **OK**	Just an alias to +GMR
Node types	Coordinator / Router / End device	

6.6.8 "&F" – Set to factory-default configuration

Table 6-48. "&F" – Set to factory-default configuration

Syntax	Explanation
&F	The command instructs the module to set all the parameters (including the persistent variables from EEPROM) to the factory defaults. This command forces hardware reset like the Z command does, so all the precautions in should be considered. Result code will be issued according to actual result code suppression setting (see Table 6-55 on page 6-35), response formatting (see Table 6-56 on page 6-35) and the transmission rate (see Table 6-59 on page 6-37) set before execution of this command. Note that &F command does not reset the password, once it has been set by the +WPASSWORD command (see Table 6-70 on page 6-46).

Table 6-48. "&F" – Set to factory-default configuration

Result codes	**OK** is always returned
Example	AT&F **OK**
Node types	Coordinator / Router / End device

6.6.9 "+WACALIBRATE" - Configure periodic internal clock calibration

Table 6-49. "+WACALIBRATE" - Configure periodic internal clock calibration

Syntax	Explanation
+WACALIBRATE=<value>	The command requests the device to automatically calibrate the internal clock. value is an unsigned integer between 0 and 65535 which determines the period of calibration in minutes (i.e. how many minutes elapse between consecutive calibrations). The command is supported on ZigBit modules only and is a no-op on RZUSBSTICK. It can be used to prevent frequency drift of MCU's internal RC-oscillator that can impact or even block serial communication with the host.
+WACALIBRATE?	The command returns the period of calibration (in minutes).
+WACALIBRATE=?	The command returns permitted range of values for the period of calibration.
Result codes	**OK** is returned on successful command completion Otherwise, value is ignored and device responds with **ERROR**.
Example	AT+WACALIBRATE=60 **OK** AT+WACALIBRATE? **+WACALIBRATE:60** **OK** AT+WACALIBRATE=? **+WACALIBRATE(0-65535)** **OK**
Default value	0
Persistence	The value is stored in the EEPROM.
Node types	Coordinator / Router / End device

6.6.10 "+WCALIBRATE" - Calibrate internal clock

Table 6-50. "+WCALIBRATE" - Calibrate internal clock

Syntax	Explanation
+WCALIBRATE	The command requests the device to calibrate the internal clock. The command is supported on ZigBit modules only and is a no-op on RZUSBSTICK. It can be used to prevent frequency drift of MCU's internal RC-oscillator that can impact or even block serial communication with the host.

Table 6-50. "+WCALIBRATE" - Calibrate internal clock

Result codes	**OK** is returned on successful calibration. Otherwise, device responds with **ERROR**.
Example	AT+WCALIBRATE **OK**
Node types	Coordinator / Router / End device

6.7 Hot Interface Commands

6.7.1 "S3" - Termination character

Table 6-51. "S3" - Termination character

Syntax	Explanation
S3=<value>	The command sets ASCII code to be used as termination character in command line, response and result code formatting. value may be specified in the range of 0…127. Note: It is strongly recommended to avoid changing of this parameter during the network operation.
S3?	The command requests for actual ASCII code currently used as the termination character.
Result codes	The module returns **OK** if value is in range, otherwise **ERROR**. Important: It is the previous value of S3 which is used in entering the command line containing the S3 setting command to specify the next command line termination character. However, the result code when issued will use the value of S3 as that one set during the processing of the command line. For example, if S3 was previously set to 13 and the 'ATS3=30' command line is issued, the command line will be terminated with a CR character, but the result code when issued will use the character with the decimal value 30 instead of **<CR>**.
Example	ATS3=13 **OK** ATS3? **13** **OK**
Node types	Coordinator / Router / End device
Default value	13 - **<CR>** (carriage return character)
Persistence	value is stored in the EEPROM.

6.7.2 "S4" - Response formatting character

Table 6-52. "S4" - Response formatting character

Syntax	Explanation
S4=<value>	The command sets ASCII code of character to be used in responses and result code formatting along with the S3 parameter (see Table 6-51 on page 6-32). The description of V command shows the parameter usage, see Table 6-56 on page 6-35 for details. value may be specified in the range of 0…127. Note: It is strongly recommended to avoid changing of this parameter during the network operation.
S4?	The command requests for actual ASCII code currently used as the response formatting character.
Result codes	The module returns **OK** if value is in the allowed range, and **ERROR** otherwise. Note: The changed value of S4 will be used in formatting of the result code and information responses immediately after processing the 'S4=<value>' command. If the value of S4 is changed in a command line, the result code issued in response to that command line will use the new value of S4.
Example	ATS4=10 **OK** ATS4? **10** **OK**
Node types	Coordinator / Router / End device
Default value	10 - <LF> (Line Feed character)
Persistence	value is stored in the EEPROM.

6.7.3 "S5" - Command editing character

Table 6-53. "S5" - Command editing character

Syntax	Explanation
S5=<value>	The command sets ASCII code to be used as the control character pointing to delete the just preceding character in the command line, see "Basic Command-Line Operations" on page 6-1. value may be specified in the range of 0…127. Note: It is strongly recommended not to set this parameter to any letter or other symbol that can be a part of a command. For example, setting it to letter A, either upper- or lowercase (ASCII code 65 or 97) would effectively prevent entering of any subsequent AT command.
S5?	The command requests for actual ASCII code of the command editing character.

Table 6-53. "S5" - Command editing character

Result codes	The module returns **OK** if `value` is in range, otherwise **ERROR**. Note: The changed value of S5 will be used in editing of subsequent command lines and will be applied after processing the line containing S5 register change.

Example	`ATS5=8`
	OK
	`ATS5?`
	8
	OK

Node types	Coordinator / Router / End device
Default value	`8 - <BS> (Backspace Character)`
Persistence	`value` is stored in the EEPROM.

6.7.4 "E" - Command echo

Table 6-54. "E" - Command echo

Syntax	Explanation	
`E[<value>]`	Setting this parameter instructs if the module should echo the characters received from UART. `value` may be specified as 0 or 1 to disable or enable echoing, correspondingly. If `value` is omitted 0 is implied by default.	
Result codes	The module returns **OK** if `value` is 0 or 1, otherwise **ERROR**.	
Example	`ATE` **OK**	Disable echo
	`ATE1` **OK**	Enable echo
Node types	Coordinator / Router / End device	
Default value	`1` - echoing is enabled	
Persistence	`value` is stored in the EEPROM.	

6.7.5 "Q" - Result code suppression

Table 6-55. "Q" - Result code suppression

Syntax	Explanation	
`Q[<value>]`	Setting this parameter instructs if the module should transmit the result codes to UART. When result codes are being suppressed, no portion of any intermediate, final, or unsolicited result code – header, result text, line terminator, or trailer (see “Parameter Persistence” on page 5-3, and) – is transmitted. Information text transmitted in response to a command is not affected by setting of this parameter. There are two possibilities for `value`: `0` The module transmits result codes. `1` Result codes are suppressed and so not transmitted. If `value` is omitted, 0 is implied.	
Result codes	Nothing will be received for `ATQ1` command, **OK** if `value` is 0, otherwise the module returns **ERROR**.	
Example	`ATQ0` **OK**	Enable the result codes
	`ATQ1`	Suppress the result codes. No **OK** will be sent because it is suppressed
Node types	Coordinator / Router / End device	
Default value	0 – enables result codes	
Persistence	`value` is stored in the EEPROM.	

6.7.6 "V" - Response format

Table 6-56. "V" - Response format

Syntax	Explanation	
`V[<value>]`	Setting this parameter defines the contents of header and trailer transmitted with result codes and information responses. It also determines whether result codes are transmitted in numeric, alphabetic, or "verbose", form. The text portion of information responses is not affected by this setting. shows the effect of the setting of this parameter on the format of information text and result codes. If `value` is omitted, `0` is implied.	
Result codes	**0**	If `value` is `0` (because numeric response text is being used)
	OK	If `value` is `1`.
	4	For unsupported values (if previous `value` was `0`).
	ERROR	For unsupported values (if previous `value` was `1`).

Example	`ATV1`
	OK
	`ATV0`
	0
		`0` will be output on the same line because <LF> is not used for formatting of result code!

Table 6-56. "V" - Response format

Node types	Coordinator / Router / End device
Default value	1 – verbose format
Persistence	value is stored in the EEPROM

Table Table 6-52 below summarizes the usage of response formats. All references to <CR> mean "the character ASCII coded as specified in parameter S3 (see Table 6-51 on page 6-32)"; all references to <LF> likewise mean "the character ASCII coded as specified in parameter S4 (see Table 6-52 on page 6-33)". Numeric and verbose codes are discussed in "Parameter Persistence" on page 5-3.

Table 6-57. Response formatting

Value	0	1
Information responses	**<text><CR><LF>**	**<CR><LF><text><CR><LF>**
Result codes	**<numeric code><CR>**	**<CR><LF><verbose code><CR><LF>**

6.7.7 "X" - Result code selection

Table 6-58. "X" - Result code selection

Syntax	Explanation	
X[<value>]	Setting this parameter defines whether the module transmits particular result codes (see "Parameter Persistence" on page 5-3) to the host, or it does not	
	value	Description
	0 1 2	all result codes are sent to the host **EVENT** result codes are not sent **EVENT** and **DATA** result codes are not sent
	If value is omitted, 0 is implied	
Result codes	**OK** if value is from valid range. Otherwise, **ERROR** is returned.	
Example	ATX2 **OK** — Disable events and data indications	
Node types	Coordinator / Router / End device	
Default value	1 – all result codes will be sent, excluding **EVENT**.	
Persistence	value is stored in the EEPROM.	

6.7.8 "+IPR" - Serial port communication rate

Table 6-59. "+IPR" - Serial port communication rate

Syntax	Explanation
`+IPR=<value>`	The command specifies the data rate at which the DCE will accept commands and will respond. At least, 1200 bit/s and 9600 bit/s are supported, but particular hardware version can support extended set of rates. Note: The rate specified takes effect following the issuance of any result code associated with the current command line even subsequent commands in a command line will return **ERROR**.
`+IPR?`	The command requests for actual communication rate.
`+IPR=?`	The command requests for the list of supported rates. This depends on the hardware capabilities of the particular model.
Result codes	The module returns **OK** if the requested rate is present in the supported list, otherwise **ERROR**.

Example	`AT+IPR=38400`
	OK
	`AT+IPR?`
	+IPR:38400
	OK
	`AT+IPR=?`
	+IPR:(1200,9600,38400)
	OK

Node types	Coordinator / Router / End device
Default value	38400
Persistence	`value` is stored in the EEPROM

6.7.9 "+IFC" - Serial port flow control

Table 6-60. "+IFC" - Serial port flow control

Syntax	Explanation	
`+IFC=<rx_flow>, <tx_flow`	The command is used to specify the methods for local flow control over the UART interface between the host and the module. It accepts two numeric sub-parameters: ■ `rx_flow`, which specifies the method for the host to control the flow of data received from the module ■ `tx_flow`, which specifies the method for the module to control the flow of data transmitted from the host	
	rx_flow	
	`0` `2`	None use RTS (Request to Send) line
	`tx_flow`	
	`0` `2`	None use CTS (Clear to Send) line
	Note: It is strongly recommended to use the CTS method because, if no flow control method is selected, there would be no means to use power-down modes when the module would not accept any data coming to UART.	
`+IFC?`	The command requests for actual flow control settings.	
`+IFC=?`	The command requests to list the flow control settings supported.	
Result codes	**`OK`** is returned if specified flow control combinations are supported, otherwise **ERROR**.	

Example	`AT+IFC=2,2`
	`OK`
	`AT+IFC?`
	`+IFC:2,2`
	`OK`
	`AT+IFC=?`
	`+IFC:(0,2),(0,2)`
	`OK`	

Node types	Coordinator / Router / End device	
Default value	Depends on the hardware version. For MeshBean2 boards it is `0,0`	
Persistence	`value` is stored in the EEPROM	

6.7.10 "&D" - DTR behavior

Table 6-61. "&D" - DTR behavior

Syntax	Explanation	
`&D<value>`	The command specifies the method how the module manages DTR line.	
	value	Description
	0 1	module ignores DTR line module wakes up if it is sleeping, thus it can process the data coming from UART with a shortest delay
S-register	S50 (RW).	
Result codes	**OK** is returned if the requested mode is supported, otherwise **ERROR**.	
Example	AT&D1 **OK**	
Node types	Coordinator / Router / End device	
Default value	0	
Persistence	`value` is stored in the EEPROM.	

6.7.11 S0 - Request for the latest result code

Table 6-62. S0 - Request for the latest result code

Syntax	Explanation	
`S0?`	Request for result code from the latest executed command. If the latest executed command was completed with **ERROR** result code, register `S0` will contain nonzero value. Returned values:	
	0 1 2 3 4 5 6 7 8	no error syntax error improper number of parameters parameter value(s) is out of range (example: `AT+IFC=12,34`) unspecified error requested value cannot be read (example: `+WLQI` command for non-existent link) operation is not permitted in current state (example: setting PAN ID in the connected state or `+WSLEEP` for router) operation cannot be completed due to networking problems, e.g. due to connection loss data transmission error

Table 6-62. S0 - Request for the latest result code

Result codes	Always **OK**	

Example	`AT+WROLE=0+WPWR=30,30`
	ERROR
	`ATS0?`
	6
	OK
		6 is returned as setting +WPWR is not permitted for coordinator

	`AT+ABCD`
	ERROR
	`ATS0?`
	1
	OK
		syntax error

	`AT+IFC=12,34`
	ERROR
	`ATS0?`
	3
	OK
		parameter is out of range

Node types	Coordinator / Router / End device	

6.8 Hardware Control

AT-commands described in this section are supported for ZigBit modules only and provide control over such hardware functionality as GPIO, ADC and PWM.

6.8.1 GPIO configuration

Table 6-63. GPIO configuration

Syntax	Explanation	
`S<reg>=<value`	Command selects configuration of particular GPIO pins. `reg` corresponds to GPIO pins, GPIO0…GPIO8, on the module and it is in the range of 120…128.	

	`value`
	
	`0`
	`3`
	`2`
	`1`
		Description
		
		input pin, no internal pull-up
		output
		tri-state
		input pin, internal pull-up is turned on

	Note: Using of internal pull-up improves noise immunity but take in mind that it results in power consumption increased. On the MeshBean2 board, tri-stated pins are configured as input with no pull-up.	
`S<reg>?`	The command requests for actual GPIO pin configuration.	
Result codes	**OK** is returned if the `value` is in valid range, otherwise **ERROR** is returned.	
Example	`ATS120=1 S121=3` **OK**	Set GPIO0 as input with internal pull-up and GPIO 1 as output

Table 6-63. GPIO configuration

Default value	2, tri-state
Persistence	Values are stored in the EEPROM.
Node types	Coordinator / Router / End device

6.8.2 GPIO

Table 6-64. GPIO

Syntax	Explanation	
`S<reg>=<value>`	he command assigns value to a particular GPIO pin. Each of pins GPIO0…GPIO8 of the module is numbered by `reg` which is in the range of 130…138, correspondingly.	

	`<value>`
	
	`0`
	`1`
		Description
		
		Logical 0
		Logical 1

	Note: Command does not affect any pin configured as input or tri-state.	
`S<reg>?`	The command reads a particular GPIO pin numbered and coded as above, so it returns 0 or 1. If pin is configured for output or as tri-state, returned value is not defined	
Result codes	**`OK`** is returned if `value` is 0 or 1, otherwise **ERROR** is returned.	

Example	`ATS120=1 S121=3`
	`ATS130?`
	`1`
	`OK`
	`ATS131=0`
	`OK`
		Set GPIO0 as input and GPIO1 as output, both with internal pull-up
		GPIO0 is 1
		
		Clear GPIO1

Default value	`0`	
Persistence	Values are not stored in the EEPROM because GPIO pins are configured as tri-state at the startup.	
Node types	Coordinator / Router / End device	

6.8.3 A/D configuration

Table 6-65. A/D configuration

Syntax	Explanation	
S100=<value>	The command selects configuration of particular A/D pins. value is a hexadecimal number containing a bit-field. 4 least significant bits (b0... b3) can be used to enable or disable each of 4 A/D channels. Bits b4... b7 are ignored in value field. If bit is cleared then A/D conversion of a corresponding channel is disabled and A/D pin goes to the high impedance without internal pull-up. Note: Take in mind that enabling A/D conversion increases power consumption. Note: Conversion is executed in single conversion mode (see ATmega datasheet with 125 kHz clock rate and external reference), thus enabling the maximum conversion rate of approximately 5 kbps. Note: Proper conversion results are achieved for ZigBit if the external reference signal of 1.25V is applied to the A_VREF pin. If conversion is disabled on all A/D pins, the A_VREF pin is moved to tri-state. Note: Pins AD4…AD7 can be also used as JTAG port and ADC function for this inputs are disabled. Note: When using the ZigBit module installed on the MeshBean2 board, the following restriction is imposed due to the board schematics. Before configuring or reading of the particular A/D pins, you must configure GPIO6, GPIO7 and GPIO8 for output, next set GPIO6 to 0 while setting GPIO7 and GPIO8 to 1. For example, you must send the following commands: ATS126=3 S127=3 S128=3 ATS136=0 S137=1 S138=1 before performing ATS100=0F See additionally Table 6-66 on page 6-43	
S<reg>?	The command requests for actual A/D configuration.	
Result codes	**OK** is always returned.	
Example	ATS100=08 **OK**	Enable conversion on pin AD3
Default value	00 – disable A/D conversion for all 4 A/D pins	
Persistence	Value is stored in the EEPROM.	
Node types	Coordinator / Router / End device	

6.8.4 A/D

Table 6-66. A/D

Syntax	Explanation	

`S<reg>?`	The command reads particular A/D pin and returns its value in decimal format. `reg` corresponds to pins AD0…AD3 on the module and it is in the range of 101…104. If A/D conversion for particular channel is disabled by the `S100` register, no value is returned.
	Note: When using the ZigBit module installed on the MeshBean2 board, the following restriction is imposed due to the board schematics. Configure GPIO 6, GPIO 7 and GPIO 8 for output. Set GPIO6 to 0 while setting GPIO7 and GPIO8 to 1. Then you can configure or read the particular A/D pins. For example, you must send the following commands:
	`ATS126=3 S127=3 S128=3`
	`ATS136=0 S137=1 S138=1`
	before performing these commands:
	`ATS100=0F`
	`ATS101? S102? S103? S104?`	

Result codes	**OK** is always returned.	

Example	`ATS100=08`
	OK
	`ATS104?`
	125
	OK
		Enable conversion on pin AD3
		
		Read AD3 pin

Node types	Coordinator / Router / End device	

6.8.5 PWM configuration

Table 6-67. PWM configuration

<table>
<tr><th>Syntax</th><th colspan="3">Explanation</th></tr>
<tr><td></td><td colspan="3">The command configures particular PWM channel:</td></tr>
<tr><td>S<reg>=<value></td><td>PWM channel
0
1
2</td><td>Output pin
GPIO0
GPIO1
GPIO2</td><td>reg
140
141
142</td></tr>
<tr><td></td><td><value>
0, 2
1

3</td><td colspan="2">Description
Disable PWM channel
Enable channel, setting non-inverted output polarity (output is low when duty cycle = 0% and it is high when duty cycle = 100%)
Enable channel, setting inverted output polarity (output is high when duty cycle = 0% and it is low when duty cycle = 100%)</td></tr>
<tr><td></td><td colspan="3">Notes: 1. When PWM channel is enabled, the corresponding output pin is configured as output to be controlled by that PWM channel. Duty cycle is set to 0 for the channel. PWM frequency is set for the channel to default value (5kHz) if there was no channel opened, otherwise that very frequency is valid for the channel which has been set the last for any other channel.
2. When PWM channel is disabled by setting reg to 0 or 2, the corresponding output pin is configured as tri-state, so it is fully controlled as GPIO.
3. On MeshBean2 board, GPIO0...GPIO2 pins are connected to LEDs</td></tr>
<tr><td>Result codes</td><td colspan="3">OK is returned if the value is in valid range, otherwise ERROR is returned.</td></tr>
<tr><td>S<reg>?</td><td colspan="3">The command requests for current PWM configuration.</td></tr>
<tr><td>Result codes</td><td colspan="3">OK is always returned.</td></tr>
<tr><td>Example</td><td>ATS140=1
S142=3
OK</td><td colspan="2">Enable PWM channel 0, setting non-inverted polarity output, and enable PWM channel 2, setting inverted polarity output.</td></tr>
<tr><td>Default value</td><td colspan="3">0, disabled</td></tr>
<tr><td>Persistence</td><td colspan="3">value is not stored in the EEPROM.</td></tr>
<tr><td>Node types</td><td colspan="3">Coordinator / Router / End device</td></tr>
</table>

6.8.6 PWM frequency control

Table 6-68. PWM frequency control

Syntax	Explanation		
S<reg>=<value>	The command selects PWM operating frequency for particular PWM channel.		
	PWM channel 0 1 2.	**Output pin** GPIO0 GPIO1 GPIO2	**Frequency reg** 143 144 145
	<value> 0 1 2 3 4	**PWM frequency** 5 kHz 10 kHz 20 kHz 50 kHz 100 kHz	
	In fact, PWM frequency selection for any channel affects all channels (frequency is common for all channels). Changing frequency for any PWM channel results in the reset of duty cycle to 0 for all channels		
Result codes	**OK** is returned if value is in valid range, otherwise **ERR0R** is returned.		
S<reg>?	The command reads PWM operating frequency for particular PWM channel coded as above, so it returns 0 to 4.		
Result codes	**OK** is always returned.		
Example	ATS143=2 **OK** ATS144=4 **OK** ATS143? **4** **OK**	Set PWM frequency to 20kHz for PWM channel 0. Set PWM frequency to 100kHz for PWM channel 1. Request for PWM frequency on channel 0. The latest set frequency is returned which has been set recently for channel 1.	
Default value	0 (meaning 5kHz)		
Persistence	value is not stored in the EEPROM.		
Node types	Coordinator / Router / End device		

6.8.7 PWM duty cycle control

Table 6-69. PWM duty cycle control

Syntax	Explanation		
	The command selects duty cycle value for particular PWM channel.		
S<reg>=<value>	**PWM channel** 0 1 2	**Output pin** GPIO0 GPIO1 GPIO2	**Duty cycle reg** 146 147 148

Table 6-69. PWM duty cycle control

	<value> is an integer number in the range of 0 to 100 representing PWM duty cycle in percents. Note: Currently stated duty cycle on the output pin will be changed as soon as current period of PWM frequency.is ended. Note: Resolution of duty cycle setting depends on the PWM frequency, as below:	
	PWM frequency 5 kHz 10 kHz 20 kHz 50 kHz 100 kHz	**Duty cycle resolution** 1% 1% 1% 2,5% 5%
Result codes	**OK** is returned if value is in valid range, otherwise **ERROR** is returned.	
S<reg>?	The command reads duty cycle given for particular PWM channel in percents.	
Result codes	**OK** is always returned.	
Example	ATS146=45 **OK**	Set duty cycle to 45% for PWM channel 0.
Default value	0 (%)	
Persistence	value is not stored in the EEPROM.	
Node types	Coordinator / Router / End device	

6.9 Remote Management

Remote management functions include the password-protected AT-commands that come from originating node to a target node. The received AT-command sequences are executed on the destination node, as if they would come from a serial port. Information response and result code of the command execution are sent back to the originating node in the form as if they are normally returned over serial interface.

Remote execution service is protected by 32-bit password that can be set during the node installation or manufacturing.

Remote management function is an important tool that allows organization of commissioning procedures on PC, using commercial off-the-shelf terminal software.

6.9.1 "+WPASSWORD" - Set a password

Table 6-70. "+WPASSWORD" - Set a password

Syntax	Explanation
+WPASSWORD <psw>	The command sets a new password for remote management command. Password is in form of 32-bit hexadecimal number. Note: This command is not to be confused with the parameter set commands. Unlike those, it does not include the “=” symbol.
Result codes	**OK** is always returned.
Example	AT+WPASSWORD 65432178 **OK**

Table 6-70. "+WPASSWORD" - Set a password

Default value	0
Persistence	psw value is stored in the EEPROM. Note: The password cannot be reloaded with default value through *&F* command (see Table 6-48 on page 6-30) but it can be rewritten over the air using remote AT-command (see Table 6-71 on page 6-47).
Node types	Coordinator / Router / End device

6.9.2 "R"-Remote execution of AT command

Table 6-71. "R"-Remote execution of AT command

Syntax	Explanation	
R<addr>,<psw>, <cmd>	The command lets the execution of AT-commands on a remote node, with output redirected. Password (psw) is a 32-bit hexadecimal number, which is set for this specific node. addr specifies short (network) address of the destination node. cmd is a sequence of AT-commands without AT prefix. Note: It is strongly recommended not to use the &H and %H commands for cmd, as they produce extremely lengthy output.	
Result codes	All the responses and result codes are received from the remote node in text form and thus can be normally processed. If a connection loss will be detected, the **ERROR** result code will be returned after time-out since last response packet is received (approx 3 sec). In particular, remote execution of +WLEAVE command will result in **ERROR** code, despite being executed successfully. If remote command is send to End device with sleeping period longer than time-out, **ERROR** will be returned. If the controlled node is not in the PAN, **ERROR** will be returned. Remote execution is not allowed for commands that cause the receiving node to send data over the network: D, DU, DS, +WPING, R. Attempting will result in **ERROR** code with the command processing aborted.	
Example	ATR1,65432178,+GMM?+WRSSI 2 **+GMM:ZIGBIT** **+WRSSI:-80** **OK** ATR1,65432178,+WLEAVE **ERROR**	Get model number and RSSI Remove node from network – **ERROR** will be returned but delayed.
Node types	Coordinator / Router / End device	

Section 7

User Guide Revision History

7.1 Rev.8021A – 11/09

1. First version of the BitCloud SerialNet User Guide

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature

Guía de Usuario de Arduino

Rafael Enríquez Herrador

13 de noviembre de 2009

I.T.I. Sistemas

Universidad de Córdoba

i52enher@uco.es

Índice general

1. PREFACIO **7**

2. INTRODUCCIÓN **8**
2.1. ¿Qué es ARDUINO? 8
2.2. ¿Por qué ARDUINO? 8

3. HARDWARE **10**
3.1. Placas E/S 10
3.2. Arduino Diecimila 11
3.2.1. Visión General 11
3.2.2. Resumen 12
3.2.3. Alimentación 12
3.2.4. Memoria 13
3.2.5. Entrada y Salida 13
3.2.6. Comunicación 14
3.2.7. Programación 14
3.2.8. Reseteo Automático (Software) 14
3.2.9. Protección de Sobrecarga del USB 15
3.2.10. Características Físicas 15

4. SOFTWARE **16**
4.1. Instalar el Software Arduino 16
4.1.1. Windows 17
4.1.2. MAC OS X (v. 10.3.9 o posterior) 23
4.1.3. GNU/Linux 26
4.2. Introducción al Entorno Arduino 27
4.2.1. Barra de herramientas 27
4.2.2. Menús 29
4.2.3. Preferencias 29

5. COMENZANDO CON ARDUINO **30**
5.1. Estructura 30
5.2. Variables 32
5.3. Tipos de datos 34
5.4. Aritmética 35

5.5. Constantes . . . 36
5.6. Control de flujo . . . 37
5.7. E/S digital . . . 39
5.8. E/S analógica . . . 40
5.9. Tiempo . . . 41
5.10. Matemáticas . . . 42
5.11. Aleatorio . . . 42
5.12. Serie . . . 43

A. Ejemplos de Aplicación con Arduino **45**
A.1. Salida digital . . . 45
A.2. Entrada digital . . . 46
A.3. Salida PWM . . . 46
A.4. Entrada de potenciometro . . . 47

B. Esquemático de Arduino Diecimila **49**

Índice de cuadros

3.1. Características técnicas de Arduino Diecimila 12

5.1. Relación valor-salida con *analogWrite()* 41

Índice de figuras

3.1. Placa Arduino Diecimila (USB) 11

4.1. Descripción de componentes de la placa Arduino Diecimila 17
4.2. Conexión del cable USB a la placa Arduino 18
4.3. Asistente para Nuevo Hardware MS-Windows - Paso 1 18
4.4. Asistente para Nuevo Hardware MS-Windows - Paso 2 19
4.5. Asistente para Nuevo Hardware MS-Windows - Paso 3 19
4.6. Asistente para Nuevo Hardware MS-Windows - Paso 4 20
4.7. Entorno Arduino 21
4.8. Administrador de Dispositivos MS-Windows 21
4.9. Menú de selección de puerto del Entorno Arduino 22
4.10. Menú de selección de placa del Entorno Arduino 22
4.11. Botón de subida de la rutina a la placa 22
4.12. Instalación de drivers en Mac OS-X 23
4.13. Conexión del cable USB a la placa Arduino 24
4.14. Entorno Arduino 25
4.15. Menú de selección de puerto del Entorno Arduino 25
4.16. Menú de selección de placa del Entorno Arduino 26
4.17. Botón de subida de la rutina a la placa 26

A.1. Esquema de salida digital 45
A.2. Esquema de entrada digital 46
A.3. Esquema de salida PWM 46
A.4. Esquema de entrada de potenciometro 47

Capítulo 1

PREFACIO

Esta guía de usuario intenta ser una forma de acercarse al diseño y desarrollo de proyectos basados en Arduino para aquellas personas que nunca han trabajado con él pero que poseen un buen nivel en programación y electrónica. Por esta razón y para hacerlo fácil, se ha excluido mucha información existente en Internet y otros manuales para centrarse en los aspectos más básicos de las características y la programación de Arduino.

Otro de los objetivos de esta guía es organizar un poco la gran cantidad de información sobre este tema existente en la red. Para ello casi toda la información se ha obtenido a través de la fuente `http://www.arduino.cc` o de manuales basados en ella pero algo más estructurados. En general, el texto es una traducción libre al español del documento original *«Arduino Programming Notebook»* escrito y compilado por «Brian W. Evans».

Por último, la guía está pensada para aquellas personas que no han usado Arduino pero les gustaría iniciarse en este campo, por lo que si eres un usuario avanzado de esta plataforma no te aportará nada nuevo (sólo te servirá para repasar conceptos básicos).

Espero que les sea de utilidad.

Capítulo 2

INTRODUCCIÓN

2.1. ¿Qué es ARDUINO?

Arduino es una plataforma de prototipos electrónica de código abierto (open-source) basada en hardware y software flexibles y fáciles de usar. Está pensado para artistas, diseñadores, como hobby y para cualquiera interesado en crear objetos o entornos interactivos.

Arduino puede «sentir» el entorno mediante la recepción de entradas desde una variedad de sensores y puede afectar a su alrededor mediante el control de luces, motores y otros artefactos. El microcontrolador de la placa se programa usando el «Arduino Programming Language» (basado en Wiring[1]) y el «Arduino Development Environment» (basado en Processing[2]). Los proyectos de Arduino pueden ser autonomos o se pueden comunicar con software en ejecución en un ordenador (por ejemplo con Flash, Processing, MaxMSP, etc.).

Las placas se pueden ensamblar a mano[3] o encargarlas preensambladas[4]; el software se puede descargar[5] gratuitamente. Los diseños de referencia del hardware (archivos CAD) están disponibles bajo licencia open-source, por lo que eres libre de adaptarlas a tus necesidades.

Arduino recibió una mención honorífica en la sección Digital Communities del Ars Electronica Prix en 2006.

2.2. ¿Por qué ARDUINO?

Hay muchos otros microcontroladores y plataformas microcontroladoras disponibles para computación física. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, y muchas otras ofertas de funcionalidad similar. Todas estas herramientas toman los desordenados detalles de la programación de microcontrolador y la encierran en un paquete fácil de usar. Arduino también simplifica el proceso de trabajo con microcontroladores, pero ofrece algunas ventajas para profesores, estudiantes y aficionados interesados sobre otros sistemas:

[1]Más información en `http://wiring.org.co`
[2]Más información en `http://www.processing.org`
[3]Más información en `http://www.arduino.cc/en/Main/USBAssembly`
[4]Más información en `http://www.arduino.cc/en/Main/Buy`
[5]Más información en `http://www.arduino.cc/en/Main/Software`

- Barato: Las placas Arduino son relativamente baratas comparadas con otras plataformas microcontroladoras. La versión menos cara del modulo Arduino puede ser ensamblada a mano, e incluso los módulos de Arduino preensamblados cuestan menos de 50$.
- Multiplataforma: El software de Arduino se ejecuta en sistemas operativos Windows, Macintosh OSX y GNU/Linux. La mayoría de los sistemas microcontroladores están limitados a Windows.
- Entorno de programación simple y claro: El entorno de programación de Arduino es fácil de usar para principiantes, pero suficientemente flexible para que usuarios avanzados puedan aprovecharlo también. Para profesores, está convenientemente basado en el entorno de programación Processing, de manera que estudiantes aprendiendo a programar en ese entorno estarán familiarizados con el aspecto y la imagen de Arduino.
- Código abierto y software extensible: El software Arduino está publicado como herramientas de código abierto, disponible para extensión por programadores experimentados. El lenguaje puede ser expandido mediante librerias C++, y la gente que quiera entender los detalles técnicos pueden hacer el salto desde Arduino a la programación en lenguaje AVR C en el cual está basado. De forma similar, puedes añadir código AVR-C directamente en tus programas Arduino si quieres.
- Código abierto y hardware extensible: El Arduino está basado en microcontroladores ATMEGA8 y ATMEGA168 de Atmel. Los planos para los módulos están publicados bajo licencia Creative Commons, por lo que diseñadores experimentados de circuitos pueden hacer su propia versión del módulo, extendiéndolo y mejorándolo. Incluso usuarios relativamente inexpertos pueden construir la versión de la placa del módulo para entender como funciona y ahorrar dinero.

Capítulo 3

HARDWARE

Hay múltiples versiones de la placa Arduino. La mayoría usan el ATmega168 de Atmel, mientras que las placas más antiguas usan el ATmega8.

Nota: Los diseños de referencia para Arduino se distribuyen bajo licencia Creative Commons Attribution-ShareAlike 2.5.

3.1. Placas E/S

- **Diecimila:** Esta es la placa Arduino más popular. Se conecta al ordenador con un cable estándar USB y contiene todo lo que necesitas para programar y usar la placa. Puede ser ampliada con variedad de dispositivos: placas hijas con características específicas.

- **Nano:** Una placa compacta diseñada para uso como tabla de pruebas, el Nano se conecta al ordenador usando un cable USB Mini-B.

- **Bluetooth:** El Arduino BT contiene un modulo bluetooth que permite comunicación y programación sin cables. Es compatible con los dispositivos Arduino.

- **LilyPad:** Diseñada para «aplicaciones listas para llevar», esta placa puede ser conectada en fábrica, y un estilo sublime.

- **Mini:** Esta es la placa más pequeña de Arduino. Trabaja bien en tabla de pruebas o para aplicaciones en las que prima el espacio. Se conecta al ordenador usando el cable Mini USB.

- **Serial:** Es una placa básica que usa RS232 como un interfaz con el ordenador para programación y comunicación. Esta placa es fácil de ensamblar incluso como ejercicio de aprendizaje.

- **Serial Single Sided:** Esta placa está diseñada para ser grabada y ensamblada a mano. Es ligeramente más grande que la Diecimila, pero aun compatible con los dispositivos.

3.2. Arduino Diecimila

3.2.1. Visión General

El Arduino Diecimila es una placa microcontroladora basada en el ATmega168. Tiene 14 pines de entrada/salida digital (de los cuales 6 pueden ser usados como salidas PWM), 6 entradas analógicas, un oscilador de cuarzo a 16MHz, una conexión USB, un conector para alimentación, una cabecera ICSP, y un botón de reset. Contiene todo lo necesario para soportar el microcontrolador; simplemente conéctalo a un ordenador con un cable USB o enchúfalo con un adaptador AC/DC o batería para comenzar.

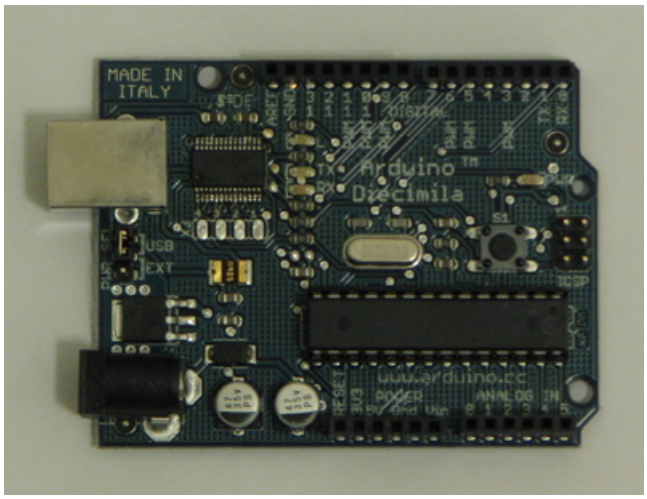


Figura 3.1: Placa Arduino Diecimila (USB)

«Diecimila» quiere decir 10000 en italiano y fue llamado así para resaltar el hecho de que más de 10000 placas Arduino han sido fabricadas. El Diecimila es el último en la serie de placas USB Arduino.

3.2.2. Resumen

Característica	Descripción
Microcontrolador	ATmega168
Voltaje de operación	5 V
Tensión de entrada (recomendada)	7 - 12 V
Tensión de entrada (límite)	6 - 20 V
Pines digitales de E/S	14 (de los cuales 6 proveen salidas PWM)
Pines de entrada analógicos	6
Corriente DC por pin E/S	40 mA
Corriente DC para pin 3.3 V	50 mA
Memoria Flash	16 KB (de los cuales 2 KB usados para bootloader)
SRAM	1 KB
EEPROM	512 bytes
Frecuencia de reloj	16 MHz

Cuadro 3.1: Características técnicas de Arduino Diecimila

3.2.3. Alimentación

El Arduino Diecimila puede ser alimentado a través de la conexión USB o con un suministro de energía externo. La fuente de energía se selecciona mediante el jumper PWR_SEL: para alimentar a la placa desde la conexión USB, colocarlo en los dos pines más cercanos al conector USB, para un suministro de energía externo, en los dos pines más cercanos al conector de alimentación externa.

La alimentación externa (no USB) puede venir o desde un adaptador AC-a-DC (wall-wart) o desde una batería. El adaptador puede ser conectado mediante un enchufe centro-positivo en el conector de alimentación de la placa. Los cables de la batería pueden insertarse en las cabeceras de los pines Gnd y Vin del conector POWER. Un regulador de bajo abandono proporciona eficiencia energética mejorada.

La placa puede operar con un suministro externo de 6 a 20 voltios. Si es suministrada con menos de 7 V, sin embargo, el pin de 5 V puede suministrar menos de cinco voltios y la placa podría ser inestable. Si usa más de 12 V, el regulador de tensión puede sobrecalentarse y dañar la placa. El rango recomendado es de 7 a 12 voltios.

Los pines de alimentación son los siguientes:

- **VIN**. La entrada de tensión a la placa Arduino cuando está usando una fuente de alimentación externa (al contrario de los 5 voltios de la conexión USB u otra fuente de alimentación regulada). Puedes suministrar tensión a través de este pin, o, si suministra tensión a través del conector de alimentación, acceder a él a través de este pin.

- **5V.** El suministro regulado de energía usado para alimentar al microcontrolador y otros componentes de la placa. Este puede venir o desde VIN a través de un regulador en la placa, o ser suministrado por USB u otro suministro regulado de 5 V.

- **3V3.** Un suministro de 3.3 V generado por el chip FTDI de la placa. La corriente máxima es de 50 mA.

- **GND.** Pines de Tierra.

3.2.4. Memoria

El ATmega168 tiene 16 KB de memoria Flash para almacenar código (de los cuales 2 KB se usa para el «bootloader»). Tiene 1 KB de SRAM y 512 bytes de EEPROM (que puede ser leida y escrita con la librería EEPROM[1]).

3.2.5. Entrada y Salida

Cada uno de los 14 pines digitales del Diecimila puede ser usado como entrada o salida, usando funciones *pinMode()*, *digitalWrite()* y *digitalRead()*[2]. Operan a 5 voltios. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia interna «pull-up» (desconectada por defecto) de 20-50 KOhms. Además, algunos pines tienen funciones especiales:

- **Serial: 0 (Rx) y 1 (Tx).** Usados para recibir (Rx) y transmitir (Tx) datos TTL en serie. Estos pines estan conectados a los pines correspondientes del chip FTDI USB-a-TTL Serie.
- **Interruptores externos: 2 y 3.** Estos pines pueden ser configurados para disparar un interruptor en un valor bajo, un margen creciente o decreciente, o un cambio de valor. Mirar la función *attachInterrupt()*[3].
- **PWM: 3, 5, 6, 9, 10 y 11.** Proporcionan salida PWM de 8 bits con la función *analogWrite()*[4].
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** Estos pines soportan comunicación SPI, la cual, aunque proporcionada por el hardware subyacente, no está actualmente incluida en el lenguaje Arduino.
- **LED: 13.** Hay un LED empotrado conectado al pin digital 13. Cuando el pin está a valor HIGH, el LED está encendido, cuando el pin está a LOW, está apagado.

El Diecimila tiene 6 entradas analógicas, cada una de las cuales proporciona 10 bits de resolución (por ejemplo 1024 valores diferentes). Por defecto miden 5 voltios desde tierra, aunque es posible cambiar el valor más alto de su rango usando el pin ARF y algún código de bajo nivel. Además, algunos pines tienen funcionalidad especializada:

- **I²C: 4 (SDA) y 5 (SCL).** Soportan comunicación I²C (TWI) usando la *libreria Wire*[5].

Hay otro par de pines en la placa:

- **AREF.** Voltaje de referencia para las entradas analógicas. Usado con *analogReference()*[6].
- **Reset.** Pone esta linea a LOW para resetear el microcontrolador. Típicamente usada para añadir un botón de reset a dispositivos que bloquean a la placa principal.

[1]Más información en: `http://www.arduino.cc/en/Reference/EEPROM`
[2]Más información en: `http://www.arduino.cc/en/Reference/`
[3]Más información en: `http://www.arduino.cc/en/Reference`
[4]Más información en: `http://www.arduino.cc/en/Reference`
[5]Más información en: `http://wiring.org.co/reference/libraries/Wire/index.html`
[6]Más información en: `http://www.arduino.cc/en/Reference`

3.2.6. Comunicación

El Arduino Diecimila tiene un numero de infraestructuras para comunicarse con un ordenador, otro Arduino, u otros microcontroladores. El ATmega168 provee comunicación serie UART TTL (5 V), la cual está disponible en los pines digitales 0 (Rx) y 1 (Tx). Un FTDI FT232RL en la placa canaliza esta comunicación serie al USB y los drivers FTDI (incluidos con el software Arduino) proporcionan un puerto de comunicación virtual al software del ordenador. El software Arduino incluye un monitor serie que permite a datos de texto simple ser enviados a y desde la placa Arduino.

Una libreria *SoftwareSerial*[7] permite comunicación serie en cualquiera de los pines digitales del Diecimila.

El ATmega168 también soporta comunicación 12C (TWI) y SPI. El software Arduino incluye una librería Wire para simplificar el uso del bus 12C[8]. Para usar la comunicación SPI, consultar el esquema del ATmega168.

3.2.7. Programación

El Arduino Diecimila puede ser programado con el software Arduino[9].

El ATmega168 del Arduino Diecimila viene con un *bootloader*[10] pregrabado que te permite subirle nuevo código sin usar un programador hardware externo. Se comunica usando el protocolo original STK500.

También puedes saltar el *bootloader* y programar el ATmega168 a través de la cabecera ICSP (In-Circuit Serial Programming)[11].

3.2.8. Reseteo Automático (Software)

En lugar de requerir una pulsación física del botón de reset antes de una subida, el Arduino Diecimila esta diseñado de forma que permite ser reseteado por software en ejecución en una computadora conectada. Una de las lineas de control de flujo de hardware (DTR) del FT232RL esta conectada a la linea de reset del ATmega168 a través de un condensador de 100 nF. Cuando esta linea toma el valor LOW, la linea reset se mantiene el tiempo suficiente para resetear el chip. La version 0009 del software Arduino usa esta capacidad para permitirte cargar código simplemente presionando el botón *upload* en el entorno Arduino. Esto significa que el *bootloader* puede tener un tiempo de espera más corto, mientras la bajada del DTR puede ser coordinada correctamente con el comienzo de la subida.

Esta configuración tiene otras repercusiones. Cuando el Diecimila esta conectado a un ordenador que ejecuta o Mac OS X o Linux, se resetea cada vez que se hace una conexión a él por software (a través de USB). Durante el siguiente medio segundo aproximadamente, el *bootloader* se ejecutará en el Diecimila. Mientras esté programado para ignorar datos «malformados» (por ejemplo, cualquiera excepto una subida de código nuevo), interceptará los primeros bytes de datos enviados a la placa despues de abrir la conexión. Si una rutina que se ejecuta en la placa recibe una configuración una

[7] Más información en: `http://www.arduino.cc/en/Reference/SoftwareSerial`

[8] Para más detalles visitar: `http://wiring.org.co/reference/libraries/Wire/index.html`

[9] Descargar desde: `http://www.arduino.cc/en/Main/Software`

[10] Más información en: `http://www.arduino.cc/en/Tutorial/Bootloader`

[11] Más detalles en:`http://www.arduino.cc/en/Hacking/Programmer`

vez u otros datos cuando empieza, asegurarse de que el software con el que se comunica espera un segundo después de abrir la conexión y antes de enviar estos datos.

3.2.9. Protección de Sobrecarga del USB

El Arduino Diecimila tiene un fusible reseteable que protege tus puertos USB del ordenador de cortes y sobrecargas. Aunque la mayoría de los ordenadores proporcionan su propia protección interna, el fusible proporciona una capa de protección extra. Si más de 500 mA se aplican al puerto USB, el fusible automáticamente romperá la conexión hasta que el corte o la sobrecarga sean eliminados.

3.2.10. Características Físicas

La máxima longitud y anchura del Diecimila PCB son 2.7 y 2.1 pulgadas respectivamente, con el conector USB y el conector de alimentación que se extienden más alla de las primeras dimensiones. Tres agujeros de tornillo permiten a la placa atornillarse a una superficie o caja.

Capítulo 4

SOFTWARE

4.1. Instalar el Software Arduino

Esta sección explica como instalar el software Arduino en un ordenador que ejecute cualquiera de los siguientes Sistemas Operativos: Windows, Mac OS X, GNU/Linux.

Este documento explica como conectar tu placa Arduino al ordenador y cargar tu primera rutina.

Estos son los pasos que seguiremos:

1. Obtener una placa Arduino y un cable.
2. Descargar el entorno Arduino.
3. Instalar los drivers USB.
4. Conectar la placa.
5. Conectar un LED.
6. Ejecutar el entorno Arduino.
7. Subir un programa.
8. Buscar el Led que parpadea.

1.Obtener una placa Arduino y un cable

En este tutorial se asume que estas usando un Arduino Diecimila. Si tienes otra placa, lee la información correspondiente en `http://www.arduino.cc/en/Guide/HomePage`. El Arduino Diecimila es una placa simple que contiene todo lo que necesitas para empezar a trabajar con electrónica y programación de microcontrolador.

También necesitas un cable USB estándar (del tipo que conectarías a una impresora USB, por ejemplo).

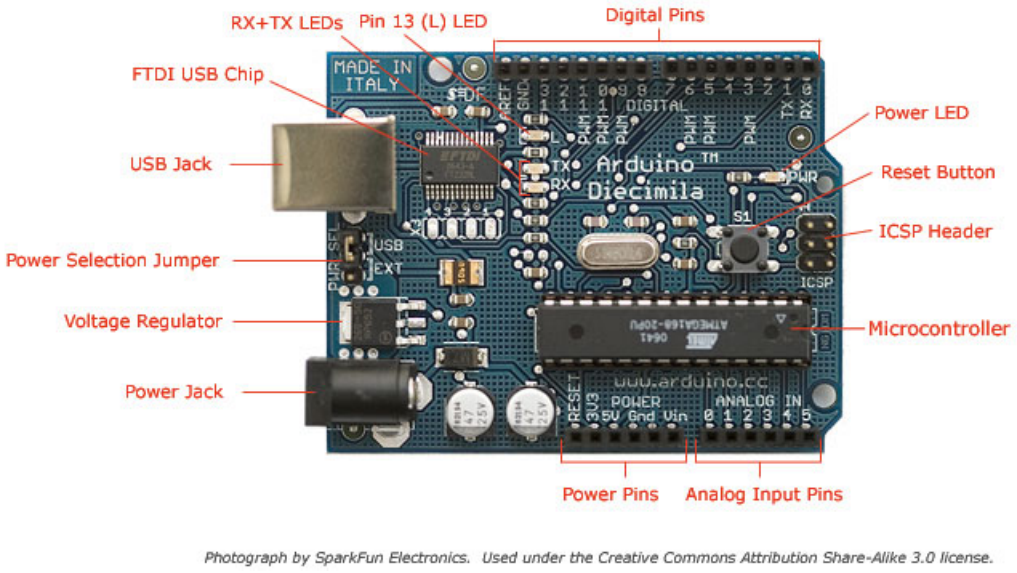


Figura 4.1: Descripción de componentes de la placa Arduino Diecimila

4.1.1. Windows

2.Descargar el entorno Arduino.

Para programar la placa Arduino necesitas el entorno Arduino.

Descarga la última versión desde `http://www.arduino.cc/en/Main/Software`.

Cuando termine la descarga, descomprime el archivo descargado. Asegurate de conservar la estructura de carpetas. Haz doble click en la carpeta para abrirla. Debería haber archivos y subcarpetas en su interior.

3.Instalar los drivers USB.

Si estas usando un Arduino USB, necesitarás instalar los drivers para el chip FTDI de la placa. Estos pueden encontrarse en el directorio *drivers/FTDI USB Drivers* de la distribución Arduino. En el siguiente paso («Conectar la placa»), se mostrará el asistente para *Añadir Nuevo Hardware de Windows* para estos drivers.

La última versión de los drivers se puede encontrar en `http://www.ftdichip.com/Drivers/VCP.htm`.

4.Conectar la placa.

La fuente de alimentación se selecciona mediante el *jumper* entre los conectores del USB y alimentación. Para alimentar la placa desde el puerto USB (bueno para controlar dispositivos de baja potencia como LEDs), coloca el *jumper* en los dos pines más cercanos al conector USB. Para alimentar la placa desde una fuente externa (6-12 V), coloca el *jumper* en los dos pines más cercanos al conector de alimentación. En cualquier caso, conecta la placa a un puerto USB de tu ordenador.

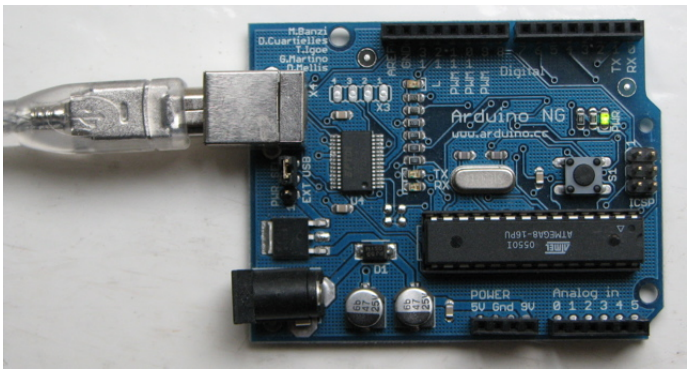


Figura 4.2: Conexión del cable USB a la placa Arduino

El LED de alimentación debería encenderse.

El asistente para *Añadir Nuevo Hardware* debería abrirse. Indícale que no conecte a *Windows Update* y haz click en siguiente.

Figura 4.3: Asistente para Nuevo Hardware MS-Windows - Paso 1

Selecciona «Instalar desde una lista o ubicación especificada (Avanzado)» y haz click en siguiente.

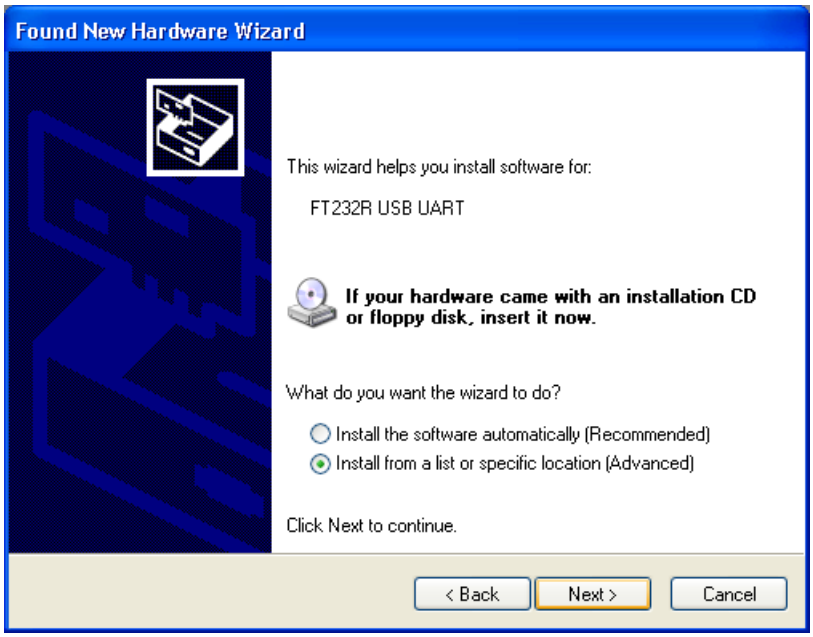


Figura 4.4: Asistente para Nuevo Hardware MS-Windows - Paso 2

Asegurate que «Buscar el mejor driver en estas ubicaciones» está marcado; desmarca «Buscar dispositivos extraibles»; marca «Incluir esta ubicación en la búsqueda» y navega a la ubicación donde descomprimiste los drivers USB en el paso anterior. Haz click en siguiente.

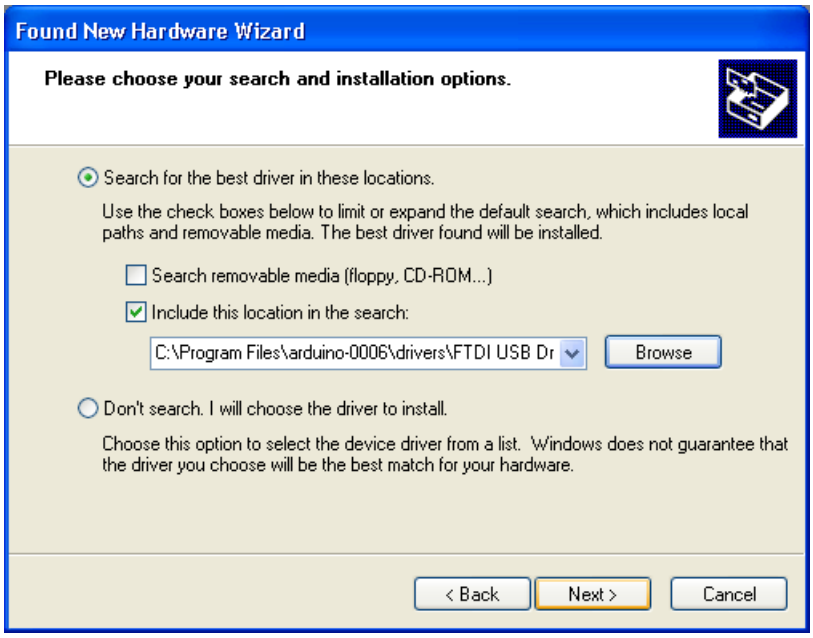


Figura 4.5: Asistente para Nuevo Hardware MS-Windows - Paso 3

El asistente buscará el driver y dirá que se encontró «USB Serial Converter». Haz click en finalizar.

Figura 4.6: Asistente para Nuevo Hardware MS-Windows - Paso 4

El asistente para *Añadir Nuevo Hardware* aparecerá de nuevo. Realiza los mismos pasos. Esta vez, se encontrará un «USB Serial Port».

5. Conectar un LED (si estas usando una placa antigua).

La primera rutina que subirás a la placa Arduino hace parpadear un LED. El Arduino Diecimila (y el Arduino NG original) tiene una resistencia incorporada y un LED en el pin 13. En el Arduino NG Rev. C y placas Arduino pre-NG, sin embargo, el pin 13 no tiene un LED incorporado. En estas placas, necesitarás conectar la patilla positiva (más larga) de un LED al pin 13 y la negativa (más corta) a tierra (marcada como «GND»). Normalmente, también necesitaras usar una resistencia con el LED, pero estas placas tienen una resistencia integrada en el pin 13.

6.Ejecutar el entorno Arduino.

Abrir la carpeta de Arduino y hacer doble click en la aplicación Arduino.

7.Subir un programa.

Abrir la rutina de ejemplo de parpadeo del LED: *File* > *Sketchbook* > *Examples* > *Digital* > *Blink*.

Seleccionar el dispositivo serie de la placa Arduino desde el menu *Herramientas* > *Puerto Serie*. En Windows, este debería ser *COM1* o *COM2* para la placa serie Arduino, o *COM3, COM4* o *COM5* para la placa USB. Para descubrirlo, abrir el *Administrador de Dispositivos de Windows* (En la pestaña *Hardware* o en el *Panel de Control de Sistema).* Buscar un «USB Serial Port» en la sección *Puertos;* esa es la placa Arduino.

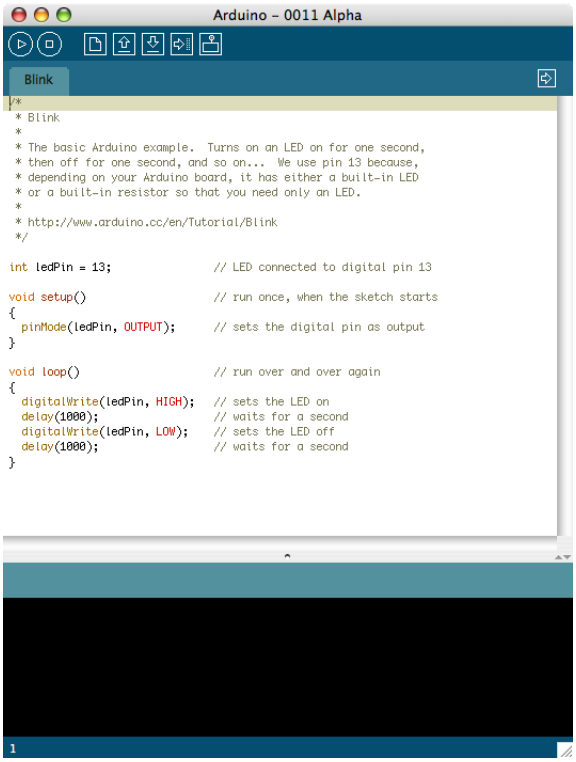


Figura 4.7: Entorno Arduino

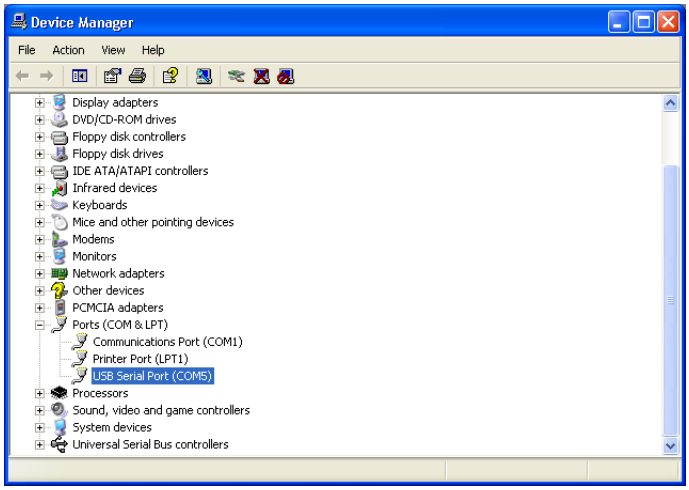


Figura 4.8: Administrador de Dispositivos MS-Windows

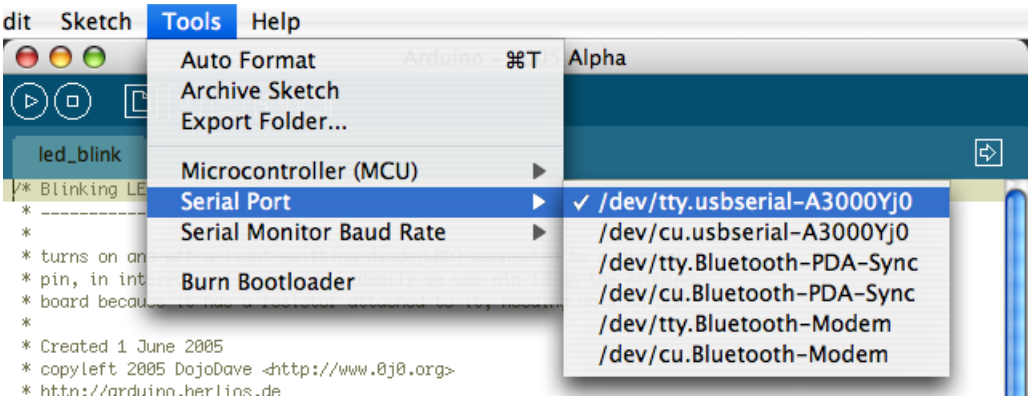


Figura 4.9: Menú de selección de puerto del Entorno Arduino

Asegurarse de que «Arduino Diecimila» está seleccionada en el menú *Tools* > *Board*.

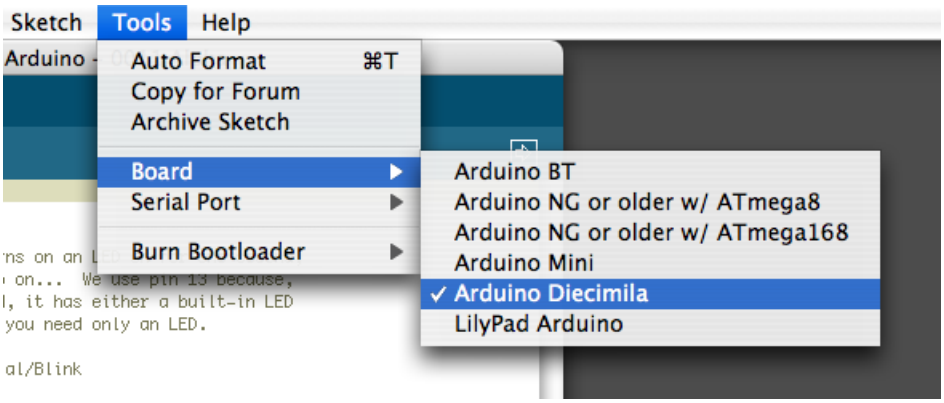


Figura 4.10: Menú de selección de placa del Entorno Arduino

Ahora, simplemente haz click en el botón «Upload» del entorno. Espera unos pocos segundos (deberías ver los LEDs Rx y Tx de la placa iluminándose). Si la carga es correcta, el mensaje «Done uploading» aparacerá en la barra de estado.

Figura 4.11: Botón de subida de la rutina a la placa

8.Buscar el LED que parpadea.

Unos pocos segundos después de que la subida termine, deberías ver el LED ámbar (amarillo) en la placa empezar a parpadear. Si lo hace ¡enhorabuena! Has conseguido Arduino cargado y ejecutándose.

Si tienes problemas, consulta `http://www.arduino.cc/en/Guide/Troubleshooting`.

4.1.2. MAC OS X (v. 10.3.9 o posterior)

2.Descargar el entorno Arduino.

Para programar la placa Arduino necesitas el entorno Arduino.

Descarga la última versión desde `http://www.arduino.cc/en/Main/Software`.

Cuando termine la descarga, descomprime el archivo descargado. Asegurate de conservar la estructura de carpetas. Haz doble click en la carpeta para abrirla. Debería haber archivos y subcarpetas en su interior.

3.Instalar los drivers USB.

Si estas usando un Arduino USB, necesitarás instalar los drivers para el chip FTDI de la placa. Estos pueden encontrarse en el directorio *drivers* de la distribución Arduino.

Si tienes un Mac más antiguo como un Powerbook, iBook, G4 o G5, deberías usar los drivers PPC: *FTDIUSBSerialDriver_ v2_ 1_ 9.dmg*. Si tienes un Mac más nuevo como un MacBook, MacBook Pro o Mac Pro, necesitas los drivers de Intel: *FTDIUSBSerialDriver_ v2_ 2_ 9_ Intel.dmg*. Haz doble click para montar la imagen del disco y ejecutar el *FTDIUSBSerialDriver.pkg* incluido.

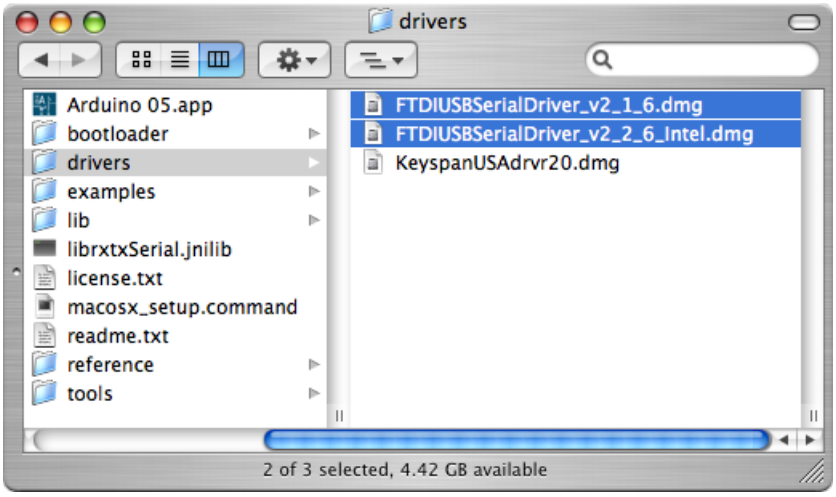


Figura 4.12: Instalación de drivers en Mac OS-X

La última versión de los drivers se puede encontrar en `http://www.ftdichip.com/Drivers/VCP.htm`.

4.Conectar la placa.

La fuente de alimentación se selecciona mediante el *jumper* entre los conectores del USB y alimentación. Para alimentar la placa desde el puerto USB (bueno para controlar dispositivos de baja potencia como LEDs), coloca el *jumper* en los dos pines más cercanos al conector USB. Para alimentar la placa desde una fuente externa (6-12 V), coloca el *jumper* en los dos pines más cercanos al conector de alimentación. En cualquier caso, conecta la placa a un puerto USB de tu ordenador.

El LED de alimentación debería encenderse.

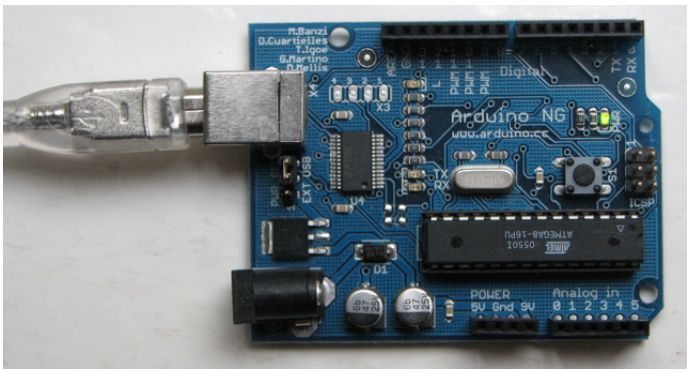


Figura 4.13: Conexión del cable USB a la placa Arduino

5. Conectar un LED (si estas usando una placa antigua).

La primera rutina que subirás a la placa Arduino hace parpadear un LED. El Arduino Diecimila (y el Arduino NG original) tiene una resistencia incorporada y un LED en el pin 13. En el Arduino NG Rev. C y placas Arduino pre-NG, sin embargo, el pin 13 no tiene un LED incorporado. En estas placas, necesitarás conectar la patilla positiva (más larga) de un LED al pin 13 y la negativa (más corta) a tierra (marcada como «GND»). Normalmente, también necesitaras usar una resistencia con el LED, pero estas placas tienen una resistencia integrada en el pin 13.

6.Ejecutar el entorno Arduino.

Abrir la carpeta de Arduino y hacer doble click en la aplicación Arduino.

7.Subir un programa.

Abrir la rutina de ejemplo de parpadeo del LED: *File* > *Sketchbook* > *Examples* > *Digital* > *Blink*.

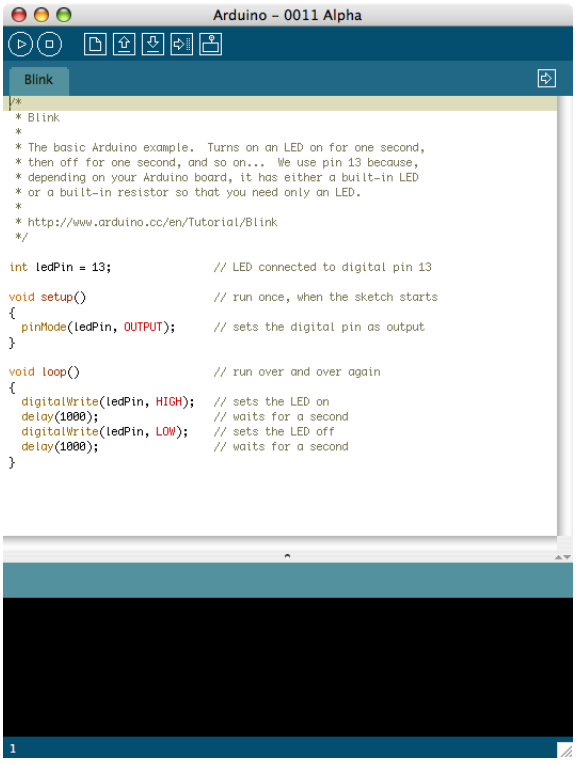


Figura 4.14: Entorno Arduino

Selecciona el dispositivo de la placa Arduino desde el menú *Tools* > *Serial Port*. En el Mac, debería ser algo con */dev/tty.usbserial*.

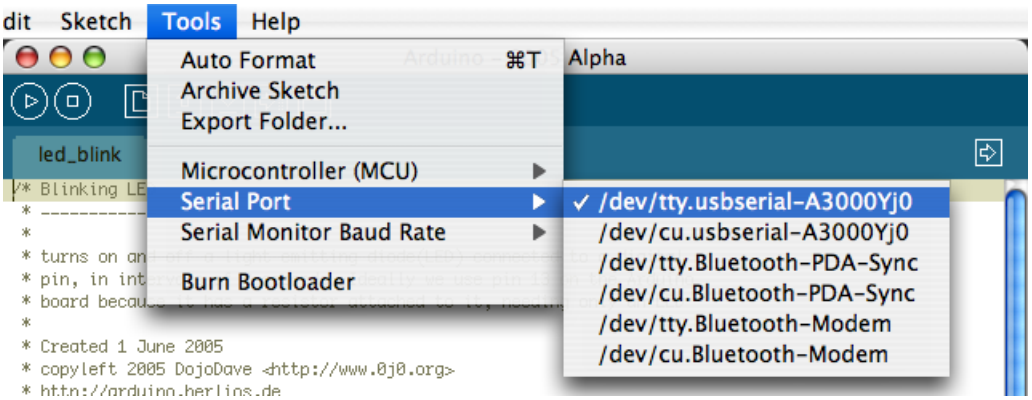


Figura 4.15: Menú de selección de puerto del Entorno Arduino

Asegurate de que «Arduino Diecimila» está seleccionado en el menú *Tools* > *Board*.

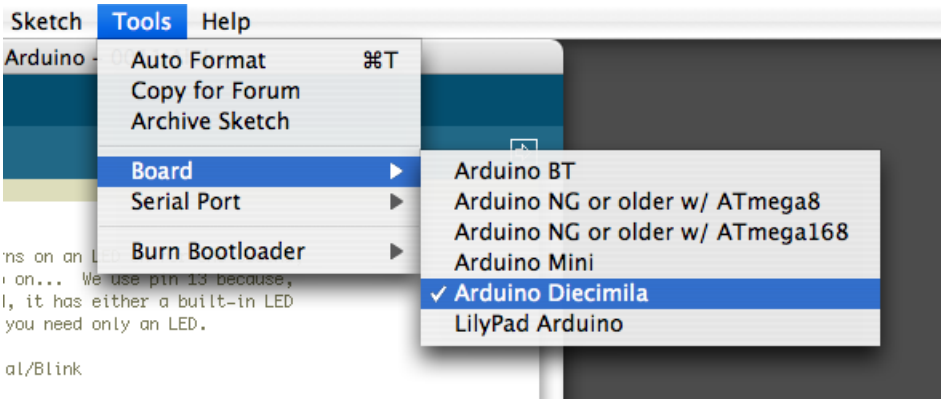


Figura 4.16: Menú de selección de placa del Entorno Arduino

Ahora simplemente haz click en el botón «Upload» en el entorno. Espera unos pocos segundos (deberías ver los LEDs Rx y Tx en la placa iluminándose). Si la subida es correcta, el mensaje «Done uploading» aparecerá en la barra de estado.

Figura 4.17: Botón de subida de la rutina a la placa

8.Buscar el LED que parpadea.

Unos pocos segundos después de que la subida termine, deberías ver el LED ámbar (amarillo) en la placa empezar a parpadear. Si lo hace ¡enhorabuena! Has conseguido Arduino cargado y ejecutándose.

Si tienes problemas, consulta `http://www.arduino.cc/en/Guide/Troubleshooting`.

4.1.3. GNU/Linux

Estas instrucciones se centran en la distribución Ubuntu[1], para más información sobre cómo instalar el entorno Arduino en otras distribuciones visitar `http://www.arduino.cc/playground/Learning/Linux`.

1. Ejecutar el *Gestor de Paquetes Synaptic* (en *Sistema* > *Administración*).

2. Primero necesitas habilitar los repositorios «Universe» y «Multiverse» para que puedas acceder a todos los paquetes que necesitas.

   - Ir a *Configuración* > *Repositorios*.
   - Haz click en *Añadir*.

[1]Más información sobre la distribución en: `http://www.ubuntu.com/`

- Marca «Software restringido por copyright o cuestiones legales (multiverse)» y «Software libre mantenido por la comunidad (universe)» y haz click en *Añadir.*
- Haz click en *Cerrar* y click en *Cerrar* en el diálogo «Los repositorios han cambiado».

3. Haz click en el botón *Recargar* de la barra de herramientas.
4. Marca para instalar: «sun-java5-jre», «gcc-avr», «avr-libc».
5. Haz click en *Aplicar* en la barra de herramientas.
6. Haz click en *Aplicar* en el cuadro de diálogo. Esto instalará los paquetes seleccionados.
7. Acepta la licencia de Java.
8. Espera hasta completar la instalación: el cuadro de diálogo dirá «Cambios aplicados». Haz click en *Cerrar.*
9. Cierra *Synaptic.*
10. Descarga la distribición de GNU/Linux de Arduino desde `http://www.arduino.cc/en/Main/Software`. Haz doble click en el archivo *.zip* y arrastra la carpeta que contiene a algún lugar (por ejempl el Escritorio).
11. Ejecuta el *Terminal* (en *Aplicaciones > Accesorios).*
12. Escribe «sudo update-alternatives –config java» y presiona *Enter.* Teclea el número de opción que tiene «java-1.5.0-sun» en él y presiona *Enter.* Esto hará de la versión de Java de Sun la predeterminada de tu sistema (necesaria porque la versión GNU todavía no soporta todo lo necesitado por el entorno Arduino).
13. Haz doble click en «arduino» en el directorio de aplicación de Arduino. Esto debería lanzar un diálogo preguntando dónde guardas los archivos de tus rutinas de Arduino. Un directorio «Arduino» en tu carpeta *home* es la ubicación típica. Haz click en *OK.* El entorno Arduino debería abrirse.

4.2. Introducción al Entorno Arduino

4.2.1. Barra de herramientas

Verify/Compile

Chequea el código en busca de errores.

Stop

Para el «Serial monitor», o minimiza otros botones.

New

Crea una nueva rutina.

Open

Muestra un menú con todas las rutinas de tu «sketchbook».

Save

Guarda tus rutinas.

Upload to I/O board

Carga tu código a la placa Arduino I/O. Asegúrate de guardar o verificar tu rutina antes de cargarla.

Serial Monitor

Muestra datos serie enviados desde la placa Arduino (placa serie o USB). Para enviar datos a la placa, introduce el texto y haz click en el botón «Send» o presiona «Enter». Elige la velocidad de transmisión de datos desde el desplegable que asigna la velocidad pasada al **Serial.being** en tu rutina. Recuerda que en Mac o GNU/Linux, la placa Arduino se reiniciará (vuelve a ejecutar tu rutina desde del principio) cuando conectes con el «Serial monitor».

Puedes comunicarte también con la placa desde *Processing*, *Flash*, *MaxMSP,* etc (consulta `http://www.arduino.cc/playground/Main/Interfacing` para más detalles).

Tab Menu

Permite gestionar las rutinas con más de un archivo (cada uno de los cuales aparece en su propia pestaña). Estos pueden ser:

- Archivos de código de Arduino (sin extensión).
- Archivos de C (extensión *.c*).
- Archivos de C++ (extensión *.cpp*).
- Archivos de cabecera (extensión *.h*).

4.2.2. Menús

Sketch

- *Verify/Compile:* Comprueba tu rutina para errores.
- *Import Library:* Utiliza una librería en tu rutina. Trabaja añadiendo **#include** en la cima de tu código. Esto añade funcionalidad extra a tu rutina, pero incrementa su tamaño. Para parar de usar una librería, elimina el **#include** apropiado de la cima de tu rutina.
- *Show Sketch Folder:* Abre la carpeta de rutinas en tu escritorio.
- *Add File... :* Añade otro fichero fuente a la rutina. El nuevo archivo aparece en una nueva pestaña en la ventana de la rutina. Esto facilita y agranda proyectos con múltiples archivos fuente. Los archivos pueden ser eliminados de una rutina usando el *Tab Menu.*

Tools

- *Auto Format:* Esto formatea tu código amigablemente.
- *Copy for Discourse:* Copia el código de tu rutina al portapapeles de forma conveniente para postear en un foro, completa con resaltado de sintaxis.
- *Board:* Selecciona la placa que estas usando. Esto controla la forma en que tu rutina es compilada y cargada así como el comportamiento de los elementos del menú *Burn Bootloader.*
- *Serial Port:* Este menú contiene todos los dispositivos serie (reales o virtuales) de tu máquina. Debería actualizarse automáticamente cada vez que abres el nivel superior del menú *Tools.* Antes de subir tu rutina, necesitas seleccionar el elemento de este menú que representa a tu placa Arduino. En el Mac, esto es probablemente algo como **/dev/tty.usbserial-1B1** (para la placa USB), o **/dev/tty.USA19QW1b1P1.1** (para una placa Serie conectada con un adaptador USB-a-Serie Keyspan). En Windows, es probablemente **COM1** o **COM2** (para una placa Serie) o **COM4**, **COM5**, **COM7** o superior (para una placa USB) - para descubrirlo, busca *USB serial device* en la sección puertos del «Gestor de dispositivos de Windows».
- *Burn Bootloader:* Los elementos en este menú te permiten grabar un **bootloader** en tu placa con una variedad de programadores. Esto no es necesario para uso normal de una placa Arduino, pero puede ser útil si encargas ATmegas adicionales o estás construyendo una placa por tu cuenta. Asegurate que has seleccionado la placa correcta del menú *Boards* de antemano. Para grabar un **bootloader** con el AVR ISP, necesitas seleccionar el elemento que corresponde a tu programador del menú *Serial Port.*

4.2.3. Preferencias

Algunas preferencias pueden ser ajustadas en el diálogo *Preferences* (se encuentra bajo el menú *Arduino* en el Mac, o *File* en Windows y GNU/Linux). El resto se puede encontrar en los archivos de preferencias.

Capítulo 5

COMENZANDO CON ARDUINO

5.1. Estructura

La estructura básica del lenguaje de programación Arduino es bastante simple y se organiza en al menos dos partes o funciones que encierran bloques de declaraciones.

```
void setup()
{
  statements;
}

void loop()
{
  statements;
}
```

Ambas funciones son requeridas para que el programa funcione.

setup()

La función *setup* debería contener la declaración de cualquier variable al comienzo del programa. Es la primera función a ejecutar en el programa, es ejecutada una vez y es usada para asignar *pinMode* o inicializar las comunicaciones serie.

```
void setup()
{
  pinMode(pin, OUTPUT); //ajusta 'pin' como salida
}
```

loop()

La función *loop* se ejecuta a continuación e incluye el código que se ejecuta contínuamente - leyendo entradas, activando salidas, etc. Esta función es el núcleo de todos los programas Arduino y hace la mayor parte del trabajo.

```
void loop()
{
  digitalWrite(pin, HIGH); //Activa 'pin'
  delay(1000);             //espera un segundo
  digitalWrite(pin, LOW);  //Desactiva 'pin'
  delay(1000);             //espera un segundo
}
```

funciones

Una función es un bloque de código que tiene un nombre y un grupo de declaraciones que se ejecutan cuando se llama a la función. Podemos hacer uso de funciones integradas como *void setup()* y *void loop()* o escribir nuevas.

Las funciones se escriben para ejecutar tareas repetitivas y reducir el desorden en un programa. En primer lugar se declara el tipo de la función, que será el valor retornado por la función (*int, void...*). A continuación del tipo, se declara el nombre de la función y, entre paréntesis, los parámetros que se pasan a la función.

```
type functionName(parameters)
{
  statements;
}
```

La siguiente función *int delayVal()*, asigna un valor de retardo en un programa por lectura del valor de un potenciómetro.

```
int delayVal()
{
  int v;               //crea una variable temporal 'v'
  v = analogRead(pot); //lee el valor del potenciómetro
  v /= 4;              //convierte 0-1023 a 0-255
  return v;            //devuelve el valor final de v
}
```

llaves {}

Las llaves definen el comienzo y el final de bloques de función y bloques de declaraciones como *void loop()* y sentencias *for* e *if*. Las llaves deben estar balanceadas (a una llave de apertura { debe seguirle una llave de cierre }). Las llaves no balanceadas provocan errores de compilación.

```
void loop()
{
  statements;
}
```

El entorno Arduino incluye una práctica característica para chequear el balance de llaves. Sólo selecciona una llave y su compañera lógica aparecerá resaltada.

punto y coma ;

Un punto y coma debe usarse al final de cada declaración y separa los elementos del programa. También se usa para separar los elementos en un bucle *for*.

```
int x = 13; //declara la variable 'x' como el entero 13
```

Nota: Olvidar un punto y coma al final de una declaración producirá un error de compilación.

bloques de comentarios /*...*/

Los bloques de comentarios, o comentarios multilínea, son areas de texto ignoradas por el programa y se usan para grandes descripciones de código o comentarios que ayudan a otras personas a entender partes del programa. Empiezan con */** y terminan con ***/** y pueden abarcar multiples lineas.

```
/*
 este es un bloque de comentario encerrado
 no olvides cerrar el comentario
 tienen que estar balanceados!
*/
```

Como los comentarios son ignorados por el programa y no ocupan espacio en memoria deben usarse generósamente y también pueden usarse para «comentar» bloques de código con propositos de depuración.

comentarios de línea //

Comentarios de una línea empiezan con // y terminan con la siguiente línea de código. Como el bloque de comentarios, son ignorados por el programa y no toman espacio en memoria.

```
// este es un comentario de una línea
```

Comentarios de una línea se usan a menudo despues de declaraciones válidas para proporcionar más información sobre qué lleva la declaración o proporcionar un recordatorio en el futuro.

5.2. Variables

Una variable es una forma de llamar y almacenar un valor numérico para usarse después por el programa. Como su nombre indica, las variables son números que pueden cambiarse continuamente al contrario que las constantes, cuyo valor nunca cambia. Una variable necesita ser declarada y, opcionalmente, asignada al valor que necesita para ser almacenada.

```
int inputVariable = 0;          //declara una variable y asigna el valor a 0
inputVariable = analogRead(2); //ajusta la variable al valor del pin
                                //analógico 2
```

Una vez que una variable ha sido asignada, o reasignada, puedes testear su valor para ver si cumple ciertas condiciones, o puedes usarlo directamente.

```
if(inputVariable < 100)  //comprueba si la variable es menor que 100
{
  inputVariable = 100;   //si es cierto asigna el valor 100
}
delay(inputVariable);    //usa la variable como retardo
```

declaración de variable

Todas las variables tienen que ser declaradas antes de que puedan ser usadas. Declarar una variable significa definir su tipo de valor, como *int*, *long*, *float*, etc., definir un nombre específico, y, opcionalmente, asignar un valor inicial. Esto sólo necesita hacerse una vez en un programa pero el valor puede cambiarse en cualquier momento usando aritmética y varias asignaciones.

```
int inputVariable = 0;
```

Una variable puede ser declarada en un número de posiciones en todo el programa y donde esta deficinición tiene lugar determina que partes del programa pueden usar la variable.

ámbito de la variable

Una variable puede ser declarada al comienzo del programa antes del *void setup()*, localmente dentro de funciones, y algunas veces en un bloque de declaración, por ejemplo bucles *for*. Donde la variable es declarada determina el ámbito de la variable, o la abilidad de ciertas partes de un programa de hacer uso de la variable.

Una variable global es una que puede ser vista y usada por cualquier función y declaración en un programa. Esta variable se declara al comienzo del programa, antes de la función *setup()*.

Una variable local es una que se define dentro de una función o como parte de un bucle *for*. Sólo es visible y sólo puede ser usada dentro de la función en la cual fue declarada. Además, es posible tener dos o más variables del mismo nombre en diferentes partes del programa que contienen diferentes valores.

```
int value; //'value' es visible por cualquier función

void setup()
{
  //no se necesita setup
}

void loop()
{
  for(int i=0; i<20;) //'i' es sólo visible dentro del bucle for
  {
    i++;
```

```
  }
  float f; //'f' es sólo visible dentro de loop
}
```

5.3. Tipos de datos

byte

Byte almacena un valor numérico de 8 bits sin puntos decimales. Tienen un rango de 0 a 255.

```
byte someVariable = 180; //declara 'someVariable' como un tipo byte
```

int

Enteros son los tipos de datos primarios para almacenamiento de números sin puntos decimales y almacenan un valor de 16 bits con un rango de -32,768 a 32,767.

```
int someVariable = 1500; //declara 'someVariable' como tipo int
```

long

Tipo de datos de tamaño extendido para enteros largos, sin puntos decimales, almacenados en un valor de 32 bits con un rango de -2,146,483,648 a 2,147,483,647.

```
long someVariable = 90000; //declara 'someVariable' como tipo long
```

float

Un tipo de datos para números en punto flotante, o números que tienen un punto decimal. Los números en punto flotante tienen mayor resolución que los enteros y se almacenan como valor de 32 bits con un rango de -3.4028235E+38 a 3.4028235E+38.

```
float someVariable = 3.14; //declara 'someVariable' como tipo float
```

arrays

Un array es una colección de valores que son accedidos con un índice numérico. Cualquier valor en el array debe llamarse escribiendo el nombre del array y el índice numérico del valor. Los arrays están indexados a cero, con el primer valor en el array comenzando con el índice número 0. Un array necesita ser declarado y opcionalmente asignarle valores antes de que puedan ser usados.

```
int myArray[] = {value0, value1, value2...};
```

Asimismo es posible declarar un array declarando el tipo del array y el tamaño y luego asignarle valores a una posición del índice.

```
int myArray[5];  //declara un array de enteros con 6 posiciones
myArray[3] = 10; //asigna a la cuarta posición del índice el valor 10
```

Para recibir un valor desde un array, asignamos una variable al array y la posición del índice:

```
x = myArray[3]; //x ahora es igual a 10
```

5.4. Aritmética

Los operadores aritméticos incluyen suma, resta, multiplicación y división. Retornan la suma, diferencia, producto o cociente (respectivamente) de dos operandos.

```
y = y+3;
x = x-7;
i = j*6;
r = r/5;
```

La operación es llevada a cabo usando del tipo de datos de los operandos, así $9/4$ devuelve 2 en lugar de 2.25. Si los operandos son de tipos diferentes, el tipo mayor es usado para el cálculo.

Nota: Usar el operador *cast*, por ejemplo *(int)myFloat* para convertir un tipo de variable a otro al vuelo.

asignaciones compuestas

Las asignaciones compuestas combinan una operación aritmética con una asignación de variable. Estas son muy frecuentemente encontradas en bucles *for*. Las asignaciones compuestas más comunes incluyen:

```
x++;    //lo mismo que x = x+1
x--;    //lo mismo que x = x-1
x += y; //lo mismo que x = x+y
x -= y; //lo mismo que x = x-y
x *= y; //lo mismo que x = x*y
x /= y; //lo mismo que x = x/y
```

operadores de comparación

Las comparaciones de una variable o constante con otra se usan a menudo en declaraciones *if* para comprobar si un condición específica es cierta.

```
x == y; //x es igual a y
x != y; //x no es igual a y
x < y;  //x es menor que y
x > y;  //x es mayor que y
x <= y; //x es menor o igual que y
x >= y; //x es mayor o igual que y
```

operadores lógicos

Los operadores lógicos son normalmente una forma de comparar dos expresiones y devuelven TRUE o FALSE dependiendo del operador. Hay tres operadores lógicos, AND, OR y NOT, que se usan a menudo en declaraciones *if*.

```
//AND logico:
if(x>0 && x<5)  //verdadero sólo si las dos expresiones son ciertas

//OR logico:
if(x>0 || y>0)  //verdadero si al menos una expresion es cierta

//NOT logico:
if(!(x>0))      //verdadero sólo si la expresión es falsa
```

5.5. Constantes

El lenguaje Arduino tiene unos cuantos valores predefinidos que se llaman constantes. Se usan para hacer los programas más legibles. Las constantes se clasifican en grupos.

true/false

Estas son constantes Booleanas que definen niveles lógicos. FALSE se define como 0 (cero) mientras TRUE es 1 o un valor distinto de 0.

```
if(b == TRUE)
{
  doSomething;
}
```

high/low

Estas constantes definen los niveles de pin como HIGH o LOW y se usan cuando se leen o se escriben los pines digitales. HIGH esta definido como el nivel 1 lógico, ON ó 5 V, mientras que LOW es el nivel lógico 0, OFF ó 0 V.

```
digitalWrite(13, HIGH);
```

input/output

Constantes usadas con la función *pinMode()* para definir el modo de un pin digital como INPUT u OUTPUT.

```
pinMode(13, OUTPUT);
```

5.6. Control de flujo

if

Las sentencias *if* comprueban si cierta condición ha sido alcanzada y ejecutan todas las sentencias dentro de las llaves si la declaración es cierta. Si es falsa el programa ignora la sentencia.

```
if(someVariable ?? value)
{
  doSomething;
}
```

Nota: Cuídate de usar «=» en lugar de «==» dentro de la declaración de la sentencia *if*.

if... else

if... else permite tomar decisiones «este - o este».

```
if(inputPin == HIGH)
{
  doThingA;
}
else
{
  doThingB;
}
```

else puede preceder a otra comprobación *if*, por lo que multiples y mutuas comprobaciones exclusivas pueden ejecutarse al mismo tiempo.

```
if(inputPin < 500)
{
  doThingA;
}
else if(inputPin >= 1000)
{
  doThingB;
}
else
{
  doThingC;
}
```

for

La sentencia *for* se usa para repetir un bloque de declaraciones encerradas en llaves un número específico de veces. Un contador de incremento se usa a menudo para incrementar y terminar el bucle. Hay tres partes seperadas por punto y coma (*;*), en la cabecera del bucle.

```
for(inicializacion; condicion; expresion)
{
  doSomething;
}
```

La inicialización de una variable local, o contador de incremento, sucede primero y una sola una vez. Cada vez que pasa el bucle, la condición siguiente es comprobada. Si la condición devuelve TRUE, las declaraciones y expresiones que siguen se ejecutan y la condición se comprueba de nuevo. Cuando la condición se vuelve FALSE, el bucle termina.

```
for(int i=0; i<20; i++)     //declara i, comprueba si es menor
{                           //que 20, incrementa i en 1
  digitalWrite(13, HIGH);   //activa el pin 13
  delay(250);               //pausa por un 1/4 de segundo
  digitalWrite(13, LOW);    //desactiva el pin 13
  delay(250);               //pausa por un 1/4 de segundo
}
```

while

El bucle *while* se repetirá continuamente, e infinitamente, hasta que la expresión dentro del paréntesis se vuelva falsa. Algo debe cambiar la variable testeada, o el bucle *while* nunca saldrá. Esto podría estar en tu código, como por ejemplo una variable incrementada, o una condición externa, como un sensor de comprobación.

```
while(someVariable ?? value)
{
  doSomething;
}

while(someVariable < 200)     //comprueba si es menor que 200
{
  doSomething;                //ejecuta las sentencias encerradas
  someVariable++;             //incrementa la variable en 1
}
```

do... while

El bucle *do... while* es un bucle que trabaja de la misma forma que el bucle *while*, con la excepción de que la condición es testeada al final del bucle, por lo que el bucle *do... while* siempre se ejecutará al menos una vez.

```
do
{
  doSomething;
}while(someVariable ?? value);

do
{
  x = readSensors();     //asigna el valor de readSensors() a x
  delay(50);             //pausa de 50 milisegundos
}while(x < 100);         //repite si x es menor que 100
```

5.7. E/S digital

pinMode(pin, mode)

Se usa en *void setup()* para configurar un pin específico para que se comporte o como INPUT o como OUTPUT.

```
pintMode(pin, OUTPUT);  //ajusta 'pin' como salida
```

Los pines digitales de Arduino estan ajustados a INPUT por defecto, por lo que no necesitan ser declarados explícitamente como entradas con *pinMode()*. Los pines configurados como INPUT se dice que están e un estado de alta impedancia.

Hay también convenientes resistencias de *pull-up* de 20KOhm, integradas en el chip ATmega que pueden ser accedidas por software. A estas resistencias *pull-up* integradas se accede de la siguiente manera.

```
pinMode(pin, INPUT);       //ajusta 'pin' como entrada
digitalWrite(pin, HIGH);   //activa la resistencia de pull-up
```

Las resistencias de *pull-up* se usarían normalmente para conectar entradas como interruptores.

Los pines configurados como OUTPUT se dice que están en un estado de baja impedancia y pueden proporcionar 40 mA a otros dispositivos/circuitos.

Nota: Cortocircuitos en los pines de Arduino o corriente excesiva puden dañar o destruir el pin de salida, o dañar el chip ATmega. A menudo es una buena idea conectar un pin OUTPUT a un dispositivo externo en serie con una resistencia de 470Ohm o 1KOhm.

digitalRead(pin)

Lee el valor desde un pin digital especificado con el resultado HIGH o LOW. El pin puede ser especificado o como una variable o como una constante (0 - 13).

```
value = digitalRead(Pin);   //ajusta 'value' igual al pin de entrada
```

digitalWrite(pin, value)

Devuelve o el nivel lógico HIGH o LOW a (activa o desactiva) un pin digital especificado. El pin puede ser especificado como una variable o constante (0 - 13).

```
digitalWrite(pin, HIGH);   //ajusta 'pin' a HIGH

//Ejemplo de programa
int led = 13;   //conecta 'led' al pin 13
int pin = 7;    //conecta 'pushbutton' al pin 7
int value = 0;  //variable para almacenar el valor leido

void setup()
{
  pinMode(led, OUTPUT);   //ajusta el pin 13 como salida
  pinMode(pin, INPUT);    //ajusta el pin 7 como entrada
}

void loop()
{
  value = digitalRead(pin);   //ajusta 'value' igual al pin de entrada
  digitalWrite(led, value);   //ajusta 'led' al valor del boton
}
```

5.8. E/S analógica

analogRead(pin)

Lee el valor desde un pin analógico especificado con una resolución de 10 bits. Esta función sólo trabaja en los pines analógicos (0 - 5). Los valores enteros devueltos están en el rango de 0 a 1023.

```
value = analogRead(pin);   //ajusta 'value' igual a 'pin'
```

Nota: Los pines analógicos al contrario que los digitales, no necesitan ser declarados al principio como INPUT u OUTPUT.

analogWrite(pin, value)

Escribe un valor pseudo analógico usando *modulación por ancho de pulso* («PWM» en inglés) a un pin de salida marcado como PWM. En los Arduinos más nuevos con el chip ATmega168, esta función trabaja en los pines 3, 5, 6, 9, 10 y 11. Los Arduinos más antiguos con un ATmega8 sólo soporta los pines 9, 10 y 11. El valor puede ser especificado como una variable o constante con un valor de 0 a 255.

```
analogWrite(pin, value);   //escribe 'value' al 'pin' analogico
```

Valor	Nivel de salida
0	0 V (t)
64	0 V (3/4 de t) y 5 V (1/4 de t)
128	0 V (1/2 de t) y 5 V (1/2 de t)
192	0 V (1/4 de t) y 5 v (3/4 de t)
255	5 V (t)

Cuadro 5.1: Relación valor-salida con *analogWrite()*

El valor de salida varía de 0 a 5 V según el valor de entrada (de 0 a 255) en función del tiempo de pulso. Si t es el tiempo de pulso, la tabla 5.1 muestra la equivalencia entre el valor y la salida en función del tiempo.

Como esta es una función hardware, el pin generará una onda estática después de una llamada a *analogWrite* en segundo plano hasta la siguiente llamada a *analogWrite* (o una llamada a *digitalRead* o *digitalWrite* en el mismo pin).

```
int led = 10;                //LED con una resistencia de 220ohm en el pin 10
int pin = 0;                 //potenciometro en el pin analogico 0
int value;                   //valor para lectura

void setup(){}               //setup no es necesario

void loop()
{
  value = analogRead(pin);  //ajusta 'value' igual a 'pin'
  value /= 4;               //convierte 0-1023 a 0-255
  analogWrite(led, value);  //saca la señal PWM al led
}
```

5.9. Tiempo

delay(ms)

Pausa tu programa por la cantidad de tiempo especificada en milisegundos, donde 1000 es igual a 1 segundo.

```
delay(1000);   //espera por un segundo
```

millis()

Devuelve el número de milisegundos desde que la placa Arduino empezó a ejecutar el programa actual como un valor *long* sin signo.

```
value = millis(); //ajusta 'value' igual a millis()
```

Nota: Este número se desbordará (resetear de nuevo a cero), después de aproximadamente 9 horas.

5.10. Matemáticas

min(x,y)

Calcula el mínimo de dos números de cualquier tipo de datos y devuelve el número más pequeño.

```
value = min(value, 100);   //asigna a 'value' al más pequeño de 'value' o 100,
                           //asegurandose que nunca superara 100.
```

max(x,y)

Calcula el máximo de dos números de cualquier tipo de datos y devuelve el número más grande.

```
value = max(value, 100);    //asigna a 'value' al más grande de 'value' o 100,
                            //asegurandose que es al menos 100.
```

5.11. Aleatorio

randomSeed(seed)

Asigna un valor, o semilla («seed» en inglés), como el punto de partida para la función *random()*.

```
randomSeed(value);   //asigna 'value' como la semilla aleatoria
```

Como el Arduino es incapaz de crear un número verdaderamente aleatorio, *randomSeed* te permite colocar una variable, constante, u otra función dentro de la función *random*, lo cual ayuda a generar mas números *«random»* aleatorios. Existen una variedad de diferentes semillas, o funciones, que pueden ser usadas en esta función incluyendo *millis()* o incluso *analogRead()* para leer ruido eléctrico a través de un pin analógico.

random(max)
random(min, max)

La función *random* te permite devolver números pseudo aleatorios en un rango especificado por los valores *min* y *max*.

```
value = random(100, 200);   //asigna a 'value' un número aleatorio
                            //entre 100 y 200.
```

Nota: Usar esto después de usar la función *randomSeed()*.

```
int randNumber;                 //variable para almacenar el valor
                                //aleatorio
int led = 10;                   //LED con una resistencia de 220ohm
                                //en el pin 10
```

```
void setup(){}                    //setup no es necesario

void loop()
{
  randomSeed(millis());           //asigna millis() como semilla
  randNumber = random(255);       //numero aleatorio de 0 a 255
  analogWrite(led, randNumber);   //salida de la señal PWM
  delay(500);
}
```

5.12. Serie

Serial.begin(rate)

Abre el puerto serie y asigna la tasa de baudios para la transmisión de datos serie. La típica tasa de baudios para comunicarse con el ordenador es 9600 aunque otras velocidades están soportadas.

```
void setup()
{
  Serial.begin(9600);  //abre el puerto serie
                       //ajusta la tasa de datos a 9600 bps
}
```

Nota: Cuando se usa la comunicación serie, los pines digitales 0 (Rx) y 1 (Tx) no pueden ser usados al mismo tiempo.

Serial.println(data)

Imprime datos al puerto serie, seguido de un retorno de carro y avance de línea automáticos. Este comando toma la misma forma que *Serial.print()*, pero es más fácil para leer datos en el *Serial Monitor*[1].

```
Serial.println(analogValue);   //envia el valor de 'analogValue'

//Ejemplo de aplicacion
void setup()
{
  Serial.begin(9600);             //ajusta al serie a 9600 bps
}

void loop()
{
  Serial.println(analogRead(0));  //envia valor analogico
  delay(1000);                    //pausa por 1 segundo
}
```

[1]Más información en: *4.2 Introducción al Entorno Arduino*

Bibliografía

[1] EVANS, Brian W., (2007) *Arduino Programming Notebook.*

[2] SANTO ORCERO, David, (2007), «Hardware Libre», *Todo Linux*, Madrid: Studio Press. Pp: 12-17.

[3] ARDUINO - Wikipedia, the free encyclopedia, (última modificación, Marzo 2008). Disponible en: `http://en.wikipedia.org/wiki/Arduino`

[4] ARDUINO : Homepage, (última modificación, Marzo 2006). Disponible en: `http://www.arduino.cc/es/`

[5] ARDUINO - Homepage, (última modificación, Julio 2008). Disponible en: `http://www.arduino.cc`

[6] WIRING, (última modificación, Junio 2008). Disponible en: `http://www.wiring.org.co`

Apéndice A

Ejemplos de Aplicación con Arduino

A.1. Salida digital

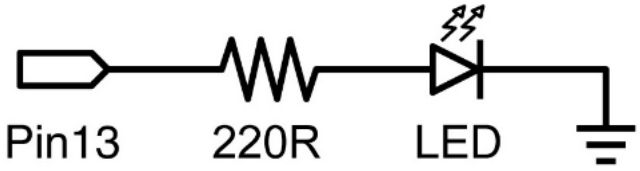


Figura A.1: Esquema de salida digital

Este es el programa básico «hola mundo» usado simplemente para activar o desactivar algo. En este ejemplo, un LED está conectado al pin 13, y parpadea cada segundo. La resistencia puede omitirse en este pin ya que el Arduino tiene una integrada.

```
int ledPin = 13;              //LED en el pin digital 13

void setup()                  //ejecutar una vez
{
  pinMode(ledPin, OUTPUT);    //asigna al pin 13 como salida
}

void loop()                   //ejecutar una y otra vez
{
  digitalWrite(ledPin, HIGH); //activa el LED
  delay(1000);                //pausa 1 segundo
  digitalWrite(ledPin, LOW);  //desactiva el LED
  delay(1000);                //pausa 1 segundo
}
```

A.2. Entrada digital

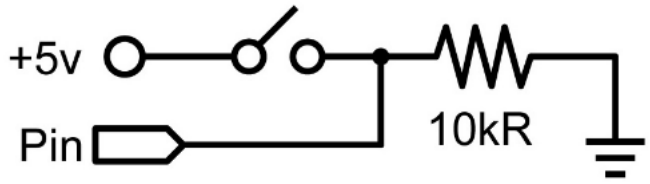


Figura A.2: Esquema de entrada digital

Esta es la forma más simple de entrada con sólo dos estados posibles: ON u OFF. Este ejemplo lee un interruptor simple o pulsador conectado al pin 2. Cuando el interruptor está cerrado el pin de entrada leerá HIGH y activará un LED.

```
int ledPin = 13;                  //pin de salida para el LED
int inPin = 2;                    //pin de entrada (para un interruptor)

void setup()
{
  pinMode(ledPin, OUTPUT);        //declara LED como salida
  pinMode(inPin, INPUT);          //declara el interruptor como entrada
}

void loop()
{
  if(digitalRead(inPin == HIGH)   //comprueba si la entrada esta a HIGH
  {
    digitalWrite(ledPin, HIGH);   //activa el LED
    delay(1000);                  //pausa 1 segundo
    digitalWrite(ledPin, LOW);    //desactiva el LED
    delay(1000);                  //pausa 1 segundo
  }
}
```

A.3. Salida PWM

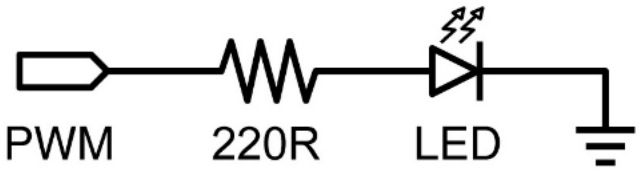


Figura A.3: Esquema de salida PWM

La *modulación de ancho de pulso* (PWM) es una forma de «falsificar» una salida analógica por la salida pulsante. Esto podría usarse para atenuar e iluminar un LED o posteriormente controlar un servomotor. El siguiente ejemplo ilumina y atenua lentamente un LED usando bucles *for*.

```
int ledPin = 9;                 //pin PWM para el LED

void setup(){}

void loop()
{
  for(int i=0; i<=255; i++)     //incrementa el valor para i
  {
    analogWrite(ledPin, i);     //asigna el nivel de brillo a i
    delay(100);                 //pausa 100 ms
  }
  for(int i=255; i>=0; i--)     //decrementa el valor para i
  {
    analogWrite(ledPin, i);     //asigna el nivel de brillo a i
    delay(100);                 //pausa 100 ms
  }
}
```

A.4. Entrada de potenciometro

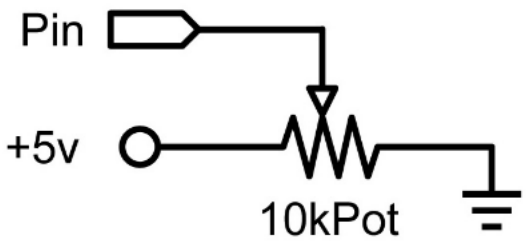


Figura A.4: Esquema de entrada de potenciometro

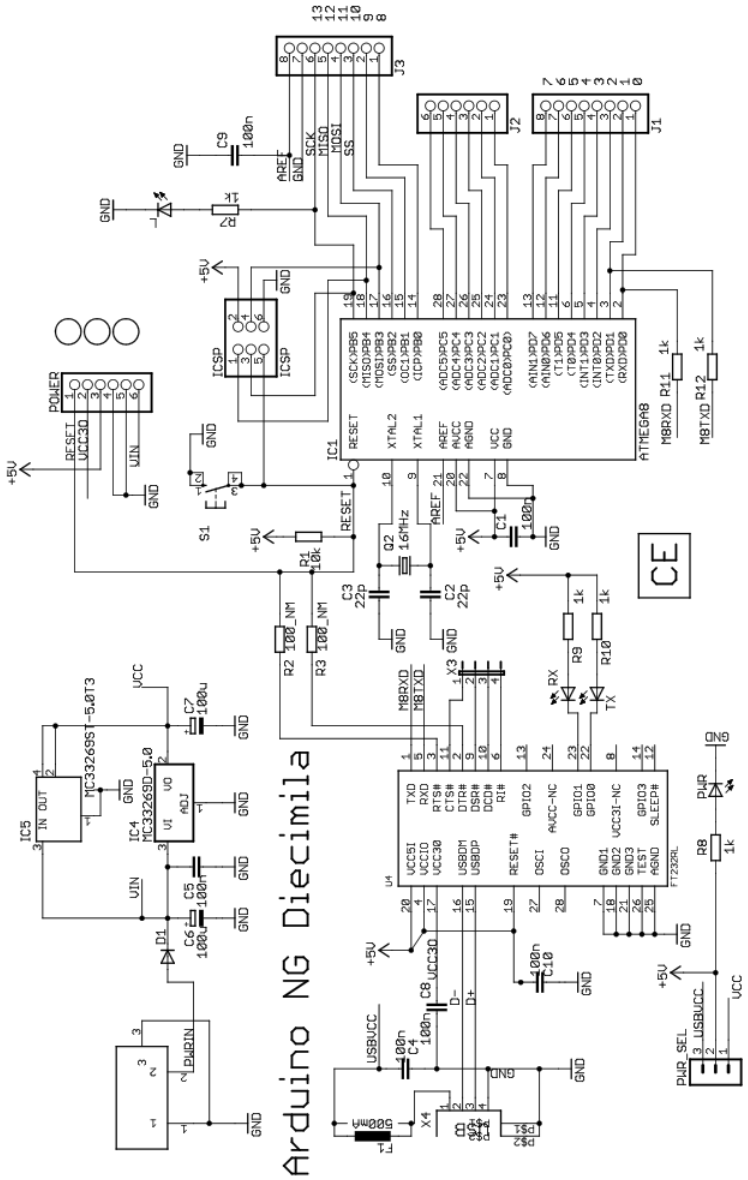
Usando un potenciometro y una de los pines de conversión analógico-digital (ADC) de Arduino es posible leer valores de 0 a 1024. El siguiente ejemplo usa un potenciometro para controlar una frecuencia de parpadeo de un LED.

```
int potPin = 0;                 //pin de entrada para el potenciometro
int ledPin = 13;                //pin de salida para el LED

void setup()
{
  pinMode(ledPin, OUTPUT);      //declara ledPin como OUTPUT
```

```
}

void loop()
{
  digitalWrite(ledPin, HIGH);  //activa ledPin
  delay(analogRead(potPin));   //pausa el programa
  digitalWrite(ledPin, LOW);   //desactiva ledPin
  delay(analogRead(potPin));   //pausa el programa
}
```

Apéndice B

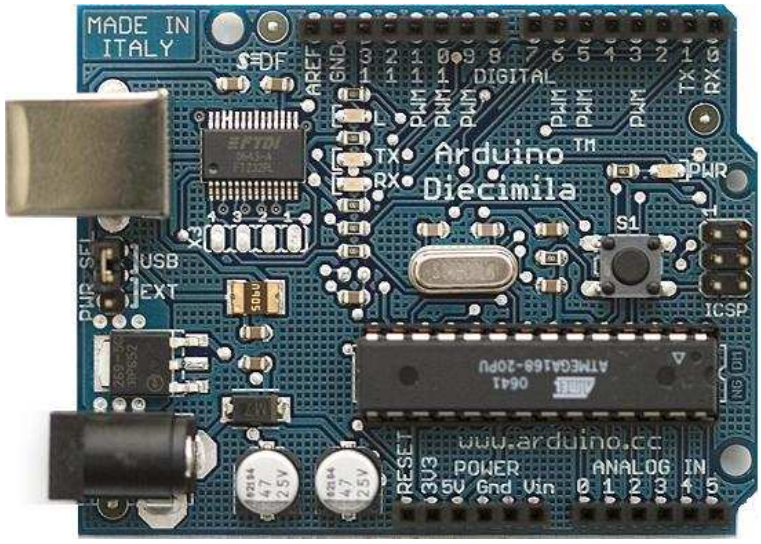
Esquemático de Arduino Diecimila



Manual de Programación

Arduino

La "inteligencia de Arduino" se expresa mediante su lenguaje de programación



Guía rápida de referencia

Traducido y adaptado:
José Manuel Ruiz Gutiérrez

Arduino: Manual de Programación

Datos del documento original

Arduino Notebook: A Beginner's Reference Written
and compiled by Brian W. Evans

With information or inspiration taken from:
http://www.arduino.cc
http://www.wiring.org.co
http://www.arduino.cc/en/Booklet/HomePage
http://cslibrary.stanford.edu/101/

Including material written by:
Massimo Banzi
Hernando Barragin
David Cuartielles
Tom Igoe
Todd Kurt
David Mellis
and others

Published:
First Edition August 2007

Arduino: Manual de Programación

Índice de contenidos

estructura
- estructura
- setup()
- loop()
- funciones
- {} uso de llaves
- ; punto y coma
- /*… */ bloque de comentarios
- // linea de comentario

variables
- variables
- declaración de variables
- variable scope

tipos de datos
- byte
- int
- long
- float
- arrays

aritmética
- aritmética
- composición de asignaciones
- operadores de comparación
- operadores lógicos

constantes
- constantes
- cierto/falso
- alto/bajo
- entrada/salida

control de flujo
- if
- if… else
- for
- while
- do… while

E/S digitales
- pinMode(pin, mode)
- digitalRead(pin)
- digitalWrite(pin, value)

E/S analógicas
- analogRead(pin)
- analogWrite(pin, value)

tiempo
- delay(ms)
- millis()

matemáticas
- min(x, y)
- max(x, y)

aleatorio
- randomSeed(seed)
- random(min, max)

Puerto serie
- Serial.begin(rate)
- Serial.println(data)
- Serial.print(data, data type)

apéndice
- salida digital
- entrada digital
- salida de alto consumo (corriente)
- salida analógica (pwm)
- potenciómetro de entrada
- Resistencia variable de entrada
- Salida a servo

APENDICES

Formas de Conexionadode entradas y salidas
Como escribir una librería para Arduino
Señales analógicas de salida en Arduino (PWM).

Arduino: Manual de Programación

Comunicando Arduino con otros sistemas
Comunicación vía puerto Serie:
Envio de datos desde el PC (PC->Arduino) a Arduino por puerto de comunicación serie:
Envío a petición (toma y dame)
Conversor Analógico-Digital (A/D)
Comunicación serie
Palabras reservadas del IDE de Arduino
Circuitos de interface con Arduino

Arduino: Manual de Programación

estructura de un programa

La estructura básica del lenguaje de programación de Arduino es bastante simple y se compone de al menos dos partes. Estas dos partes necesarias, o funciones, encierran bloques que contienen declaraciones, estamentos o instrucciones.

```
void  setup()
{
  estamentos;
}
  void  loop()
{
  estamentos;
}
```

En donde **setup()** es la parte encargada de recoger la configuración y **loop()** es la que contienen el programa que se ejecutará cíclicamente (de ahí el termino loop –bucle-). Ambas funciones son necesarias para que el programa trabaje.

La función de configuración debe contener la declaración de las variables. Es la primera función a ejecutar en el programa, se ejecuta sólo una vez, y se utiliza para configurar o inicializar pinMode (modo de trabajo de las E/S), configuración de la comunicación en serie y otras.

La función bucle (loop) siguiente contiene el código que se ejecutara continuamente (lectura de entradas, activación de salidas, etc) Esta función es el núcleo de todos los programas de Arduino y la que realiza la mayor parte del trabajo.

setup()

La función **setup()** se invoca una sola vez cuando el programa empieza. Se utiliza para inicializar los modos de trabajo de los pins, o el puerto serie. Debe ser incluido en un programa aunque no haya declaración que ejecutar.

```
void  setup()
      {
            pinMode(pin,  OUTPUT);  // configura el  'pin'  como salida
      }
```

loop()

Arduino: Manual de Programación

Después de llamar a **setup(),** la función **loop()** hace precisamente lo que sugiere su nombre, se ejecuta de forma cíclica, lo que posibilita que el programa este respondiendo continuamente ante los eventos que se produzcan en la tarjeta

```
void loop()
      {
            digitalWrite(pin, HIGH);   // pone en uno (on, 5v) el ´pin´
            delay(1000);               // espera un segundo (1000 ms)
            digitalWrite(pin, LOW);    // pone en cero (off, 0v.) el ´pin´
            delay(1000);
      }
```

funciones

Una función es un bloque de código que tiene un nombre y un conjunto de estamentos que son ejecutados cuando se llama a la función. Son funciones setup() y loop() de las que ya se ha hablado. Las funciones de usuario pueden ser escritas para realizar tareas repetitivas y para reducir el tamaño de un programa. Las funciones se declaran asociadas a un tipo de valor **"type"**. Este valor será el que devolverá la función, por ejemplo **'int'** se utilizará cuando la función devuelva un dato numérico de tipo entero. Si la función no devuelve ningún valor entonces se colocará delante la palabra "**void",** que significa "función vacía". Después de declarar el tipo de dato que devuelve la función se debe escribir el nombre de la función y entre paréntesis se escribirán, si es necesario, los parámetros que se deben pasar a la función para que se ejecute.

```
type  nombreFunción(parámetros)
      {
          estamentos;
      }
```

La función siguiente devuelve un número entero, **delayVal()** se utiliza para poner un valor de retraso en un programa que lee una variable analógica de un potenciómetro conectado a una entrada de Arduino. Al principio se declara como una variable local, ´**v**´ recoge el valor leído del potenciómetro que estará comprendido entre 0 y 1023, luego se divide el valor por 4 para ajustarlo a un margen comprendido entre 0 y 255, finalmente se devuelve el valor ´**v**´ y se retornaría al programa principal. Esta función cuando se ejecuta devuelve el valor de tipo entero ´**v**´

```
int  delayVal()
   {
      int  v;                    // crea una variable temporal  'v'
      v= analogRead(pot);        // lee el valor del potenciómetro
      v  /=  4;                  // convierte  0-1023 a  0-255
      return  v;                 // devuelve el valor final
   }
```

Arduino: Manual de Programación

{} entre llaves

Las llaves sirven para definir el principio y el final de un bloque de instrucciones. Se utilizan para los bloques de programación setup(), loop(), if.., etc.

```
type  funcion()
      {
        estamentos;
      }
```

Una llave de apertura "{" siempre debe ir seguida de una llave de cierre "}", si no es así el programa dará errores.

El entorno de programación de Arduino incluye una herramienta de gran utilidad para comprobar el total de llaves. Sólo tienes que hacer click en el punto de inserción de una llave abierta e inmediatamente se marca el correspondiente cierre de ese bloque (llave cerrada).

; punto y coma

El punto y coma ";" se utiliza para separar instrucciones en el lenguaje de programación de Arduino. También se utiliza para separar elementos en una instrucción de tipo "bucle for".

```
int  x  =  13;      // declara la variable  'x' como tipo entero de valor  13
```

Nota: Olvidarse de poner fin a una línea con un punto y coma se traducirá en un error de compilación. El texto de error puede ser obvio, y se referirá a la falta de una coma, o puede que no. Si se produce un error raro y de difícil detección lo primero que debemos hacer es comprobar que los puntos y comas están colocados al final de las instrucciones.

/*… */ bloque de comentarios

Los bloques de comentarios, o multi-línea de comentarios, son áreas de texto ignorados por el programa que se utilizan para las descripciones del código o comentarios que ayudan a comprender el programa. Comienzan con / * y terminan con * / y pueden abarcar varias líneas.

```
/*   esto es un bloque de comentario
     no se debe olvidar cerrar los comentarios
     estos deben estar equilibrados
*/
```

Debido a que los comentarios son ignorados por el programa y no ocupan espacio en la memoria de Arduino pueden ser utilizados con generosidad y también pueden utilizarse para "comentar" bloques de código con el propósito de anotar informaciones para depuración.

Nota: Dentro de una misma línea de un bloque de comentarios no se puede escribir otra bloque de comentarios (usando /* .. */)

// línea de comentarios

Una línea de comentario empieza con / / y terminan con la siguiente línea de código. Al igual que los comentarios de bloque, los de línea son ignoradas por el programa y no ocupan espacio en la memoria.

```
// esto es un comentario
```

Una línea de comentario se utiliza a menudo después de una instrucción, para proporcionar más información acerca de lo que hace esta o para recordarla más adelante.

variables

Una variable es una manera de nombrar y almacenar un valor numérico para su uso posterior por el programa. Como su nombre indica, las variables son números que se pueden variar continuamente en contra de lo que ocurre con las constantes cuyo valor nunca cambia. Una variable debe ser declarada y, opcionalmente, asignarle un valor. El siguiente código de ejemplo declara una variable llamada *variableEntrada* y luego le asigna el valor obtenido en la entrada analógica del PIN2:

```
int variableEntrada = 0;          // declara una variable y le asigna el valor 0
variableEntrada = analogRead(2);// la variable recoge el valor analógico del PIN2
```

'variableEntrada' es la variable en sí. La primera línea declara que será de tipo entero "int". La segunda línea fija a la variable el valor correspondiente a la entrada analógica PIN2. Esto hace que el valor de PIN2 sea accesible en otras partes del código.

Una vez que una variable ha sido asignada, o re-asignada, usted puede probar su valor para ver si cumple ciertas condiciones (instrucciones if..), o puede utilizar directamente su valor. Como ejemplo ilustrativo veamos tres operaciones útiles con variables: el siguiente código prueba si la variable *"entradaVariable"* es inferior a 100, si es cierto se asigna el valor 100 a *"entradaVariable*" y, a continuación, establece un retardo (delay) utilizando como valor "*entradaVariable"* que ahora será como mínimo de valor 100:

```
if (entradaVariable < 100) // pregunta si la variable es menor de 100
  {
    entradaVariable = 100;// si es cierto asigna el valor 100 a esta
  }
delay(entradaVariable);     // usa el valor como retardo
```

Nota: Las variables deben tomar nombres descriptivos, para hacer el código más legible. Nombres de variables pueden ser "contactoSensor" o "pulsador", para ayudar al programador y a cualquier otra persona a leer el código y entender lo que representa la variable. Nombres de variables como "var" o "valor", facilitan muy poco que el código sea inteligible. Una variable puede ser cualquier nombre o palabra que no sea una *palabra reservada* en el entorno de Arduino.

declaración de variables

Todas las variables tienen que declararse antes de que puedan ser utilizadas. Para declarar una variable se comienza por definir su tipo como **int** (entero), **long** (largo), **float** (coma flotante), etc, asignándoles siempre un nombre, y, opcionalmente, un valor inicial. Esto sólo debe hacerse una vez en un programa, pero el valor se puede cambiar en cualquier momento usando aritmética y reasignaciones diversas.

El siguiente ejemplo declara la variable *entradaVariable* como una variable de tipo entero *"int",* y asignándole un valor inicial igual a cero. Esto se llama una asignación.

```
int entradaVariable = 0;
```

Una variable puede ser declarada en una serie de lugares del programa y en función del lugar en donde se lleve a cabo la definición esto determinará en que partes del programa se podrá hacer uso de ella.

Utilización de una variable

Una variable puede ser declarada al inicio del programa antes de la parte de configuración *setup()*, a nivel local dentro de las funciones, y, a veces, dentro de un bloque, como para los bucles del tipo if.. for.., etc. En función del lugar de declaración de la variable así se determinara el ámbito de aplicación, o la capacidad de ciertas partes de un programa para hacer uso de ella.

Una *variable global* es aquella que puede ser vista y utilizada por cualquier función y estamento de un programa. Esta variable se declara al comienzo del programa, antes de *setup()*.

Una *variable local* es aquella que se define dentro de una función o como parte de un bucle. Sólo es visible y sólo puede utilizarse dentro de la función en la que se declaró.

Por lo tanto, es posible tener dos o más variables del mismo nombre en diferentes partes del mismo programa que pueden contener valores diferentes. La garantía de que sólo una función tiene acceso a sus variables dentro del programa simplifica y reduce el potencial de errores de programación.

El siguiente ejemplo muestra cómo declarar a unos tipos diferentes de variables y la visibilidad de cada variable:

```
int value; // 'value' es visible para cualquier función
void setup()
      {
            // no es necesario configurar
      }

void loop()
      {
         for (int i=0; i<20;)      // 'i' solo es visible
            {    // dentro del bucle for
                  i++;
            }
         float f;// 'f' es visible solo
      }     // dentro del bucle
```

byte

Byte almacena un valor numérico de 8 bits sin decimales. Tienen un rango entre 0 y 255

```
byte unaVariable = 180;  // declara 'unaVariable' como tipo byte
```

Int

Enteros son un tipo de datos primarios que almacenan valores numéricos de 16 bits sin decimales comprendidos en el rango 32,767 to -32,768.

```
int unaVariable = 1500;  // declara 'unaVariable' como una variable
                         de tipo entero
```

Nota: Las variables de tipo entero “int” pueden sobrepasar su valor máximo o mínimo como consecuencia de una operación. Por ejemplo, si x = 32767 y una posterior declaración agrega 1 a x, x = x + 1 entonces el valor se x pasará a ser -32.768. (algo así como que el valor da la vuelta)

long

El formato de variable numérica de tipo extendido “long” se refiere a números enteros (tipo 32 bits) sin decimales que se encuentran dentro del rango -2147483648 a 2147483647.

```
long unaVariable = 90000; // declara 'unaVariable' como tipo long
```

float

El formato de dato del tipo “punto flotante” “float” se aplica a los números con decimales. Los números de punto flotante tienen una mayor resolución que los de 32 bits con un rango comprendido 3.4028235E +38 a +38-3.4028235E.

```
float unaVariable = 3.14; // declara 'unaVariable' como tipo flotante
```

Nota: Los números de punto flotante no son exactos, y pueden producir resultados extraños en las comparaciones. Los cálculos matemáticos de punto flotante son también mucho más lentos que los del tipo de números enteros, por lo que debe evitarse su uso si es posible.

arrays

Un array es un conjunto de valores a los que se accede con un número índice. Cualquier valor puede ser recogido haciendo uso del nombre de la matriz y el número del índice. El primer valor de la matriz es el que está indicado con el índice 0, es decir el primer valor del conjunto es el de la posición 0. Un array tiene que ser declarado y opcionalmente asignados valores a cada posición antes de ser utilizado

```
int miArray[] = {valor0, valor1, valor2...}
```

Del mismo modo es posible declarar una matriz indicando el tipo de datos y el tamaño y posteriormente, asignar valores a una posición especifica:

```
int miArray[5];      // declara un array de enteros de 6 posiciones
miArray[3] = 10;     // asigna l valor 10 a la posición 4
```

Para leer de un array basta con escribir el nombre y la posición a leer:

```
x = miArray[3];      // x ahora es igual a 10 que está en la posición 3
                     del array
```

Las matrices se utilizan a menudo para estamentos de tipo bucle, en los que la variable de incremento del contador del bucle se utiliza como índice o puntero del array. El siguiente ejemplo usa una matriz para el parpadeo de un LED.

Utilizando un bucle tipo *for*, el contador comienza en cero 0 y escribe el valor que figura en la posición de índice 0 en la serie que hemos escrito dentro del array parpadeo[], en este caso 180, que se envía a la salida analógica tipo PWM configurada en el PIN10, se hace una pausa de 200 ms y a continuación se pasa al siguiente valor que asigna el índice "i".

```
int ledPin = 10;       // Salida LED en el PIN 10
byte parpadeo[] = {180, 30, 255, 200, 10, 90, 150, 60}; // array de 8 valores
                                                           diferentes
    void setup()
    {
      pinMode(ledPin, OUTPUT); //configura la salida PIN 10
    }

      void loop()        // bucle del programa

    {
      for(int i=0; i<8; i++) // crea un bucle tipo for utilizando la variable i de 0 a 7
    {

      analogWrite(ledPin, parpadeo[i]); // escribe en la salida PIN 10 el valor al
                                           que apunta i dentro del array
                                           parpadeo[]
    delay(200);  // espera 200ms

    }
}
```

aritmética

Los operadores aritméticos que se incluyen en el entorno de programación son suma, resta, multiplicación y división. Estos devuelven la suma, diferencia, producto, o cociente (respectivamente) de dos operandos

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

La operaciones se efectúa teniendo en cuanta el tipo de datos que hemos definido para los operandos (int, dbl, float, etc..), por lo que, por ejemplo, si definimos 9 y 4 como enteros "int", 9 / 4 devuelve de resultado 2 en lugar de 2,25 ya que el 9 y 4 se valores de tipo entero "int" (enteros) y no se reconocen los decimales con este tipo de datos.

Esto también significa que la operación puede sufrir un desbordamiento si el resultado es más grande que lo que puede ser almacenada en el tipo de datos. Recordemos el alcance de los tipos de datos numéricos que ya hemos explicado anteriormente.

Si los operandos son de diferentes tipos, para el cálculo se utilizará el tipo más grande de los operandos en juego. Por ejemplo, si uno de los números (operandos) es del tipo float y otra de tipo integer, para el cálculo se utilizará el método de float es decir el método de coma flotante.

Elija el tamaño de las variables de tal manera que sea lo suficientemente grande como para que los resultados sean lo precisos que usted desea. Para las operaciones que requieran decimales utilice variables tipo float, pero sea consciente de que las operaciones con este tipo de variables son más lentas a la hora de realizarse el computo..

Nota: Utilice el operador *(int) myFloat* para convertir un tipo de variable a otro sobre la marcha. Por ejemplo, **i = (int) 3,6** establecerá i igual a **3**.

asignaciones compuestas

Las asignaciones compuestas combinan una operación aritmética con una variable asignada. Estas son comúnmente utilizadas en los bucles tal como se describe más adelante. Estas asignaciones compuestas pueden ser:

```
x ++      // igual que x = x + 1,     o incrementar x en + 1
x --      // igual que x = x - 1,     o decrementar x en -1
x += y    // igual que x = x + y,     o incrementra x en +y
x -= y    // igual que x = x - y,     o decrementar x en -y
x *= y    // igual que x = x * y,     o multiplicar x por y
x /= y    // igual que x = x / y,     o dividir x por y
```

Nota: Por ejemplo, x * = 3 hace que x se convierta en el triple del antiguo valor x y por lo tanto x es reasignada al nuevo valor .

operadores de comparación

Las comparaciones de una variable o constante con otra se utilizan con frecuencia en las estructuras condicionales del tipo *if*.. para testear si una condición es verdadera. En los ejemplos que siguen en las próximas páginas se verá su utilización práctica usando los siguientes tipo de condicionales:

```
x == y    // x es igual a y
x != y    // x no es igual a y
x <  y    // x es menor que y
x >  y    // x es mayor que y
```

```
x <= y      // x es menor o igual que  y
x >= y      // x es mayor o igual que y
```

operadores lógicos

Los operadores lógicos son usualmente una forma de comparar dos expresiones y devolver un **VERDADERO** o **FALSO** dependiendo del operador. Existen tres operadores lógicos, **AND (&&)**, **OR (||)** y **NOT (!)**, que a menudo se utilizan en estamentos de tipo if..:

```
Logical AND:
if (x > 0 && x < 5)       // cierto sólo si las dos expresiones son ciertas

Logical OR:
if (x > 0 || y > 0)       // cierto si una cualquiera de las expresiones
                          es cierta

Logical NOT:
if (!x > 0)               // cierto solo si la expresión es falsa
```

constantes

El lenguaje de programación de Arduino tiene unos valores predeterminados, que son llamados constantes. Se utilizan para hacer los programas más fáciles de leer. Las constantes se clasifican en grupos.

cierto/falso (true/false)

Estas son constantes booleanas que definen los niveles **HIGH** (alto) y **LOW** (bajo) cuando estos se refieren al estado de las salidas digitales. **FALSE** se asocia con **0** (cero), mientras que **TRUE** se asocia con **1**, pero TRUE también puede ser cualquier otra cosa excepto cero. Por lo tanto, en sentido booleano, -1, 2 y -200 son todos también se define como TRUE. (esto es importante tenerlo en cuanta)

```
if (b == TRUE);
{
ejecutar las instrucciones;
}
```

high/low

Estas constantes definen los niveles de salida altos o bajos y se utilizan para la lectura o la escritura digital para las patillas. **ALTO** se define como en la lógica de nivel 1, **ON**, ó 5 voltios, mientras que **BAJO** es lógica nivel 0, **OFF**, o 0 voltios.

```
digitalWrite(13,  HIGH); // activa la salida 13 con un nivel alto (5v.)
```

input/output

Estas constantes son utilizadas para definir, al comienzo del programa, el modo de funcionamiento de los pines mediante la instrucción *pinMode* de tal manera que el pin puede ser una **entrada INPUT** o una **salida OUTPUT**.

```
pinMode(13,  OUTPUT); // designamos que el PIN 13 es una salida
```

if (si)

if es un estamento que se utiliza para probar si una determinada condición se ha alcanzado, como por ejemplo averiguar si un valor analógico está por encima de un cierto número, y ejecutar una serie de declaraciones (operaciones) que se escriben dentro de llaves, si es verdad. Si es falso (la condición no se cumple) el programa salta y no ejecuta las operaciones que están dentro de las llaves, El formato para if es el siguiente:

```
if (unaVariable ?? valor)
{
ejecutaInstrucciones;
}
```

En el ejemplo anterior se compara una variable con un valor, el cual puede ser una variable o constante. Si la comparación, o la condición entre paréntesis se cumple (es cierta), las declaraciones dentro de los corchetes se ejecutan. Si no es así, el programa salta sobre ellas y sigue.

Nota: Tenga en cuenta el uso especial del símbolo '=', poner dentro de if (x = 10), podría parecer que es valido pero sin embargo no lo es ya que esa expresión asigna el valor 10 a la variable x, por eso dentro de la estructura if se utilizaría X==10 que en este caso lo que hace el programa es comprobar si el valor de x es 10.. Ambas cosas son distintas por lo tanto dentro de las estructuras if, cuando se pregunte por un valor se debe poner el signo doble de igual "=="

if… else (si….. sino ..)

if… else viene a ser un estructura que se ejecuta en respuesta a la *idea "si esto no se cumple haz esto otro"*. Por ejemplo, si se desea probar una entrada digital, y hacer una cosa si la entrada fue alto o hacer otra cosa si la entrada es baja, usted escribiría que de esta manera:

```
if (inputPin == HIGH) // si el valor de la entrada inputPin es alto
        {
                instruccionesA; //ejecuta si se cumple la condición
        }
else
        {
                instruccionesB; //ejecuta si no se cumple la condición
        }
```

Else puede ir precedido de otra condición de manera que se pueden establecer varias estructuras condicionales de tipo unas dentro de las otras (anidamiento) de forma que sean mutuamente excluyentes pudiéndose ejecutar a la vez. Es incluso posible tener un número ilimitado de estos condicionales. Recuerde sin embargo qué sólo un conjunto de declaraciones se llevará a cabo dependiendo de la condición probada:

```
if (inputPin < 500)
        {
                instruccionesA; // ejecuta las operaciones A
        }
else if (inputPin >= 1000)
        {
                instruccionesB; // ejecuta las operacione B
        }
else
        {
                instruccionesC; // ejecuta las operaciones C
        }
```

Nota: Un estamento de tipo if prueba simplemente si la condición dentro del paréntesis es verdadera o falsa. Esta declaración puede ser cualquier declaración válida. En el anterior ejemplo, si cambiamos y ponemos (inputPin == HIGH). En este caso, el estamento if sólo chequearía si la entrada especificado esta en nivel alto (HIGH), o +5 v.

for

La declaración **for** se usa para repetir un bloque de sentencias encerradas entre llaves un número determinado de veces. Cada vez que se ejecutan las instrucciones del bucle se

vuelve a testear la condición. La declaración for tiene tres partes separadas por (;) vemos el ejemplo de su sintaxis:

```
for (inicialización; condición; expresión)
{
      ejecutaInstrucciones;
}
```

La *inicialización* de una variable local se produce una sola vez y la *condición* se testea cada vez que se termina la ejecución de las instrucciones dentro del bucle. Si la condición sigue cumpliéndose, las instrucciones del bucle se vuelven a ejecutar. Cuando la condición no se cumple, el bucle termina.

El siguiente ejemplo inicia el entero i en el 0, y la condición es probar que el valor es inferior a 20 y si es cierto i se incrementa en 1 y se vuelven a ejecutar las instrucciones que hay dentro de las llaves:

```
for (int i=0; i<20; i++)          // declara i, prueba que es menor que
                                  20, incrementa i en 1
{
      digitalWrite(13, HIGH);     // envía un 1 al pin 13
      delay(250);                 // espera ¼ seg.
      digitalWrite(13, LOW);      // envía un 0 al pin 13
      delay(250);                 // espera ¼ de seg.
}
```

Nota: El bucle en el lenguaje C es mucho más flexible que otros bucles encontrados en algunos otros lenguajes de programación, incluyendo BASIC. Cualquiera de los tres elementos de cabecera puede omitirse, aunque el punto y coma es obligatorio. También las declaraciones de inicialización, condición y expresión puede ser cualquier estamento válido en lenguaje C sin relación con las variables declaradas. Estos tipos de estados son raros pero permiten disponer soluciones a algunos problemas de programación raras.

while

Un bucle del tipo **while** es un bucle de ejecución continua mientras se cumpla la expresión colocada entre paréntesis en la cabecera del bucle. La *variable de prueba* tendrá que cambiar para salir del bucle. La situación podrá cambiar a expensas de una expresión dentro el código del bucle o también por el cambio de un valor en una entrada de un sensor

```
while (unaVariable ?? valor)
{
```

```
        ejecutarSentencias;
}
```

El siguiente ejemplo testea si la variable "unaVariable" es inferior a 200 y, si es verdad, ejecuta las declaraciones dentro de los corchetes y continuará ejecutando el bucle hasta que 'unaVariable' no sea inferior a 200.

```
While (unaVariable < 200)      // testea si es menor que 200
{
        instrucciones;         // ejecuta las instrucciones entre llaves
        unaVariable++;         // incrementa la variable en 1
}
```

do… while

El bucle **do while** funciona de la misma manera que el bucle while, con la salvedad de que la condición se prueba al final del bucle, por lo que el bucle siempre se ejecutará al menos una vez.

```
do
{
        Instrucciones;
} while (unaVariable ?? valor);
```

El siguiente ejemplo asigna el valor leído *leeSensor()* a la variable 'x', espera 50 milisegundos, y luego continua mientras que el valor de la 'x' sea inferior a 100:

```
do
{
        x = leeSensor();
        delay(50);
} while (x < 100);
```

pinMode(pin, mode)

Esta instrucción es utilizada en la parte de configuración *setup ()* y sirve para configurar el modo de trabajo de un **PIN** pudiendo ser **INPUT** (entrada) u **OUTPUT** (salida).

```
pinMode(pin, OUTPUT);  // configura 'pin' como salida
```

Los terminales de Arduino, por defecto, están configurados como entradas, por lo tanto no es necesario definirlos en el caso de que vayan a trabajar como entradas. Los pines

configurados como entrada quedan, bajo el punto de vista eléctrico, como entradas en estado de alta impedancia.

Estos pines tienen a nivel interno una resistencia de 20 KΩ a las que se puede acceder mediante software. Estas resistencias se accede de la siguiente manera:

```
pinMode(pin,  INPUT);      // configura el 'pin' como entrada
digitalWrite(pin,  HIGH);  //  activa las resistencias internas
```

Las resistencias internas normalmente se utilizan para conectar las entradas a interruptores. En el ejemplo anterior no se trata de convertir un pin en salida, es simplemente un método para activar las resistencias interiores.

Los pins configurado como OUTPUT (salida) se dice que están en un estado de baja impedancia estado y pueden proporcionar **40 mA** (miliamperios) de corriente a otros dispositivos y circuitos. Esta corriente es suficiente para alimentar un diodo LED (no olvidando poner una resistencia en serie), pero no es lo suficiente grande como para alimentar cargas de mayor consumo como relés, solenoides, o motores.

Un cortocircuito en las patillas Arduino provocará una corriente elevada que puede dañar o destruir el chip Atmega. A menudo es una buena idea conectar en la OUTUPT (salida) una resistencia externa de 470 o de 1000 Ω.

digitalRead(pin)

Lee el valor de un pin (definido como digital) dando un resultado **HIGH** (alto) o **LOW** (bajo). El pin se puede especificar ya sea como una variable o una constante (0-13).

```
valor = digitalRead(Pin);  // hace que 'valor sea igual al estado leído
                           en ´Pin´
```

digitalWrite(pin, value)

Envía al ´pin´ definido previamente como OUTPUT el valor HIGH o LOW (poniendo en 1 o 0 la salida). El pin se puede especificar ya sea como una variable o como una constante (0-13).

```
digitalWrite(pin,  HIGH);  //  deposita en el 'pin' un valor HIGH (alto o 1)
```

El siguiente ejemplo lee el estado de un pulsador conectado a una entrada digital y lo escribe en el ´pin´de salida LED:

```
int  led     = 13;      // asigna a LED el valor 13
int  boton   =  7;      // asigna a botón el valor  7
int  valor   =  0;      // define el valor y le asigna el valor 0

void  setup()
{
      pinMode(led,  OUTPUT); // configura el led (pin13) como salida
      pinMode(boton,  INPUT); // configura botón (pin7) como entrada
}

void  loop()
{
      valor  =  digitalRead(boton); //lee el estado de la entrada botón
      digitalWrite(led,  valor); // envía a la salida ´led´el valor leído
}
```

analogRead(pin)

Lee el valor de un determinado pin definido como entrada analógica con una *resolución de 10 bits*. Esta instrucción sólo funciona en los pines (0-5). El rango de valor que podemos leer oscila de 0 a 1023.

```
valor  =  analogRead(pin);  // asigna a valor  lo que lee en la entrada ´pin'
```

Nota: Los pins analógicos (0-5) a diferencia de los pines digitales, no necesitan ser declarados como INPUT u OUPUT ya que son siempre INPUT´s.

analogWrite(pin, value)

Esta instrucción sirve para escribir un pseudo-valor analógico utilizando el procedimiento de modulación por ancho de pulso (PWM) a uno de los pin´s de Arduino marcados como *"pin PWM"*. El más reciente Arduino, que implementa el chip **ATmega168, permite habilitar como salidas analógicas tipo PWM los pines 3, 5, 6, 9, 10 y 11**. Los modelos de Arduino más antiguos que implementan el chip **ATmega8, solo tiene habilitadas para esta función los pines 9, 10 y 11**. El valor que se puede enviar a estos pines de salida analógica puede darse en forma de variable o constante, pero siempre con un margen de 0-255.

```
analogWrite(pin,  valor);    // escribe 'valor' en el 'pin' definido como
                             analógico
```

Si enviamos el valor 0 genera una salida de 0 voltios en el pin especificado; un valor de 255 genera una salida de 5 voltios de salida en el pin especificado. Para valores de entre 0 y 255, el pin saca tensiones entre 0 y 5 voltios - el valor HIGH de salida equivale a 5v (5 voltios). Teniendo en cuenta el concepto de señal PWM , por ejemplo, un valor de 64

equivaldrá a mantener 0 voltios de tres cuartas partes del tiempo y 5 voltios a una cuarta parte del tiempo; un valor de 128 equivaldrá a mantener la salida en 0 la mitad del tiempo y 5 voltios la otra mitad del tiempo, y un valor de 192 equivaldrá a mantener en la salida 0 voltios una cuarta parte del tiempo y de 5 voltios de tres cuartas partes del tiempo restante.

Debido a que esta es una función de hardware, en el pin de salida analógica (PWN) se generará una onda constante después de ejecutada la instrucción *analogWrite* hasta que se llegue a ejecutar otra instrucción *analogWrite* (o una llamada a digitalRead o digitalWrite en el mismo pin).

Nota: Las salidas analógicas a diferencia de las digitales, no necesitan ser declaradas como INPUT u OUTPUT..

El siguiente ejemplo lee un valor analógico de un pin de entrada analógica, convierte el valor dividiéndolo por 4, y envía el nuevo valor convertido a una salida del tipo PWM o salida analógica:

```
int led = 10;          // define el pin 10 como ´led´
int analog =  0;       // define el pin 0 como ´analog´
int valor;             // define la variable ´valor´

void setup(){}         // no es necesario configurar entradas y salidas

void loop()
{
      valor = analogRead(analog);     // lee el pin 0 y lo asocia a la
                                      variable valor
      valor /= 4; //divide valor entre 4 y lo reasigna a valor
      analogWrite(led, value); // escribe en el pin10 valor
}
```

delay(ms)

Detiene la ejecución del programa la cantidad de tiempo en ms que se indica en la propia instrucción. De tal manera que 1000 equivale a 1seg.

```
delay(1000);      // espera 1 segundo
```

millis()

Devuelve el número de milisegundos transcurrido desde el inicio del programa en Arduino hasta el momento actual. Normalmente será un valor grande (dependiendo del

tiempo que este en marcha la aplicación después de cargada o después de la última vez que se pulsó el botón “reset” de la tarjeta)..

valor = millis(); *// valor recoge el número de milisegundos*

Nota: Este número se desbordará (si no se resetea de nuevo a cero), después de aproximadamente 9 horas.

min(x, y)

Calcula el mínimo de dos números para cualquier tipo de datos devolviendo el número más pequeño.

valor = min(valor, 100); *// asigna a valor el más pequeños de los dos números especificados.*

Si 'valor' es menor que 100 valor recogerá su propio valor si ´valor´ es mayor que 100 valor pasara a valer 100.

max(x, y)

Calcula el máximo de dos números para cualquier tipo de datos devolviendo el número mayor de los dos.

valor = max(valor, 100); *// asigna a valor el mayor de los dos números 'valor' y 100.*

De esta manera nos aseguramos de que valor será como mínimo 100.

randomSeed(seed)

Establece un valor, o semilla, como punto de partida para la función *random()*.

randomSeed(valor); *// hace que valor sea la semilla del random*

Debido a que Arduino es incapaz de crear un verdadero número aleatorio, *randomSeed* le permite colocar una variable, constante, u otra función de control dentro de la función random, lo que permite generar números aleatorios "al azar". Hay una variedad de semillas, o funciones, que pueden ser utilizados en esta función, incluido

millis () o incluso analogRead () que permite leer ruido eléctrico a través de un pin analógico.

random(max)
random(min, max)

La función **random** devuelve un número aleatorio entero de un intervalo de valores especificado entre los valores min y max.

```
valor = random(100, 200);      // asigna a la variable 'valor' un numero aleatorio
                               // comprendido entre 100-200
```

Nota: Use esta función después de usar el randomSeed().

El siguiente ejemplo genera un valor aleatorio entre 0-255 y lo envía a una salida analógica PWM :

```
int randNumber;      // variable que almacena el valor aleatorio
int led = 10;        // define led como 10

void setup() {}      // no es necesario configurar nada

void loop()
{
randomSeed(millis());            // genera una semilla para aleatorio a partir
                                 // de la función millis()
randNumber = random(255);        // genera número aleatorio entre 0-255
analogWrite(led, randNumber);    // envía a la salida led de tipo PWM el valor
delay(500);                      // espera 0,5 seg.
}
```

Serial.begin(rate)

Abre el puerto serie y fija la velocidad en baudios para la transmisión de datos en serie. El valor típico de velocidad para comunicarse con el ordenador es 9600, aunque otras velocidades pueden ser soportadas.

```
void setup()
    {
        Serial.begin(9600);    // abre el Puerto serie
    }                          // configurando la velocidad en 9600 bps
```

Nota: Cuando se utiliza la comunicación serie los pins digital 0 (RX) y 1 (TX) no puede utilizarse al mismo tiempo.

Serial.println(data)

Imprime los datos en el puerto serie, seguido por un retorno de carro automático y salto de línea. Este comando toma la misma forma que Serial.print (), pero es más fácil para la lectura de los datos en el Monitor Serie del software.

```
Serial.println(analogValue);      // envía el valor 'analogValue' al puerto
```

Nota: Para obtener más información sobre las distintas posibilidades de Serial.println () y Serial.print () puede consultarse el sitio web de Arduino.

El siguiente ejemplo toma de una lectura analógica pin0 y envía estos datos al ordenador cada 1 segundo.

```
void setup()
    {
        Serial.begin(9600);     // configura el puerto serie a 9600bps
    }

void loop()
    {
        Serial.println(analogRead(0));  // envía valor analógico
        delay(1000);                    // espera 1 segundo
    }
```

Serial.printnl(data, data type)

Vuelca o envía un número o una cadena de carateres al puerto serie, seguido de un caracter de retorno de carro "CR" (ASCII 13, or '\r')y un caracter de salto de línea "LF"(ASCII 10, or '\n'). Toma la misma forma que el comando Serial.print()

Serial.println(b) vuelca o envía el valor de b como un número decimal en caracteres ASCII seguido de "CR" y "LF".

Serial.println(b, DEC) vuelca o envía el valor de b como un número decimal en caracteres ASCII seguido de "CR" y "LF".

Serial.println(b, HEX) vuelca o envía el valor de b como un número hexdecimal en caracteres ASCII seguido de "CR" y "LF".

Serial.println(b, OCT) vuelca o envía el valor de b como un número Octal en caracteres ASCII seguido de "CR" y "LF".

Serial.println(b, BIN) vuelca o envía el valor de b como un número binario en caracteres ASCII seguido de "CR" y "LF".

Serial.print(b, BYTE) vuelca o envía el valor de b como un byteseguido de "CR" y "LF".

Serial.println(str) vuelca o envía la cadena de caracteres como una cadena ASCII seguido de "CR" y "LF".

Serial.println() sólo vuelca o envía "CR" y "LF". Equivaldría a printNewline().

Serial.print(data, data type)

Vuelca o envía un número o una cadena de carateres, al puerto serie. Dicho comando puede tomar diferentes formas, dependiendo de los parámetros que utilicemos para definir el formato de volcado de los números.

Parámetros

data: el número o la cadena de caracteres a volcar o enviar.

data type: determina el formato de salida de los valores numéricos (decimal, octal, binario, etc...) **DEC, OCT, BIN, HEX, BYTE , si no se pe nada vuelva ASCII**

Ejemplos

Serial.print(b)

Vuelca o envía el valor de b como un número decimal en caracteres ASCII. Equivaldría a printInteger().

int b = 79; Serial.print(b); // *prints the string "79".*

Serial.print(b, DEC)

Vuelca o envía el valor de b como un número decimal en caracteres ASCII.

Equivaldría a printInteger().

int b = 79;
Serial.print(b, DEC); // *prints the string "79".*

Serial.print(b, HEX)

Vuelca o envía el valor de b como un número hexdecimal en caracteres ASCII. Equivaldría a printHex();

```
int b = 79;
Serial.print(b, HEX); // prints the string "4F".
```

Serial.print(b, OCT)

Vuelca o envía el valor de b como un número Octal en caracteres ASCII. Equivaldría a printOctal();

```
int b = 79;
Serial.print(b, OCT); // prints the string "117".
```

Serial.print(b, BIN)

Vuelca o envía el valor de b como un número binario en caracteres ASCII. Equivaldría a printBinary();

```
int b = 79;
Serial.print(b, BIN); // prints the string "1001111".
```

Serial.print(b, BYTE)

Vuelca o envía el valor de b como un byte. Equivaldría a printByte();

```
int b = 79;
Serial.print(b, BYTE);  // Devuelve el caracter "O", el cual representa el
caracter ASCII del valor 79. (Ver tabla ASCII ).
```

Serial.print(str)

Vuelca o envía la cadena de caracteres como una cadena ASCII. Equivaldría a printString().

```
Serial.print("Hello World!"); // vuelca "Hello World!".
```

Serial.avaible()

int Serial.available()

Obtiene un número entero con el número de bytes (caracteres) diponibles para leer o capturar desde el puerto serie. Equivaldría a la función serialAvailable().

Devuelve Un entero con el número de bytes disponibles para leer desde el buffer serie, o 0 si no hay ninguno. Si hay algún dato disponible, SerialAvailable() será mayor que 0. El buffer serie puede almacenar como máximo 64 bytes.

Ejemplo

```
int incomingByte = 0; // almacena el dato serie
void setup() {
    Serial.begin(9600); // abre el puerto serie, y le asigna la velocidad de 9600
bps
}
void loop() {
    // envía datos sólo si los recibe:
    if (Serial.available() > 0) {
                // lee el byte de entrada:
                incomingByte = Serial.read();
            //lo vuelca a pantalla
                Serial.print("I received: ");
                Serial.println(incomingByte, DEC);
    }
}
```

Serial.Read()

int Serial.Read()

Lee o captura un byte (un caracter) desde el puerto serie. Equivaldría a la función serialRead().

Devuelve :El siguiente byte (carácter) desde el puerto serie, o -1 si no hay ninguno.

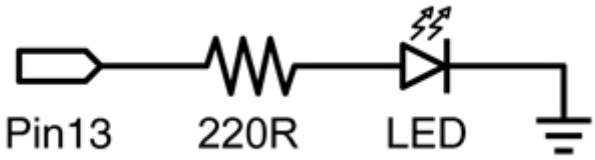
Ejemplo

```
int incomingByte = 0; // almacenar el dato serie
void setup() {
    Serial.begin(9600); // abre el puerto serie,y le asigna la velocidad de 9600
bps
}
void loop() {
    // envía datos sólo si los recibe:
    if (Serial.available() > 0) {
                // lee el byte de entrada:
                incomingByte = Serial.read();
            //lo vuelca a pantalla
                Serial.print("I received: ");
                Serial.println(incomingByte, DEC);
    }
}
```

Apendices

Formas de Conexionado de entradas y salidas

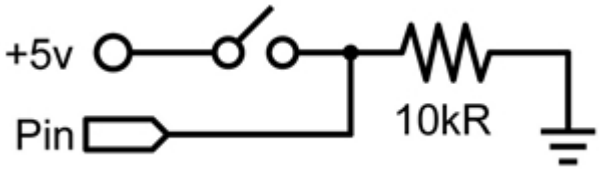
salida digital



Éste es el ejemplo básico equivalente al "hola mundo" de cualquier lenguaje de programación haciendo simplemente el encendido y apagado de un led. En este ejemplo el LED está conectado en el pin13, y se enciende y apaga "parpadea" cada segundo. La resistencia que se debe colocar en serie con el led en este caso puede omitirse ya que el pin13 de Arduino ya incluye en la tarjeta esta resistencia,

```
int ledPin = 13; // LED en el pin digital 13

void setup()      // configura el pin de salida

{
      pinMode(ledPin, OUTPUT); // configura el pin 13 como
salida
}
      void loop()       // inicia el bucle del programa
{
      digitalWrite(ledPin, HIGH); // activa el LED
      delay(1000); // espera 1 segundo
      digitalWrite(ledPin, LOW); // desactiva el LED

      delay(1000); // espera 1 segundo
}
```

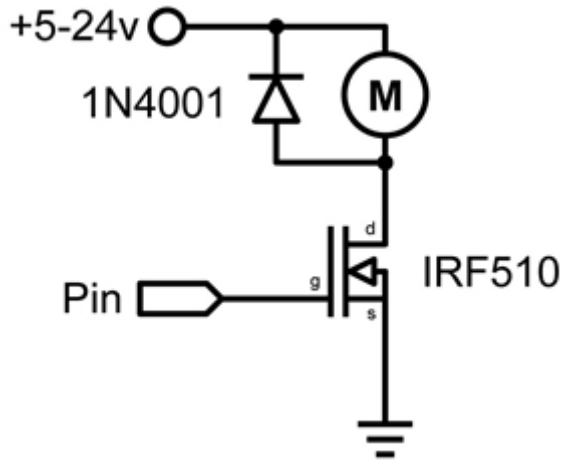
entrada digital



Ésta es la forma más sencilla de entrada con sólo dos posibles estados: encendido o apagado. En este ejemplo se lee un simple switch o pulsador conectado a PIN2. Cuando el interruptor está cerrado el pin de entrada se lee ALTO y encenderá un LED colocado en el PIN13

```
int ledPin = 13; // pin 13 asignado para el LED de salida
int inPin = 2; // pin 2 asignado para el pulsador

void setup() // Configura entradas y salidas
      {
            pinMode(ledPin, OUTPUT); // declara LED como salida
            pinMode(inPin, INPUT); // declara pulsador como entrada
      }

void loop()
      {
            if (digitalRead(inPin) == HIGH) // testea si la entrada esta activa HIGH
      {
            digitalWrite(ledPin, HIGH); // enciende el LED
            delay(1000); // espera 1 segundo
            digitalWrite(ledPin, LOW); // apaga el LED
            }
      }
```

salida de alta corriente de consumo



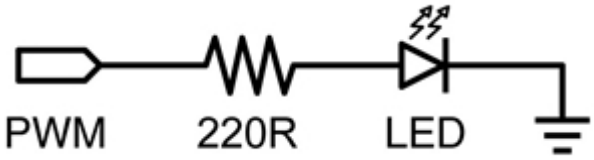
A veces es necesario controlar cargas de más de los 40 mA que es capaz de suministrar la tarjeta Arduino. En este caso se hace uso de un transistor MOSFET que puede alimentar cargas de mayor consumo de corriente. El siguiente ejemplo muestra como el transistor MOSFET conmuta 5 veces cada segundo.

Nota: El esquema muestra un motor con un diodo de protección por ser una carga inductiva. En los casos que las cargas no sean inductivas no será necesario colocar el diodo.

```
int  outPin = 5;   // pin de salida para el MOSFET

void  setup()
      {
            pinMode(outPin, OUTPUT);  // pin5 como salida
      }

void  loop()
      {
            for  (int  i=0;  i<=5;  i++)     // repetir bucle 5 veces
            {
            digitalWrite(outPin,  HIGH);  //  activa el MOSFET
            delay(250); //  espera 1/4  segundo
            digitalWrite(outPin,  LOW); //  desactiva el MOSFET
            delay(250); // espera 1/4  segundo
            }
            delay(1000); //  espera 1 segundo

      }
```

salida analógica del tipo pwm

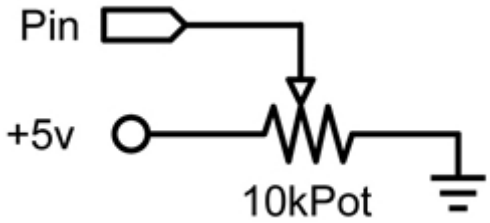
PWM (modulación de impulsos en frecuncia)



La Modulación de Impulsos en Frecuencia (PWM) es una forma de conseguir una "falsa" salida analógica. Esto podría ser utilizado para modificar el brillo de un LED o controlar un servo motor. El siguiente ejemplo lentamente hace que el LED se ilumine y se apague haciendo uso de dos bucles.

```
int ledPin = 9;  // pin PWM para el LED

void setup(){} // no es necesario configurar nada

void loop()
  {
    for (int i=0; i<=255; i++) // el valor de i asciende
    {
        analogWrite(ledPin, i); // se escribe el valor de I en el PIN de salida del LED
        delay(100); // pauses for 100ms
    }
    for (int i=255; i>=0; i--) // el valor de I desciendei
    {
        analogWrite(ledPin, i); // se escribe el valor de ii
        delay(100); // pasusa durante 100ms
    }
  }
```

entrada con potenciómetro
(entrada analógica)



El uso de un potenciómetro y uno de los pines de entrada analógica-digital de Arduino (ADC) permite leer valores analógicos que se convertirán en valores dentro del rango de 0-1024. El siguiente ejemplo utiliza un potenciómetro para controlar un el tiempo de parpadeo de un LED.

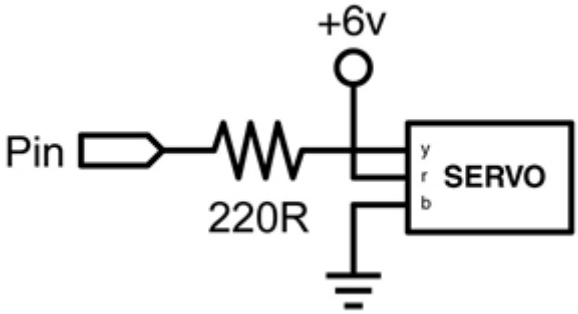
```
int potPin = 0; // pin entrada para potenciómetro
int ledPin = 13;       // pin de salida para el LED

void setup()
      {
            pinMode(ledPin, OUTPUT);      // declara ledPin como SALIDA
      }

void loop()
      {
            digitalWrite(ledPin, HIGH); // pone ledPin en on
            delay(analogRead(potPin)); // detiene la ejecución un tiempo "potPin"
            digitalWrite(ledPin, LOW); // pone ledPin en off
            delay(analogRead(potPin)); // detiene la ejecución un tiempo "potPin"

}
```

entrada conectada a resistencia variable
(entrada analógica)

Las resistencias variables como los sensores de luz LCD los termistores, sensores de esfuerzos, etc, se conectan a las entradas analógicas para recoger valores de parámetros físicos. Este ejemplo hace uso de una función para leer el valor analógico y establecer un tiempo de retardo. Este tiempo controla el brillo de un diodo LED conectado en la salida.

```
int ledPin =      9;  // Salida analógica PWM para conectar a LED
int analogPin =0;  // resistencia variable conectada a la entrada analógica pin 0

void setup(){}   // no es necesario configurar entradas y salidas

void loop()
{
      for (int i=0; i<=255; i++) // incremento de valor de i
      {
      analogWrite(ledPin, i); // configura el nivel brillo con el valor de i
      delay(delayVal()); // espera un tiempo
      }
      for (int i=255; i>=0; i--) // decrementa el valor de i
      {
      analogWrite(ledPin, i); // configura el nivel de brillo con el valor de i
      delay(delayVal()); // espera un tiempo
      }
}
int delayVal() // Método para recoger el tiempo de retardo
{
      int v; // crea una variable temporal (local)
      v      = analogRead(analogPin); // lee valor analógico
      v /= 8;     // convierte el valor leído de 0-1024 a 0-128
      return v; // devuelve el valor v
}
```

salida conectada a servo



Los servos de los juguetes tienen un tipo de motor que se puede mover en un arco de 180 º y contienen la electrónica necesaria para ello. Todo lo que se necesita es un pulso enviado cada 20ms. Este ejemplo utiliza la función servoPulse para mover el servo de 10º a 170 º.

```
int servoPin = 2; // servo conectado al pin digital 2
int myAngle; // ángulo del servo de 0-180
int pulseWidth; // anchura del pulso para la función servoPulse

void setup()
{
    pinMode(servoPin, OUTPUT);  // configura pin 2 como salida
}
void servoPulse(int servoPin, int myAngle)
{
    pulseWidth = (myAngle * 10) + 600; // determina retardo
    digitalWrite(servoPin, HIGH); // activa el servo
    delayMicroseconds(pulseWidth); // pausa
    digitalWrite(servoPin, LOW); // desactiva el servo
    delay(20); // retardo de refresco
}
void loop()
{
// el servo inicia su recorrido en 10º y gira hasta 170º
for (myAngle=10; myAngle<=170; myAngle++)
    {
        servoPulse(servoPin, myAngle);
    }
// el servo vuelve desde 170º hasta 10º
for (myAngle=170; myAngle>=10; myAngle--)
    {
        servoPulse(servoPin, myAngle);
    }
}
```

Arduino: Manual de Programación

Como escribir una librería para Arduino

Este documento explica cómo crear una librería para Arduino. Se comienza con un programa que realiza, mediante encendido y apagado de un led, el código morse y se explica cómo convertir este en una función de librería. Esto permite a otras personas utilizar fácilmente el código que has escrito cargándolo de una forma sencilla.

Se comienza con el programa de un sencillo código Morse:

La palabra a generar es SOS (. . . - - - . . .)

```
// Genera SOS en código Morse luminoso

int pin = 13;

void setup()
{
  pinMode(pin, OUTPUT);
}

void loop()  //Programa principal que gerera ". . . "- - -" y ". . . "
{
  dot(); dot(); dot(); //Genera la S (. . . )
  dash(); dash(); dash(); // Genera la O (- - -)
  dot(); dot(); dot(); // Genera la S (. . . )
  delay(3000); //Espera un tiempo
}

void dot()   //Procedimiento para generar un punto
{
  digitalWrite(pin, HIGH);
  delay(250);
  digitalWrite(pin, LOW);
  delay(250);
}

void dash()          //Procedimiento para generar una raya
{
  digitalWrite(pin, HIGH);
  delay(1000);
  digitalWrite(pin, LOW);
  delay(250);
}
```

Si se ejecuta este programa, se ejecuta el código SOS (llamada de solicitud de auxilio) en la salida PIN13.

El programa tiene distintas partes que tendremos que poner en nuestra librería. En primer lugar, por supuesto, tenemos las funciones ***dot()*** (punto) y ***dash()*** (raya) que se encargar de que el LED parpadeé de manera corta o larga respectivamente. En segundo lugar, tenemos la instrucción *ledPin* que utilizamos para determinar el pin a utilizar. Por último, está la llamada a la función ***pinMode ()*** que inicializa el pin como salida.

Vamos a empezar a convertir el programa en una librería.

Usted necesita por lo menos dos archivos en una librería: un archivo de cabecera (w / la extensión. H) y el archivo fuente (w / extensión. CPP). El fichero de cabecera tiene definiciones para la librería: básicamente una lista de todo lo que contiene, mientras que el archivo fuente tiene el código real. Vamos a llamar a nuestra biblioteca "Morse", por lo que nuestro fichero de cabecera se **Morse.h**. Echemos un vistazo a lo que sucede en ella. Puede parecer un poco extraño al principio, pero lo entenderá una vez que vea el archivo de origen que va con ella.

El núcleo del archivo de cabecera consiste en una línea para cada función en la biblioteca, envuelto en una clase junto con las variables que usted necesita:

```
class Morse
{
  public:
    Morse(int pin);
    void dot();
    void dash();
  private:
    int _pin;
};
```

Una clase es simplemente una colección de funciones y variables que se mantienen unidos todos en un solo lugar. Estas funciones y variables pueden ser públicos, lo que significa que puede ser utilizadas por quienes utilizan la librería, o privadas, lo que significa que sólo se puede acceder desde dentro de la propia clase. Cada clase tiene una función especial conocida como un **constructor**, que se utiliza para crear una instancia de la clase. El constructor tiene el mismo nombre que la clase, y no devuelve nada.

Usted necesita dos cosas más en el fichero de cabecera. Uno de ellos es un **# include** declaración que le da acceso a los tipos estándar y las constantes del lenguaje de Arduino (esto se añade automáticamente en todos los programas que hacemos con Arduino, pero no a las librerías). Por lo que debemos incluirlas (poniéndolas por encima de la definición de clase dada anteriormente):

```
# include "WConstants.h"
```

Por último, se colocara delante del código la cabecera siguiente:

```
# ifndef Morse_h
```

```
#define Morse_h

// el estamento #include y el resto del código va aquí..

#endif
```

Básicamente, esto evita problemas si alguien accidentalmente pone # include en la librería dos veces.

Por último, por lo general, se pone un comentario en la parte superior de la librería con su nombre, una breve descripción de lo que hace, quien la escribió, la fecha y la licencia.

Echemos un vistazo a la cabecera completa disposición del fichero de cabecera h:

Fichero Morse.h

```
/*
  Morse.h - Library for flashing Morse code.
  Created by David A. Mellis, November 2, 2007.
  Released into the public domain.
*/
#ifndef Morse_h
#define Morse_h
#include "WConstants.h"

class Morse
{
  public:
    Morse(int pin);
    void dot();
    void dash();
  private:
    int _pin;
};

#endif
```

Ahora vamos a escribir las diversas partes del archivo fuente de la librería, **Morse.cpp**.

Primero se ponen un par de declaraciones mediante "# include". Estas incluyen resto del código de acceso a las funciones estándar de Arduino, ya que en las definiciones figuran en el archivo de cabecera:

```
#include "WProgram.h"
#include "Morse.h"
```

Luego viene el **constructor**. Ahora se indicará lo que debería suceder cuando alguien crea una instancia a la clase. En este caso, el usuario especifica el pin que les gustaría utilizar. Configuramos el pin como salida guardarlo en una variable privada para su uso en las otras funciones:

```
Morse::Morse(int pin)
{
  pinMode(pin, OUTPUT);
  _pin = pin;
}
```

Hay un par de cosas extrañas en este código. El primero es el **Morse::** antes del nombre de la función. Esto indica que la función es parte de la clase **Morse**. Verá este de nuevo en las otras funciones en la clase. La segunda cosa inusual es el guión bajo en el nombre de nuestra variable privada, **_pin**. Esta variable puede tener cualquier nombre que desee, siempre y cuando coincida con la definición que figura en el fichero de cabecera. La adición de un guión bajo al comienzo del nombre es una convención para dejar claro que las variables son privadas, y también a distinguir el nombre de la del argumento a la función (**pin** en este caso).

Después viene el código del programa que queremos convertir en una función (¡por fin!). Parece casi igual, excepto con **Morse::** delante de los nombres de las funciones, y **_pin** en lugar de **pin**:

```
void Morse::dot()
{
  digitalWrite(_pin, HIGH);
  delay(250);
  digitalWrite(_pin, LOW);
  delay(250);
}

void Morse::dash()
{
  digitalWrite(_pin, HIGH);
  delay(1000);
  digitalWrite(_pin, LOW);
  delay(250);
}
```

Por último, es típico incluir el comentario de cabecera en la parte superior de la fuente así como el archivo. Vamos a ver el fichero completo:

Fichero Morse.cpp

```
/*
  Morse.cpp - Library for flashing Morse code.
  Created by David A. Mellis, November 2, 2007.
  Released into the public domain.
*/

#include "WProgram.h"
#include "Morse.h"

Morse::Morse(int pin)
{
   pinMode(pin, OUTPUT);
   _pin = pin;
}

void Morse::dot()
{
   digitalWrite(_pin, HIGH);
   delay(250);
   digitalWrite(_pin, LOW);
   delay(250);
}

void Morse::dash()
{
   digitalWrite(_pin, HIGH);
   delay(1000);
   digitalWrite(_pin, LOW);
   delay(250);
}
```

Y eso es todo lo que necesita (hay algunas otras cosas opcionales, pero vamos a hablar de eso más adelante).

Ahora vamos a ver cómo se utiliza la librería.

En primer lugar, debemos crear una carpeta llamada **Morse** dentro del subdirectorio **hardware/libraries** de la aplicación Arduino. Copiar o mover los archivos **Morse.h** y **Morse.cpp** en esa carpeta. Ahora lanzar la aplicación Arduino. Cuando se inicia, compilará la recién creada librería, generando un fichero objeto (**Morse.o**) y mostrando cualquier tipo de advertencias o errores. Si usted abre el menú **Sketch> Import Library**, usted deberá ver el interior el fichero objeto Morse. Como usted trabaja con su librería, tendrá que borrar el archivo Morse.o y relanzar Arduino (o elegir una nueva tarjeta en el menú **Tools>Boards**) para recompilar su biblioteca. Si la biblioteca no se

construye, asegúrese de que están realmente los archivos CPP y. H (con y sin suplemento alguno. Pde o la extensión. Txt, por ejemplo).

Veamos como podemos escribir nuestro nuevo programa SOS haciendo uso de la nueva librería:

Programa para Arduino

```
#include <Morse.h>

Morse morse(13);

void setup()
{
}

void loop()
{
  morse.dot(); morse.dot(); morse.dot();
  morse.dash(); morse.dash(); morse.dash();
  morse.dot(); morse.dot(); morse.dot();
  delay(3000);
}
```

Hay algunas diferencias con respecto al antiguo programa (además del hecho de que algunos de los códigos se han incorporado a la librería).

En primer lugar, hemos añadido un estamento **"# include"** en la parte superior del programa. Esto hace que la librería Morse quede a disposición del programa y la incluye en el código. Esto significa que ya no necesitan una librería en el programa, usted debe borrar el # include para ahorrar espacio.

En segundo lugar, nosotros ahora podemos crear una instancia de la clase Morse llamado **morse**:

Morse morse(13);

Cuando esta línea se ejecuta (que en realidad sucede antes incluso de **setup()**), el constructor de la clase Morse será invocado y le pasara el argumento que se ha dado aquí (en este caso, sólo **13**).

Tenga en cuenta que nuestra parte **setup()** del programa está vacía, porque la llamada a **pinMode ()** se lleva a cabo en el interior de la librería (cuando la instancia se construye).

Por último, para llamar a las funciones punto **dot()** y raya **dash()**, es necesario colocar el prefijo **morse**. – delante de la instancia que queremos usar. Podríamos tener varias

instancias de la clase Morse, cada uno en su propio pin almacenados en la variable privada _pin de esa instancia. Al llamar una función en un caso particular, especificaremos qué variables del ejemplo debe utilizarse durante esa llamada a una función. Es decir, si hemos escrito:

```
Morse morse(13);
Morse morse2(12);
```

entonces dentro de una llamada a **morse2.dot (), _pin** sería 12.

Si ha escrito el nuevo programa, probablemente se habrá dado cuenta de que ninguna de nuestras funciones de la librería fue reconocida por el entorno de Arduino destacando su color. Por desgracia, el software de Arduino no puede averiguar automáticamente lo que se ha definido en su librería (a pesar de que sería una característica interesante), lo que tiene que darle un poco de ayuda. Para hacer esto, cree un archivo llamado **keywords.txt** Morse en el directorio. Debe tener un aspecto como este:

```
Morse	KEYWORD1
dash	KEYWORD2
dot	KEYWORD2
```

Cada línea tiene el nombre de la palabra clave, seguida de un código (sin espacios), seguido por el tipo de palabra clave. Las clases deben ser KEYWORD1 y son de color naranja; funciones deben ser KEYWORD2 y será de color marrón. Tendrás que reiniciar el entorno Arduino para conseguir reconocer las nuevas palabras clave.

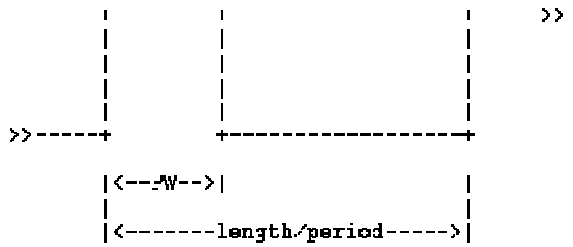
Es interesante que quienes utilicen la libreia Morse tengan algun ejemplo guardado y que aparezca en el IDE Arduino cuando seleccionamos dentro de la carpeta ejemplos (Sketch). Para hacer esto, se crea una carpeta de **ejemplos** dentro de la carpeta que contiene la librería Morse. A continuación, movemos o copiamos el directorio que contiene el programa (lo llamaremos SOS) que hemos escrito anteriormente en el directorio de ejemplos. (Usted puede encontrar el ejemplo mediante el menú **Sketch> Sketch Show Folder.)** Si reiniciamos Arduino reiniciar veremos una **Library_Morse** dentro del menú **File > Sketchbook > Examples** que contiene su ejemplo. Es posible que desee añadir algunos comentarios que explicar mejor cómo utilizar la biblioteca.

Si deseas probar la librería completa(con palabras clave y el ejemplo), puede descargarlo en: Morse.zip.

Señales analógicas de salida en Arduino (PWM).

En este apartado vamos a ver los fundamentos en los que se basa la generación de salidas analógicas en Arduino. El procedimiento para generar una señal analógica es el llamado PWM.

Señal PWM (Pulse-width modulation) señal de modulación por ancho de pulso.



Donde:

- PW (Pulse Width) o ancho de pulso, representa al ancho (en tiempo) del pulso.

- period/length (periodo), o ciclo , es el tiempo total que dura la señal.

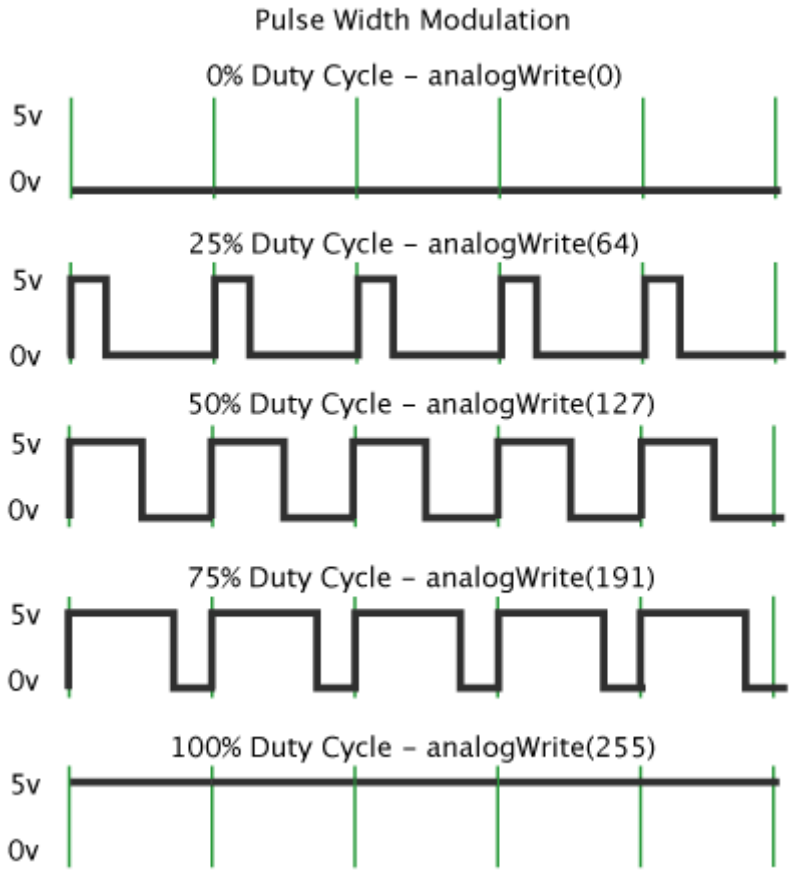
La frecuencia se define como la cantidad de pulsos (estado on/off)por segundo y su expresión matemática es la inversa del periodo, como muestra la siguiente ecuación.

$$frequency = \frac{1}{period}$$

El periodo se mide en segundos, de este modo la unidad en la cual se mide la frecuencia (hertz) es la inversa a la unidad de tiempo (segundos).

Existe otro parámetro asociado o que define a la señal PWM, denominado "Duty cycle", el cual determina el porcentaje de tiempo que el pulso (o voltaje aplicado) está en estado activo (on) durante un ciclo.

Por ejemplo, si una señal tiene un periodo de 10 ms y sus pulsos son de ancho (PW) 2ms, dicha señal tiene un duty cycle de 20% (20% on y 80% off). El siguiente gráfico muestra tres señales PWM con diferentes "duty cycles".



La señal PWM se utiliza como técnica para controlar circuitos analógicos. El periodo y la frecuencia del tren de pulsos puede determinar la potencia entregada a dicho circuito. Si, por ejemplo, tenemos un voltaje de 9v y lo modulamos con un duty cycle del 10%, obtenemos 0.9V de señal analógica de salida.

Las señales PWM son comúnmente usadas para el control de motores DC (si decrementas la frecuencia, la inercia del motor es más pequeña y el motor se mueve más lentamente), ajustar la intensidad de brillo de un LED, etc.

En Arduino la señal de salida PWM (pines 9,10) es una señal de frecuencia constante (30769 Hz) y que sólo nos permite cambiar el "duty cycle" o el tiempo que el pulso está activo (on) o inactivo (off), utilizando la función analogWrite().

Otra forma de generar señales PWM es utilizando la capacidad del microprocesador. La señal de salida obtenida de un microprocesador es una señal digital de 0 voltios (LOW) y de 5 voltios (HIGH).

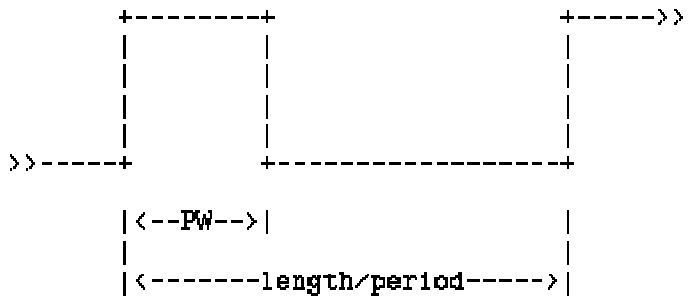
Con el siguiente código y con sólo realizar modificaciones en los intervalos de tiempo que el pin seleccionado tenga valor HIGH o LOW, a través de la función digitalWrite (), generamos la señal PWM.

```
/* señal PWM */

int digPin = 10; // pin digital 10

void setup() {
```

```
    pinMode(digPin, OUTPUT);    // pin en modo salida
}

void loop() {
    digitalWrite(digPin, HIGH); // asigna el valor HIGH al pin
    delay(500);                 // espera medio segundo
    digitalWrite(digPin, LOW);  // asigna el valor LOW al pin
    delay(500);                 // espera medio segundo
}
```

El programa pone el pin a HIGH una vez por segundo, la frecuencia que se genera en dicho pin es de 1 pulso por segundo o 1 Hertz de pulso de frecuencia (periodo de 1 segundo) . Cambiando la temporización del programa, podremos cambiar la frecuencia de la señal. Por ejemplo, si cambiamos las dos líneas con delay(500) a delay(250), multiplicaremos la frecuencia por dos, de forma que estamos enviando el doble de la cantidad de pulsos por segundo que antes.

Calculo de tonos:



Donde:

Frecuencia-tono=1/length-Periodo

Si "duty cycle"=50%, es decir, el ancho de los pulsos activos (on) e inactivos (off) son iguales---> Periodo=2*PW

Obteniendo la siguiente fórmula matemática:

PW o ancho de pulso = 1/(2 * toneFrequency) = period / 2

De forma que a una frecuencia o periodo dados, podemos obtener la siguiente tabla:

Nota musical	Frecuencia-tono	Periodo (us)	PW (us)
c	261 Hz	3830	1915
d	294 Hz	3400	1700
e	329 Hz	3038	1519
f	349 Hz	2864	1432
g	392 Hz	2550	1275
a	440 Hz	2272	1136
b	493 Hz	2028	1014
C	523 Hz	1912	956

(cleft) 2005 D. Cuartielles for K3

Con Arduino, tenemos dos formas de generar tonos. Con el primer ejemplo construiremos y enviaremos una señal cuadrada de salida al piezo, mientras que con el segundo haremos uso de la señal de modulación por ancho de pulso o PWM de salida en Arduino.

Ejemplo 1:

> */*Con el siguiente código y con sólo realizar modificaciones en los intervalos de tiempo que el pin seleccionado tenga valor HIGH o LOW, a través de la función*

```
digitalWrite (), generamos la señal PWM a una determinada frecuencia de
salida=261Hz*/

int digPin = 10; // pin digital 10

int PW=1915; // valor que determina el tiempo que el pulso va a estar en on/off

void setup() {
   pinMode(digPin, OUTPUT);    // pin digital en modo salida
}

void loop() {
   delayMicroseconds(PW);          // espera el valor de PW
   digitalWrite(digPin, LOW);  // asigna el valor LOW al pin
   delayMicroseconds(PW);          // espera el valor de PW
   digitalWrite(digPin, HIGH); // asigna el valor HIGH al pin
}
```

Ejemplo 2:

En Arduino la señal de salida PWM (pines 9,10) es una señal de frecuencia constante (30769 Hz) y que sólo nos permite cambiar el "duty cycle" o el tiempo que el pulso está activo (on) o inactivo (off), utilizando la función analogWrite().

Usaremos la característica "Pulse Width" con "analogWrite" para cambiar el volumen.

```
analogWrite(, value)
```

value: representa al parámetro "duty cycle" (ver PWM) y puede tomar valores entre 0 y 255.

0 corresponde a una señal de salida de valor constante de 0 v (LOW) o 0% de "duty cycle";

255 es una señal de salida de valor constante de 5 v (HIGH) o 100% de "duty cycle"; .

Para valores intermedios, el pin rápidamente alterna entre 0 y 5 voltios - el valor más alto, lo usual es que el pin esté en high (5 voltios).

La frecuencia de la señal PWM es constante y aproximadamente de 30769 Hz.

```
int speakerOut = 9; int volume = 300; // máximo volume es 1000 ¿?
 int PW=1915;
void loop() {
analogWrite(speakerOut, 0);
analogWrite(speakerOut,volume);
delayMicroseconds(PW);
analogWrite(speakerOut, 0);
delayMicroseconds(PW); }
```

Arduino: Manual de Programación

Comunicando Arduino con otros sistemas

Hoy en día la manera más común de comunicación entre dispositivos electrónicos es la comunicación serial y Arduino no es la excepción. A través de este tipo de comunicación podremos enviar datos a y desde nuestro Arduino a otros microcontroladores o a un computador corriendo alguna plataforma de medios (Processing, PD, Flash, Director, VVVV, etc.). En otras palabras conectar el comportamiento del sonido o el video a sensores o actuadores. Explicaré aquí brevemente los elementos básicos de esta técnica:

Funciones básicas

El mismo cable con el que programamos el Arduino desde un computador es un cable de comunicación serial. Para que su función se extienda a la comunicación durante el tiempo de ejecución, lo primero es abrir ese puerto serial en el programa que descargamos a Arduino. Para ello utilizamos la función

```
beginSerial(19200);
```

Ya que solo necesitamos correr esta orden una vez, normalmente iría en el bloque void setup(). El número que va entre paréntesis es la velocidad de transmisión y en comunicación serial este valor es muy importante ya que todos los dispositivos que van a comunicarse deben tener la misma velocidad para poder entenderse. 19200 es un valor estándar y es el que tienen por defecto Arduino al iniciar.

Una vez abierto el puerto lo más seguro es que luego queramos enviar al computador los datos que vamos a estar leyendo de uno o varios sensores. La función que envía un dato es

```
Serial.print(data);
```

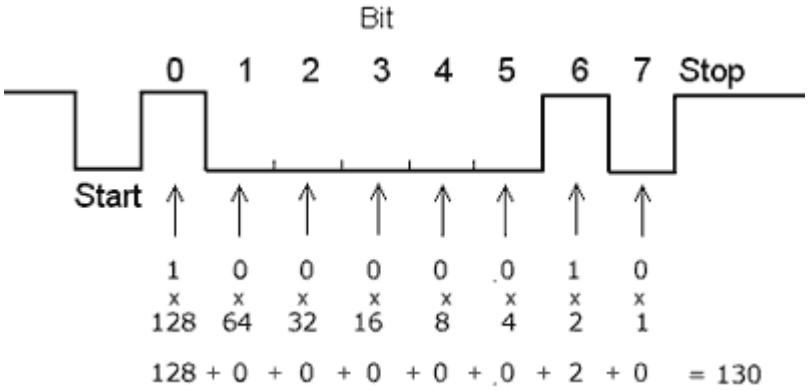
Una mirada en la referencia de Arduino permitirá constatar que las funciones print y println (lo mismo que la anterior pero con salto de renglón) tienen opcionalmente un modificador que puede ser de varios tipos:

```
Serial.print(data, DEC);   // decimal en ASCII
Serial.print(data, HEX);   // hexadecimal en ASCII
Serial.print(data, OCT);   // octal en ASCII
Serial.print(data, BIN);   // binario en ASCII
Serial.print(data, BYTE);  // un Byte
```

Como puede verse, prácticamente todos los modificadores, menos uno, envían mensajes en ASCII. Explicaré brevemente:

Series de pulsos

En el modo más sencillo y común de comunicación serial (asincrónica, 8 bits, más un bit de parada) siempre se está enviando un byte, es decir un tren de 8 pulsos de voltaje legible por la máquina como una serie de 8, 1s ó 0s:



O sea que no importa cual modificador usemos siempre se están enviando bytes. La diferencia esta en lo que esos bytes van a representar y sólo hay dos opciones en el caso del Arduino: una serie de caracteres ASCII o un número.

Si Arduino lee en un sensor analógico un valor de 65, equivalente a la serie binaria 01000001 esta será enviada, según el modificador, como:

dato	Modificador	Envío (pulsos)
65	---DEC----	(“6” y “5” ACIIs 54-55) 000110110-000110111
65	---HEX----	(“4” Y “1” ACIIs 52-49) 000110100-000110001
65	---OCT----	(“1”, “0” y “1” ACIIs 49-48-49) 000110001-000110000-000110001
65	---BIN----	(“0”,”1”,”0”,”0”,”0”,”0”,”0”y”1” ACIIs 49-48-49-49-49-49-49-48) 000110000-…
65	---BYTE---	01000001

No explicaremos conversiones entre los diferentes sistemas de representación numérica, ni la tabla ASCII (google), pero es evidente como el modificador BYTE permite el envío de información más económica (menos pulsos para la misma cantidad de información) lo que implica mayor velocidad en la comunicación. Y ya que esto es importante cuando se piensa en interacción en tiempo real es el modo que usaremos acá.

Un ejemplo sencillo

Enviar un sólo dato es realmente fácil. En el típico caso de un potenciómetro conectado al pin 24 del ATmega:

```
int potPin = 2;
int ledPin = 13;
int val = 0;

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);  //activamos  el pin para saber cuando arranco
}

void loop() {
```

```
  val = analogRead(potPin);    // lee el valor del Pot
  Serial.println(val);
}
```

Si no utilizamos ningún modificador para el Serial.println es lo mismo que si utilizáramos el modificador DEC. Así que no estamos utilizando el modo más eficiente pero si el más fácil de leer en el mismo Arduino. Al correr este programa podremos inmediatamente abrir el monitor serial del software Arduino (último botón a la derecha) y aparecerá el dato leído en el potenciómetro tal como si usáramos el println en Processing.

Envío a Processing (versión ultra simple)

Para enviar este mismo dato a Processing si nos interesa utilizar el modo BYTE así que el programa en Arduino quedaría así:

```
int potPin = 2;
int ledPin = 13;
int val = 0;

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);  // activamos el pin para saber cuando arranco
}

void loop() {
;    // lee el Pot y lo divide entre 4 para quedar entre 0-255
  val = analogRead(potPin)/4
  Serial.print(val, BYTE);
}
```

En Processing tenemos que crear un código que lea este dato y haga algo con él:

```
import processing.serial.*;

Serial puerto;// Variable para el puerto serial
byte pot;// valor entrante
int PosX;

void setup() {
  size(400, 256);
  println(Serial.list()); // lista los puertos seriales disponibles
//abre el primero de esa lista con velocidad 9600
```

```
  port = new Serial(this, Serial.list()[0], 9600);
  fill(255,255,0);
  PosX = 0;
  pot = 0;
}

void draw() {
  if (puerto.available() > 0) { // si hay algún dato disponible en el puerto
    pot = puerto.read();// lo obtiene
    println(pot);
  }
  ellipse(PosX, pot, 3, 3); // y lo usa
  if (PosX < width) {
    PosX++;
  } else {
    fill(int(random(255)),int(random(255)),int(random(255)));
    PosX = 0;
  }
}
```

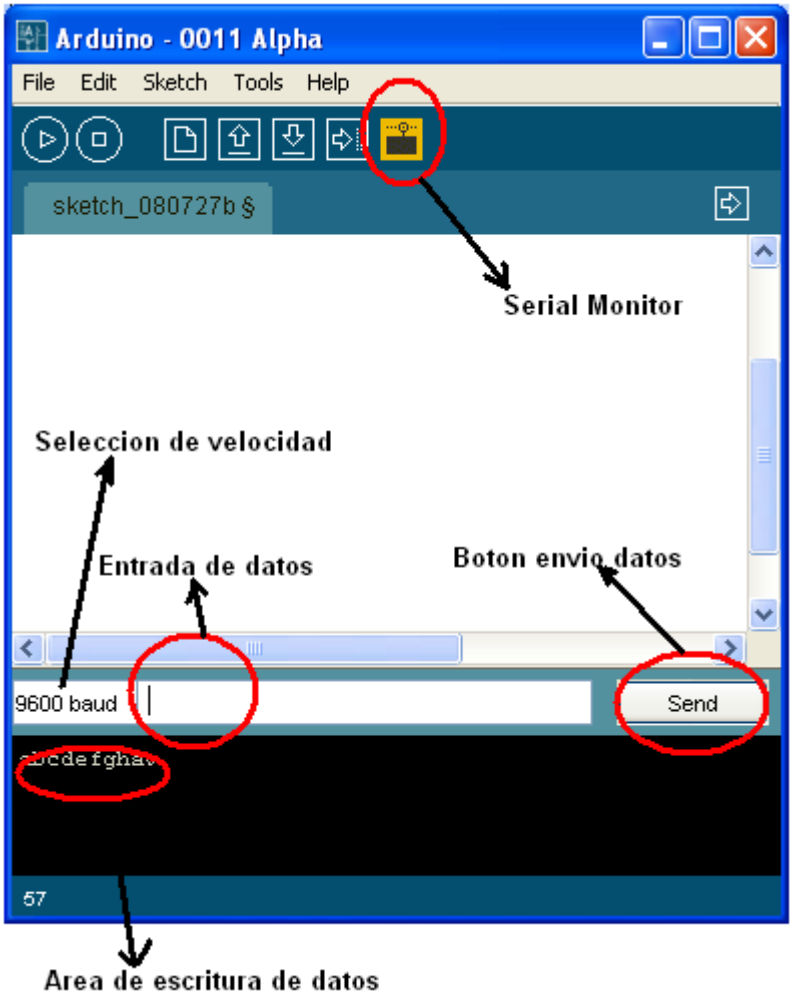
Si ya se animó a intentar usar más de un sensor notará que no es tan fácil como duplicar algunas líneas.

Comunicación vía puerto Serie:

La tarjeta Arduino puede establecer comunicación serie (recibir y enviar valores codificados en ASCII) con un dispositivo externo, a través de una conexión por un cable/puerto USB (tarjeta USB) o cable/puerto serie RS-232(tarjeta serie) (Enlace)

Igual que para la descarga de los programas, sólo será necesario indicar el número de puerto de comunicaciones que estamos utilizando y la velocidad de transferencia en baudios (enlace).También hay que tener en cuenta las limitaciones de la transmisión en la comunicación serie, que sólo se realiza a través de valores con una longitud de 8-bits (1 Byte)(Ver serialWrite(c) o serialRead(c)), mientras que como ya se hemos indicado, el A/D (Convertidor) de Arduino tiene una resolución de 10-bits.(enlace)

Dentro del interfaz Arduino, disponemos de la opción "Monitorización de Puerto Serie", que posibilita la visualización de datos procedentes de la tarjeta.



Para definir la velocidad de transferencia de datos, hay que ir al menú "Herramientas" y seleccionar la etiqueta "Velocidad de monitor Serie". La velocidad seleccionada, debe coincidir con el valor que hemos determinado o definido en nuestro programa y a través

del comando beginSerial().Dicha velocidad es independiente de la velocidad definida para la descarga de los programas.

La opción de "Monitorización de puerto serie" dentro del entorno Arduino, sólo admite datos procedentes de la tarjeta. Si queremos enviar datos a la tarjeta, tendremos que utilizar otros programas de monitorización de datos de puerto serie como HyperTerminal (para Windows) -Enlace o ZTerm (para Mac)-XXXX- Linux-Enlace, etc.

También se pueden utilizar otros programas para enviar y recibir valores ASCII o establecer una comunicación con Arduino: Processing (enlace), Pure Data (enlace), Director(enlace), la combinación o paquete serial proxy + Flash (enlace), MaxMSP (enlace), etc.

Nota: Hay que dejar tiempos de espera entre los envíos de datos para ambos sentidos, ya que se puede saturar o colapsar la transmisión. ¿?

Envío de datos desde Arduino(Arduino->PC) al PC por puerto de comunicación serie:

Ejercicio de volcado de medidas o valores obtenidos de un sensor analógico

Código

```
/* Lectura de una entrada analógica en el PC
 El programa lee una entrada analógica, la divide por 4
 para convertirla en un rango entre 0 y 255, y envía el valor al PC en
 diferentes formatos ASCCI.
 A0/PC5: potenciómetro conectado al pin analógico 1 y puerto de PC-5
 Created by Tom Igoe 6 Oct. 2005
 Updated
 */

int val; // variable para capturar el valor del sensor analógico

void setup() {
  // define la velocidad de transferencia a 9600 bps (baudios)
  beginSerial(9600);
}

void loop() {
  // captura la entrada analógica, la divide por 4 para hacer el rango de 0-255
  val = analogRead(0)/4;
// texto de cabecera para separar cada lectura:
    printString("Valor Analogico =");
```

```
// obtenemos un valor codificado en ASCII (1 Byte) en formato decimal :
   printInteger(val);
   printString("\t");  //Carácter espacio
// obtenemos un valor codificado en ASCII (1 Byte) en formato hexadecimal :
   printHex(val);
   printString("\t");
// obtenemos un valor codificado en ASCII (1 Byte) en formato binario
  printBinary(val);
  printString("\t");
// obtenemos un valor codificado en ASCII (1 Byte)en formato octal:
   printOctal(val);
   printString("\n\r"); //caracter salto de linea y retorno de carro
// espera 10ms para la próxima lectura
  delay(10);
}
```

Otra solución puede ser la de transformar los valores capturados en un rango entre 0 y 9 y en modo de codificación ASCII o en caracteres ASCII. De forma que dispongamos de un formato más sencillo o legible, sobre la información capturada.

El siguiente código incluye una función llamada **treatValue()** que realiza dicha transformación.

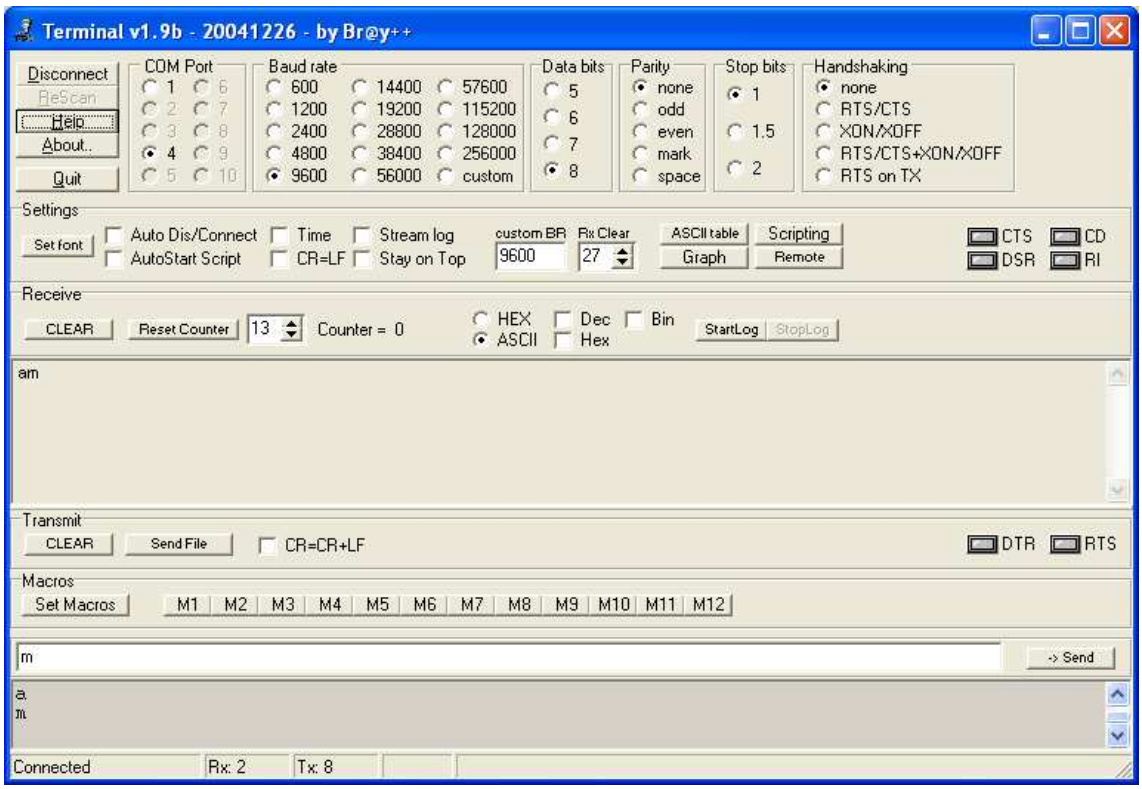
```
int val; // variable para capturar el valor del sensor analógico

void setup() {
  // define la velocidad de transferencia a 9600 bps (baudios)
  beginSerial(9600);
}

int treatValue(int data) {
  return (data * 9 / 1024) + 48; //fórmula de transformación
 }
void loop() {

val= analogRead(0); //captura del valor de sensor analógico (0-1023)

serialWrite(treatValue(val)); //volcado al puerto serie de 8-bits
```

```
serialWrite(10); //caracter de retorno de carro
serialWrite(13); //caracter de salto de línea
delay(10); }
```

```
// Serial Output // by BARRAGAN <http://people.interaction-ivrea.it/h.barragan>
int switchpin = 0; // interruptor conectado al pin 0
void setup() {
  pinMode(switchpin, INPUT);  // pin 0 como ENTRADA
  Serial.begin(9600);       // inicia el puerto serie a 9600bps
}
void loop() {
  if(digitalRead(switchpin) == HIGH)  //si el interruptor esta en ON
  {
    Serial.print(1);          // envía 1 a Processing
  }else{
    Serial.print(0);         // en caso contrario envía 0 a Processing
  }
  delay(100);                // espera 100ms
}
```

Envío de datos desde el PC (PC->Arduino) a Arduino por puerto de comunicación serie:

En primer lugar, necesitamos instalar un programa como Hyperterminal en nuestro PC, en caso de que sea Windows.....



Software Terminal para realizar comunicaciones con el puerto serie

Seleccionar el puerto que estamos utilizando con la tarjeta, la velocidad de transferencia y el formato de salida de los datos. Y finalmente conectar...

Se puede realizar una comprobación con el ejercicio mostrado arriba.

Nota: El programa de monitorización de datos está ocupando el puerto utilizado para la conexión a la tarjeta, por lo que si quieres realizar una nueva descarga del programa, tendrás que desconectarte previamente de este último.

```
/*by BARRAGAN <http://people.interaction-ivrea.it/h.barragan>
*Demuestra como leer un dato del puerto serie. Si el dato recibido es una 'H', la luz se
*enciende ON, si es una  'L', la luz se apaga OFF. Los datos provienen del PC o de un
*programa como Processing..
*created 13 May 2004 revised 28 Aug 2005
*/

char val; // variable que recibe el dato del puerto serie
int ledpin = 13; // LED conectado al pin 13
```

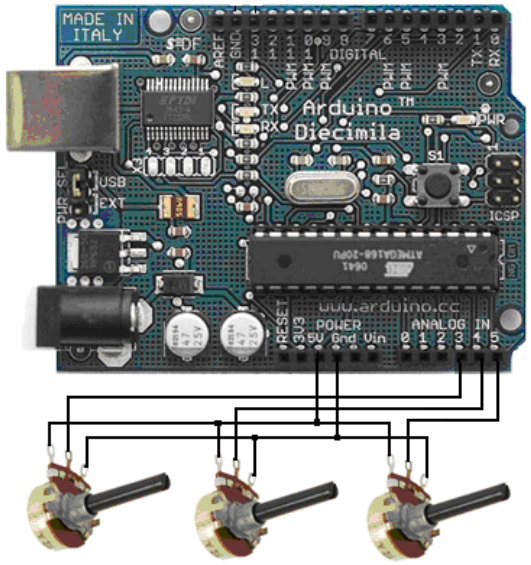
```
void setup() {
  pinMode(ledpin, OUTPUT);  // pin 13 (LED)actua como SALIDA
  Serial.begin(9600);       // inicia la comunicación con el puerto serie a 9600bps
}
void loop() {
  if( Serial.available() )      // si hay dato e el puerto lo lee
  {
    val = Serial.read();        // lee y almacena el dato en  'val'
  }
  if( val == 'H' )              //su el dato recibido es ´H´
  {
    digitalWrite(ledpin, HIGH);  //activa el LED
  } else {
    digitalWrite(ledpin, LOW);   // en caso contrario lo desactiva
  }
  delay(100);                   // espera  100ms para una nueva lectura
}
```

Para probar este programa bastará con iniciar el programa que actúe de “terminal de comunicación” Hyperterminal de Windowws o el programa mostrado anteriormente y podemos enviar los datos y comprobar como actúa.

Envío a petición (toma y dame)

Cuando se envía más de un dato del Arduino a otro sistema es necesario implementar reglas de comunicación adicionales para poder distinguir a que dato corresponde cada uno de los paquetes de bytes recibidos. Una manera simple y eficiente de hacer esto es jugando al *"toma y dame"*. Arduino no enviará los valores de los sensores hasta que Processing no le envíe también un valor por el puerto serial y Processing, a su vez, no enviara ese valor hasta no tener los datos que espera completos.

Este sería el código para Arduino usando tres potenciómetros en los últimos tres pines analógicos del ATmega:



Codigo para cargar en la tarjeta Arduino desde el IDE Arduino

```
int pot1= 0;          // valores de los sensores analógicos
int pot2= 0;
int pot3= 0;
int inByte = 0;       // valor entrante de Processing

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  if (Serial.available() > 0) {  // sólo si algo ha llegado
    inByte = Serial.read();      // lo lee
    // hace la lectura de los sensores en pines 3,4y5 (análogos)
    pot1 = analogRead(3)/4;
    pot2 = analogRead(4)/4;
    pot3 = analogRead(5)/4;
```

```
   // y los envía
   Serial.print(pot1, BYTE);
   Serial.print(pot2, BYTE);
   Serial.print(pot3, BYTE);
  }
}
```

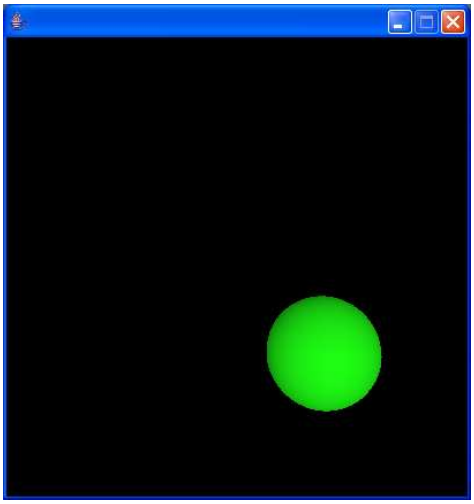
Una vez cargado este programa en la tarjeta Arduino está en disposición de enviar los datos de las lecturas de los potenciómetros cuando le sean demandados por el programa que los requiera. En nuestro ejemplo vamos a escribir un programa en el IDE Processing y será este el que se ocupe de leer los datos y con ellos modificar la posición de una bola que aparecerá en pantalla

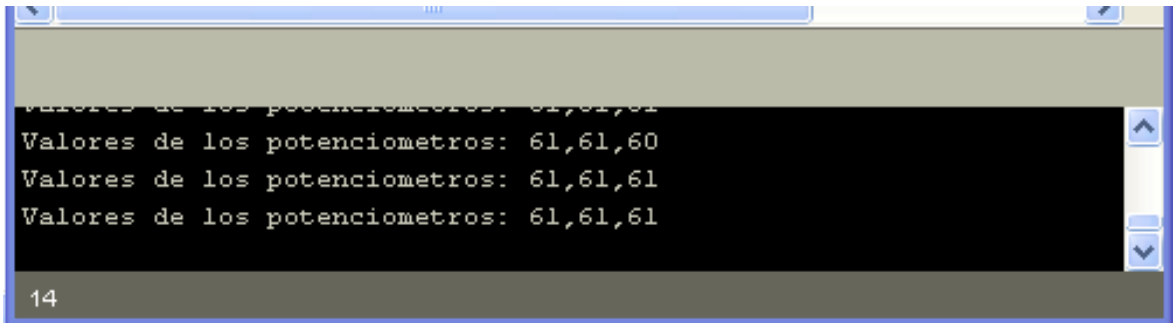
Será processing quién empezará el "toma y dame" y deberá reconocer cada dato. Este es el código:

Código para Processing

```
import processing.serial.*;

Serial puerto;
int[] datosEntrantes = new int[3];   // arreglo para recibir los tres datos
int cuantosDatos = 0;                // contador
int posX, posY, posZ;                // posición de un objeto 3D
boolean hayDatos = false;            // control de verdad

void setup() {
  size(400, 400, P3D);
  noStroke();

  println(Serial.list());// puertos serie disponibles
  puerto = new Serial(this, Serial.list()[0], 9600); // Configuración del puerto
  puerto.write(65);    // Envía el primer dato para iniciar el toma y dame
}

void draw() {
  background(0);
  lights();
  fill(30,255,20);
  translate(width/2 + posX, height/2 + posY, posZ);
  sphere(40);

  if (hayDatos == false) { //si no hay datos envía uno
    puerto.write(65);
  }
}
// esta función corre cada vez que llega un dato serial
```

```
void serialEvent(Serial puerto) {
  if (hayDatos == false) {
    hayDatos = true;    // de ahora en adelante el dato de envío se dará por el toma y
dame
  }
  // Lee el dato y lo añade al arreglo en su última casilla
  datosEntrantes[cuantosDatos] = puerto.read();
  cuantosDatos++;
  if (cuantosDatos > 2 ) { // Si ya hay tres datos en el arreglo
    posX = datosEntrantes[0];
    posY = datosEntrantes[1];
    posZ = datosEntrantes[2];
    println("Valores de los potenciometros: " + posX + "," + posY + "," + posZ);
    puerto.write(65); // y envía para pedir más
    cuantosDatos = 0; // y todo empieza de nuevo
  }
}
```

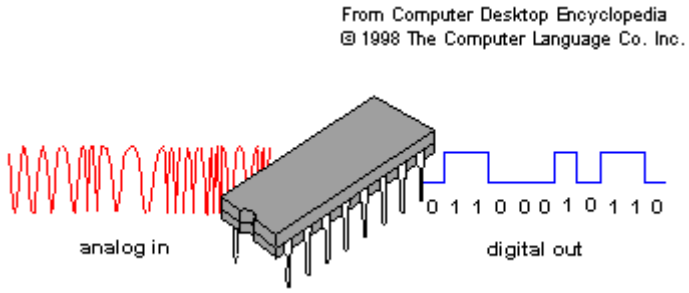




Aspecto del IDE Processing cuando esta en funcionamiento el programa de captura de valores de los tres potenciómetros.

Conversor Analógico-Digital (A/D)

Un conversor analógico-digital es un dispositivo electrónico capaz de convertir una señal analógica en un valor binario, en otras palabras, este se encarga de transformar señales analógicas a digitales (0's y 1's).



El dispositivo establece una relación entre su entrada (señal analógica) y su salida (Digital) dependiendo de su resolución . La resolución determina la precisión con la que se reproduce la señal original.

Esta resolución se pude saber, siempre y cuando conozcamos el valor máximo de la entrada a convertir y la cantidad máxima de la salida en dígitos binarios.

Resolución = +Vref/2^{n}(n-bits)

Por ejemplo, un conversor A/D de 8-bits puede convertir valores que van desde 0V hasta el voltaje de referencia (Vref) y su resolución será de:

Resolución = Vref/256 (2^{8})

Lo que quiere decir que mapeará los valores de voltaje de entrada, entre 0 y Vref voltios, a valores enteros comprendidos entre 0 y 255 (2^{n-1}).

La tarjeta Arduino utiliza un conversor A/D de 10-bits, así que:

Resolución = Vref/1024 (2^{10})

Mapeará los valores de voltaje de entrada, entre 0 y Vref voltios, a valores enteros comprendidos entre 0 y 1023 (2^{n-1}). Con otras palabras, esto quiere decir que nuestros sensores analógicos están caracterizados con un valor comprendido entre 0 y 1023. (Ver analogRead()).

Si Vref es igual a 5v, la resolución es aproximadamente de 5 milivoltios. Por lo tanto el error en las medidas de voltaje será siempre de sólo 5 milivoltios.

Caso de transmisión o envío de datos (comunicación) por el puerto serie:

Al enviar datos por el puerto serie, tenemos que tener en cuenta que la comunicación se realiza a través de valores con una longitud de 8-bits (Ver serialWrite(c) o serialRead(c)), mientras que como ya se hemos indicado, el A/D (Convertidor) de Arduino tiene una resolución de 10-bits.

Por ejemplo, si capturamos los valores de un sensor analógico (e.j. potenciómetro) y los enviamos por el puerto serie al PC, una solución podría se transformarlos en un rango entre 0 y 9 y en modo de codificación ASCII (carácter).

(dato capturado del sensor analógico * 9 / 1024) + 48;

0 ASCII -->decimal = 48

1 ASCII -->decimal = 49

etc..

En forma de código podría quedar como:

```
value1 = analogRead(analogPin1); //captura del valor de sensor analógico (0-1023)
serialWrite(treatValue(value1)); //volcado al puerto serie 8-bits
int treatValue(int data) {
return (data * 9 / 1024) + 48;// fórmula de transformación
}
```

Otra fórmula sería dividiendo por 4 ¿Esto es correcto? (1024/256)los valores capturados de los sensores analógicos, para convertirlos en valor de byte válido (0 - 255).

```
value = analogRead(analogPin)/4;
serialWrite(value);
```

Comunicación serie

Para hacer que dos dispositivos se comuniquen necesitamos un método de comunicación y un lenguaje o protocolo común entre ambos dispositivos. La forma más común de establecer dicha comunicación es utilizando la comunicación serie. La comunicación serie consiste en la transmisión y recepción de pulsos digitales, a una misma velocidad.

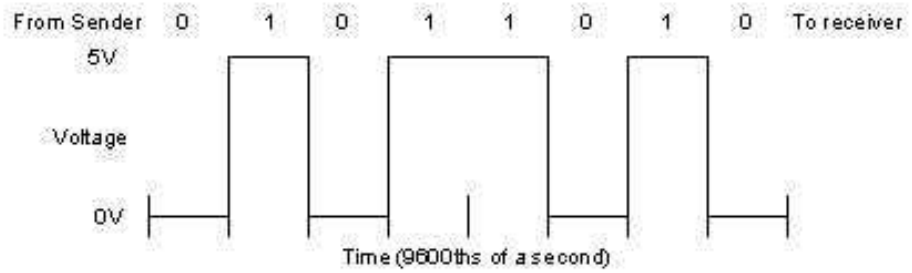
El transmisor envía pulsos que representan el dato enviado a una velocidad determinada, y el receptor escucha dichos pulsos a esa misma velocidad. Esta técnica es conocida como comunicación serie asíncrona. Un caso práctico es el de un MODEM externo conectado a un PC.

Por ejemplo, si tenemos dos dispositivos conectados y que intercambian datos a una velocidad de 9600 bits por segundo (también llamados baudios), el receptor capturará el voltaje que le está enviando el transmisor, y cada 1/9600 de un segundo, interpretará dicho voltaje como un nuevo bit de datos. Si el voltaje tiene valor HIGH (+5v en la comunicación con Arduino), interpretará el dato como 1, y si tiene valor LOW (0v), interpretará el dato como 0. De esta forma, interpretando una secuencia de bits de datos, el receptor puede obtener el mensaje transmitido.

Los dispositivos electrónicos usan números para representar en bytes caracteres alfanuméricos (letras y números). Para ello se utiliza el código estándar llamado ASCII (enlace), el cual asigna a cada número o letra el valor de un byte comprendido entre el rango de 0 a 127 ¿?. El código ASCII es utilizado en la mayoría de los dispositivos como parte de su protocolo de comunicaciones serie.

Así que si queremos enviar el número 90 desde un dispositivo a otro. Primero, se pasa el número desde su formato decimal a su formato binario. En binario 90 es 01011010 (1 byte).

Y el dispositivo lo transmitiría como secuencia de pulsos según el siguiente gráfico:



Otro punto importante, es determinar el orden de envío de los bits. Normalmente, el transmisor envía en primer lugar, el bit con más peso (o más significativo), y por último el de menos peso (o menos significativo) del formato binario.

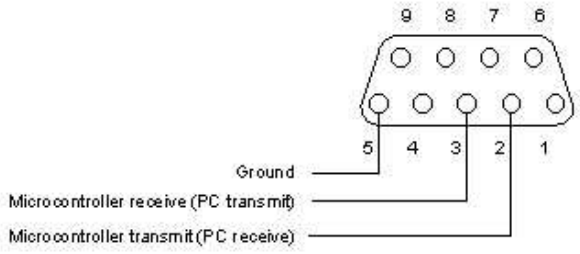
Entonces y como conclusión, para que sea posible la comunicación serie, ambos dispositivos deben concordar en los niveles de voltaje (HIGH y LOW), en la velocidad de transmisión, y en la interpretación de los bits transmitidos. Es decir, que deben de tener el mismo protocolo de comunicación serie(conjunto de reglas que controlan la

secuencia de mensajes que ocurren durante una comunicación entre dispositivos). Generalmente se usa el protocolo serie llamado RS-232 y interfaces (conectores vs puertos serie) que utilizan dicha norma.

Hasta no hace mucho, la mayoría de los PCs utilizaban el estandar RS-232 para la comunicación serie, pero actualmente los PCs están migrando hacia otras formas de comunicación serie, tales como USB (Bus Serie Universal), y Firewire, que permiten una configuración más flexible y velocidades de transmisión más altas.

Para conectar un dispositivo a un PC (o sistema operativo) necesitamos seleccionar un puerto serie y el cable apropiado para conectar al dispositivo serie.

Gráfico de Puerto serie RS-232 en PC (versión de 9 pines DB-9)



En Arduino y en función del modelo de placa que hayamos adquirido tendremos que elegir un cable RS-232 (estándar, no debe ser de tipo null modem) o USB o bien un adaptador RS-232/USB. (enlace a guía de instalación)

Palabras reservadas del IDE de Arduino

Estas palabras son constante, variables y funciones que se definen en el lenguaje de programación de Arduino. No se deben usar estas palabras clave para nombres de variables.

 # Constantes

HIGH
LOW
INPUT
OUTPUT
SERIAL
DISPLAY
PI
HALF_PI
TWO_PI
LSBFIRST
MSBFIRST
CHANGE
FALLING
RISING
false
true
null

 # Variables de designacion de puertos y constantes

DDRB
PINB
PORTB
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7

DDRC
PINC
PORTC
PC0
PC1
PC2

private
protected
public
return
short
signed
static
switch
throw
try
unsigned
void

 # Other

abs
acos
+=
+
[]
asin
=
atan
atan2
&
|
boolean
byte
case
ceil
char
char
class
,
//
?:
constrain
cos
{}
--
default
delay

loop
max
millis
min
-
%
/*
*
new
null
()
PI
return
>>
;
Serial
Setup
sin
sq
sqrt
-=
switch
tan
this
true
TWO_PI
void
while
Serial
begin
read
print
write
println
available
digitalWrite
digitalRead
pinMode
analogRead
analogWrite
attachInterrupts
detachInterrupts

PC3
PC4
PC5
PC6
PC7

DDRD
PIND
PORTD
PD0
PD1
PD2
PD3
PD4
PD5
PD6
PD7

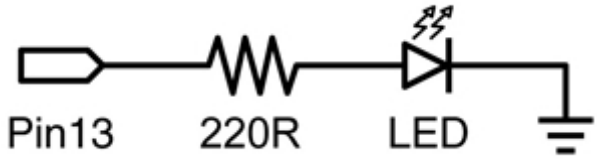
 # Tipos de datos

boolean
byte
char
class
default
do
double
int
long

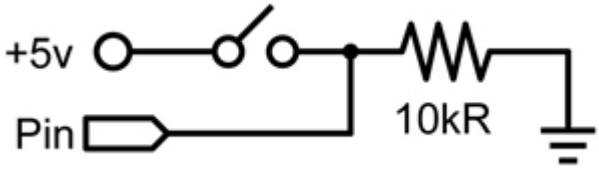
delayMicroseconds
/
/**
.
else
==
exp
false
float
float
floor
for
<
<=
HALF_PI
if
++
!=
int
<<
<
<=
log
&&
!
||

beginSerial
serialWrite
serialRead
serialAvailable
printString
printInteger
printByte
printHex
printOctal
printBinary
printNewline
pulseIn
shiftOut

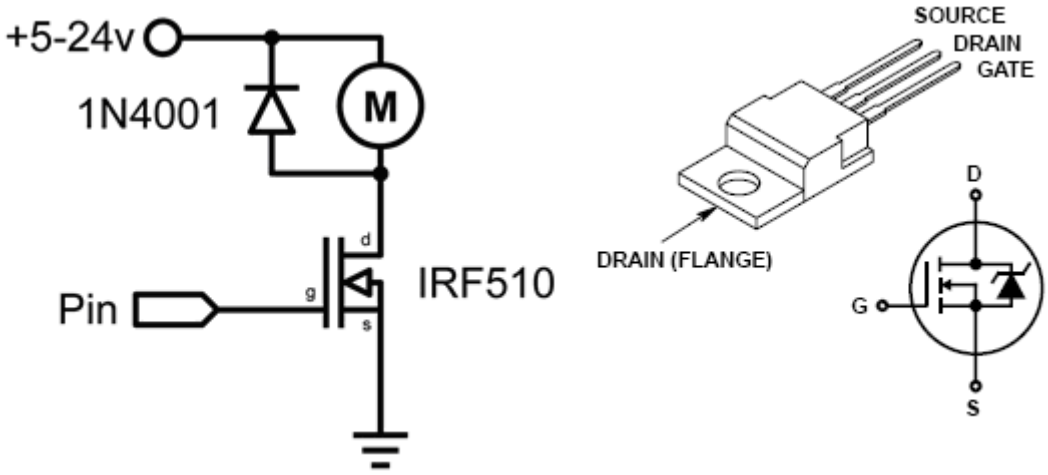
CIRCUITOS DE INTERFACE CON ARDUINO



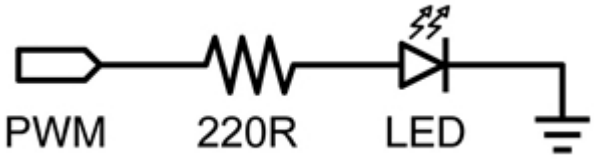
Conexión de un diodo Led a una salida de Arduino



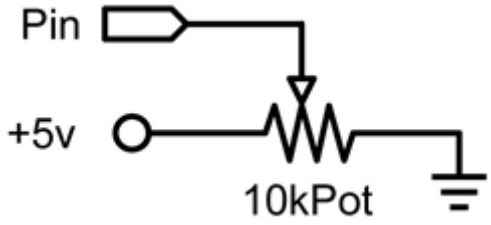
Conexión de un pulsador/interruptor



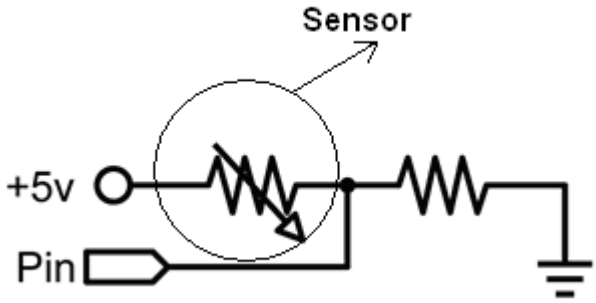
Conexión de una carga inductiva de alto consumo mediante un MOSFET



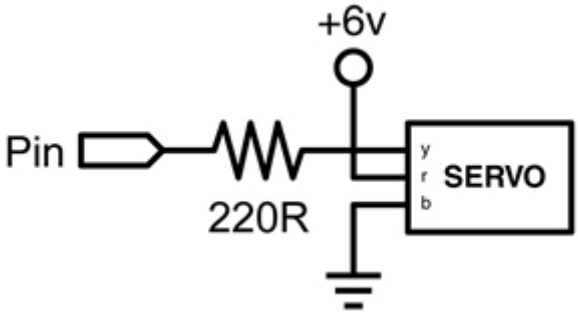
Conexión de una salida analógica a un LED



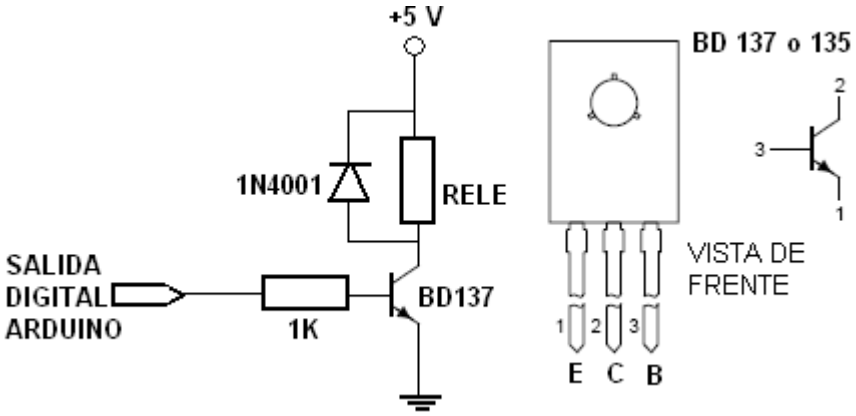
Entrada analógica mediante un potenciómetro



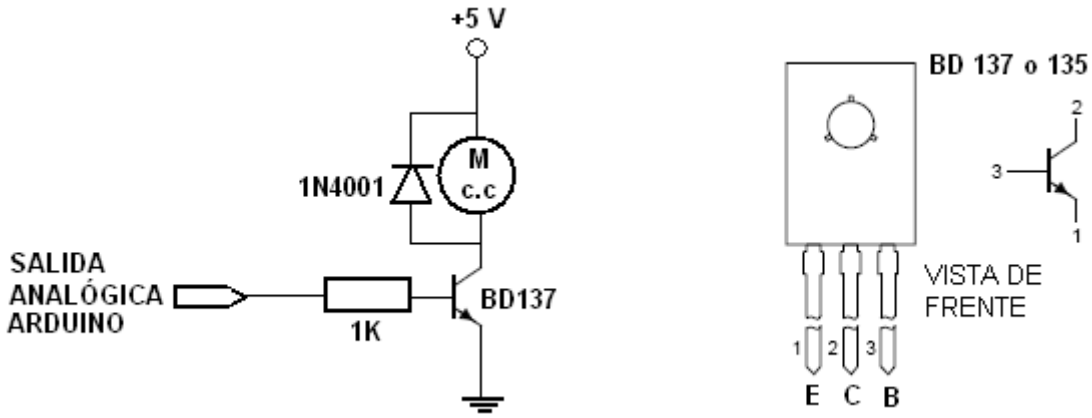
Conexión de un sensor de tipo resistivo (LRD, NTC, PTC..) a una entrada analógica



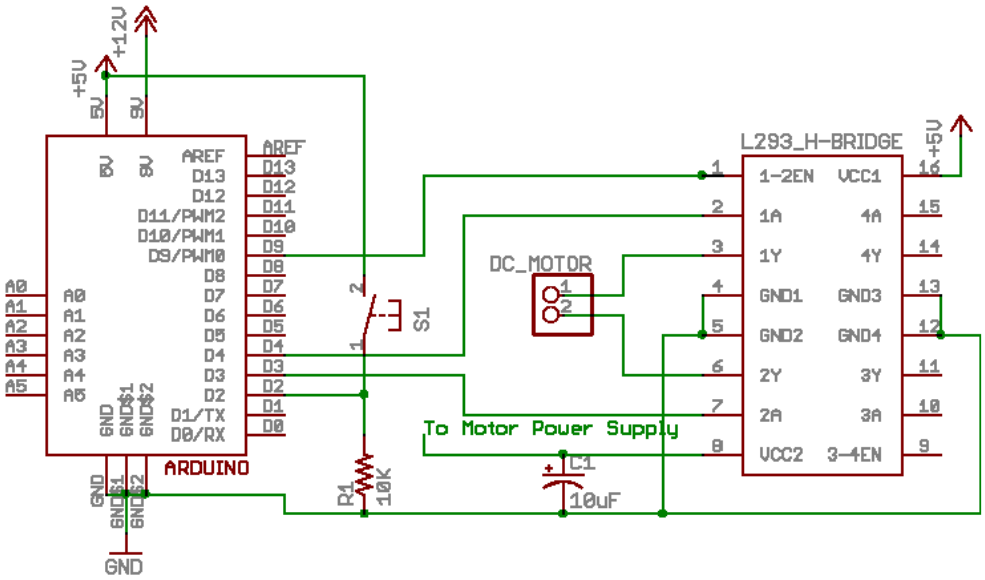
Conexión de un servo a una salida analógica.



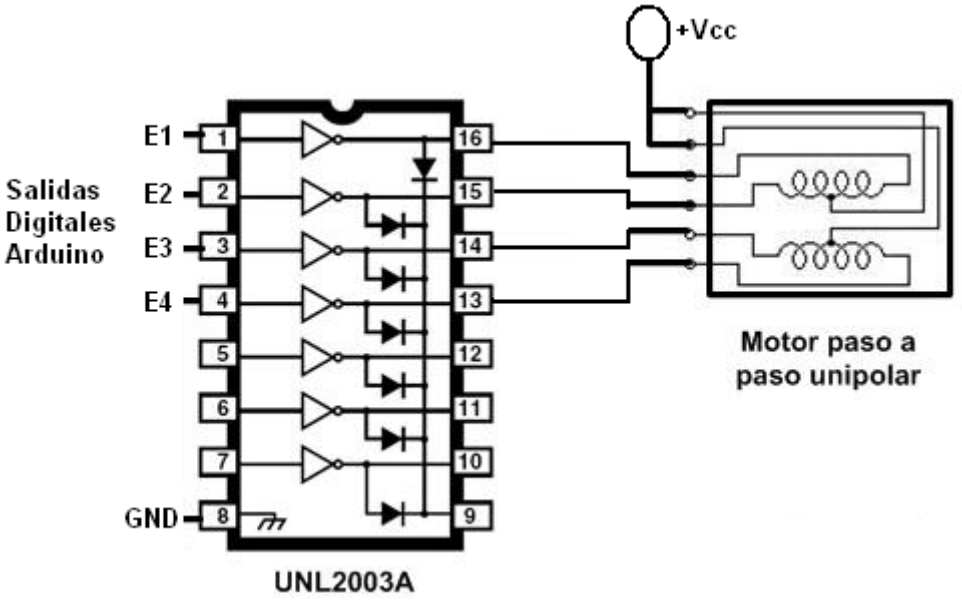
Gobierno de un Relé mediante una salida digital de Arduino



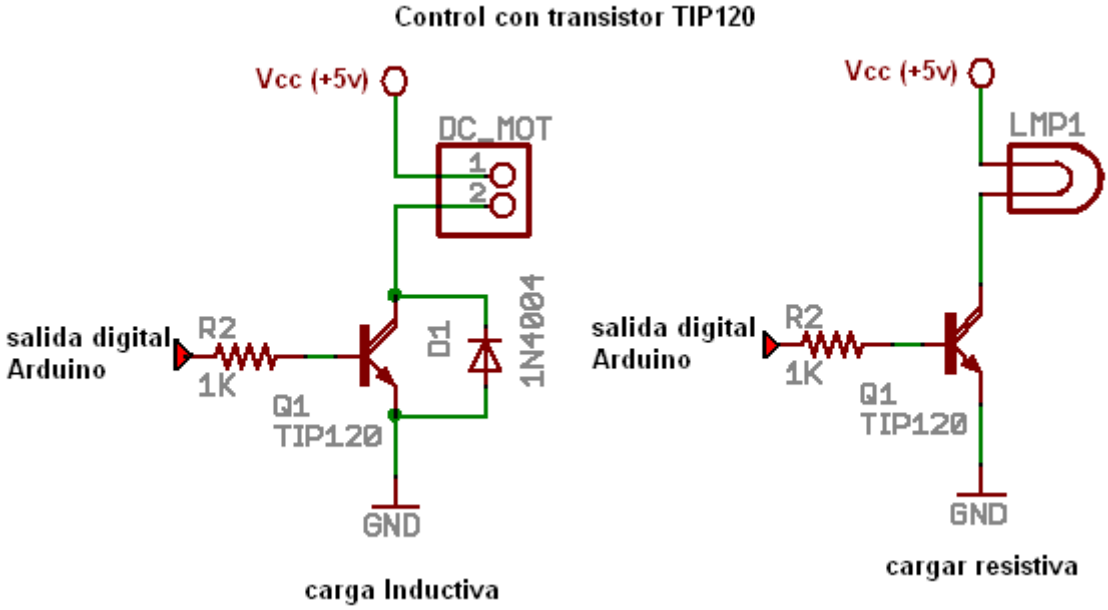
Gobierno de un motor de cc mediante una salida analógica de Arduino controlando la velocidad del motor



Control de un motor de cc mediante el CI L293



Control de un motor paso a paso unipolar



Control mediante transistor TIP120