The RoboBricks Project

Table of Contents

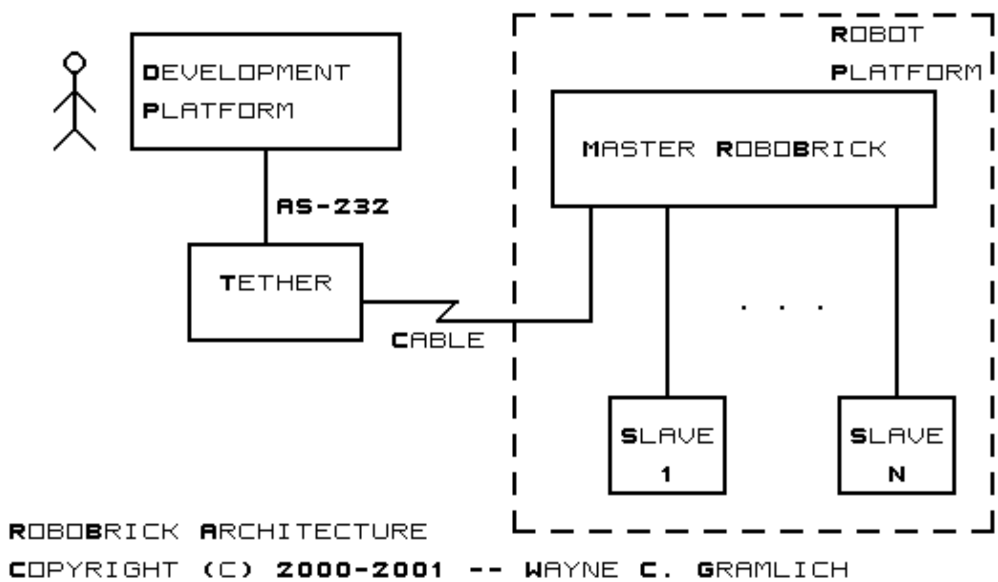
RoboBricks Introduction

The RoboBricks project provides a bunch of sensory and control modules that can be easily plugged together to form interesting robot systems. Indeed, they can be attached together with some plastic Lego® bricks to build robots, just like the Lego MindStorms® product. (Hence, the name RoboBricks.)

The basic concept behind RoboBricks is based on the small family of chips sold by FerretTronics®. The differences between RoboBricks and the FerretTronics chips are 1) RoboBricks support two way communication between the RoboBricks whereas the FerretTronics chips only offer one−way communication and 2) RoboBricks are at the printed circuit board level, whereas the FerretTronics products are at the chip level.

The current batch of RoboBricks are based around the PIC12Cxx 8−pin OTP (One Time Programmable) embedded microcontroller chips from Microchip®. From DigiKey®, the quantity 1 price is less than $2.00 a chip and the quantity 25 price is about $1.00 each. These chips do not have hardware UART's (Universal Asynchronous Receiver/Transmitter) in them, but a 2400 baud link can be emulated in firmware.

The overall RoboBrick architecture is shown below:



Bascially all software is developed on a full 32−bit development platform such as Windows®, MacOS®, or some flavor of Unix® (e.g. Linux®, Solaris®, BSD®, etc.) An RS−232 cable connects to a Tether RoboBrick which connects to the master RoboBrick via a 4 write cable. After the master RoboBrick has been programmed, the tether cable can be disconnected. The master RoboBrick is responsible for sending and commands and receiving data back from the slave robobricks.

When the master RoboBrick runs out of slave RoboBrick connections, processing power, or bandwidth, the robot platform can be repartioned to have two or more master RoboBricks with another supreme master RoboBrick in control of the masters. Thus, master RoboBricks can be cascaded in a hierarchical fashion.

This is news section of the RoboBricks projects. It is currently a work in progress.

RoboBricks Project News

The current RoboBricks News is:

2004−Apr
: Wayne is scambling to get the µCL compiler rewritten and ported to the Microsoft® before the next RoboBRiX article is published in Servo magazine.

2004−Mar
: The third article on RoboBRiX is published in Servo magazine.

2004−Feb
: The second article on RoboBRiX is published in Servo magazine.

2004−Jan
: The first article on RoboBRiX is published in Servo magazine.

2004−Dec
: Robobricks are renamed to RoboBRiX to make acquiring a register trademark easier. The first batch of RoboBRiX go on sale at the RobotStore. RobotStore is run by Mondo−tronics, Inc.

2003−Sep
: Contract negociations complete and contract is signed.

2003−Aug
: Contract negociation impass resolved and negociations continue.

2003−May
: Contract negociations with vendor reach an impass.

2003−Mar
: Selected a vendor to manufacture and market RoboBricks.

2003−Jan
: Dealt with a 50 day delivery time with our replacement PCB vendor. It was not really their fault though.

2002−Oct
: Dealt crappy boards from our PCB vendor. Ultimately wound up ordering replacement boards from another PCB vendor. Scratch one PCB vendor off our list. 2002−Aug−10
: Finished ordering parts for RoboBrick alpha program from AcroName, DigiKey, and Jameco. We were able to get a 10% discount from Jameco through our membership with the Robotics Society of America. The price of the sonar modules from Acroname was increased from $25 to $30.

2002−Aug−9
: The boards from CustomPCB have arrived back. No silkscreen was applied to the boards.

2002−Jul−18
: Ordered 20 copies of panel5 for the RoboBrick alpha program from CustomPCB in Mylasia for $265.

2002−Jul−11
: Sent out last call for the RoboBricks alpha program.

2002−Jun−28
: Sent a message to the Home Brew Robotics Club mail list announcing the RoboBricks alpha program.

2002−Jun−05
: Shipped Panel 4 off to OliMex for fabrication.

2002−Apr−30
: By the way, RoboBrick development is currently awaiting the completion of my newCNC motion controller board. This board is needed so that we can take panels that contain several RoboBricks and cut out the individuale RoboBricks under computer control.

2002−Jan−30
: The first of 4 RoboBricks talks was give at the Home Brew Robotics Club. Again, there is great interest in getting them. The talk slides are available.

2002−Jan−27
: The RoboBricks project was on display at the Tech Museum for a second day.

2002−Jan−26
: The RoboBricks project was on display at the Tech Museum. One of the most commonly asked questions was `How can we get some?'

2002−Jan−10
: The panel3 boards have come back, been cut into smaller boards and the assembly process continues.

2002−Jan−3
: The panel3 files have been sent off to Alberta Printed Circuits. Yeah!

2001−Dec−28
: The Light4−B and Servo4−C and are now panel3 ready. Added panels directory.

2001−Dec−27
: The MotorScan−A is now panel3 ready.

2001−Dec−24
: The IRSense2−A is now panel3 ready.

2001−Dec−22
: The SpeechQV1−A is now panel3 ready.

2001−Dec−21
: The OOPicHub15−A is now panel3 ready.

2001−Dec−20
: The IRBeacon8−A is now panel3 ready.

2001−Dec−17
: The LCD32−A is now panel3 ready.

2001−Dec−15
: The Motor3−A is now panel3 ready.

2001−Dec−14
: The SonarDT1−A and CompassDT1−A are now Fab3 ready.

2001−Dec−13
: The Motor2−C and Shaft2−C are now panel3 ready.

2001−Dec−12
: The Laser1−B is now panel3 ready.

2001−Dec−11
: The ProtoPIC−B and PIC876Hub10−B are now panel3 ready.

2001−Dec−10
: The Tether−C, Switch8−C, LaserHead1−B, and IRRemote1−A are now panel3 ready.

2001−Dec−8
: The LED10−B is now panel3 ready.

2001−Dec−7
: The InOut10−B is now panel3 ready.

2001−Dec−6
: The Harness−C is now panel3 ready.

2001−Dec−5
: The Compass8−B, Compass360−B, and BS2Hub8−B are now panel3 ready.

2001−Dec−4
: Both AnalogIn4−C and BIROD5−A are now panel3 ready.

2001−Dec−3
: AIROD4−A is now panel3 ready.

2001−Dec−1
: Starting to prepare RoboBricks for Fab3. Activity9−B is now panel3 ready.

2001−Nov−30
: Pretty much done with Laser1−A RoboBrick. panel2 is basically done. Panel3 will start shortly.

2001−Oct−23
Got the Light4−A RoboBrick working.
2001−Oct−22
Added the IRRemote1−A RoboBrick that Bill is working on. Got the AIROD2−A RoboBrick working.
2001−Oct−15
The AnalogIn4−B RoboBrick is done.
2001−Oct−10
The BIROD2−B RoboBrick is done.
2001−Oct−4
The LaserHead1−A RoboBrick is done. We are now getting a usable signal from across the room with very inexpensive IR sensors.
2001−Oct−1
The Activity9−A, PIC876Hub10−A, Tether−B RoboBricks are done.
2001−Sep−29
The Shaft2−B RoboBrick is done.
2001−Sep−28
The InOut10−A RoboBrick is done.
2001−Sep−16
The Servo4−B and Compass8−A RoboBricks are done. The Servo4−B boards have problems whenever the servo runs up against a stop; the next revision will need a separate power supply.
2001−Sep−12
The LED4−B, Switch8−B, Motor2−B RoboBricks are now done. Unfortuately, the Motor2 board required some trace rerouting; so a revision C will definitely be necessary. Also, the BS2Hub8 RoboBricks has been successfully programmed to talk to both a LED10−B and a Switch8−B. A working robot is sure to be on−line soon.
2001−Sep−6
The panel2 order has been sliced and diced and at least one of most of the boards have been built. The LED10−B board is the first one to burned into a OTP (One Time Programmable) device.
2001−Aug−22
The panel2 order has arrived back from Alberta Printed Circuits.
2001−Aug−20
The RoboBrick Specifications have been updated.
2001−Aug−19
The panel2 order was sent off to Alberta Printed Circuits.
2001−Aug−3
The panel2 order is ready to go. I will have to wait until I get back from a two week vacation before I submit it to Alberta Printed Circuits though.
2001−Aug−2
All of the master and slave RoboBricks are now in ready for panel2. There are some changes that need to be made to HobECAD, but that should only take a day or two. After I come back from a two week vacation, the panel2 run will take place.
2001−Jul−29
The various master and slave RoboBricks are now panel2 ready. The debug RoboBricks still need to be processed.
2001−Jun−21
The Activity4 RoboBrick and the BIROD2 RoboBricks are now panel2 ready.
2001−Jun−18
The Shaft2 RoboBrick is now panel2 ready.
2001−Jun−12
The LED10 and Out10 RoboBricks are now panel2 ready.

2001−Jun−5

The Switch8 and In8 RoboBricks are now 100% done. The release 0.46 version of µCL fixes yet another subtraction bug that was encountered.

2001−Jun−2

The Motor2 RoboBrick is 100% done. It was necessary to do some clock adjustment to get the Motor2 Robobrick to work every time. Thus, the clock adjust commands in the shared protocol really payed off. The release 0.45 version of µCL fixes yet another register bank swapping problem that was encountered (produces tighter code too.)

2001−May−23

At long last the Servo4 RoboBrick is 100% done. This is a big milestone, since Servo4 is one of the very hardest of the RoboBricks to implement. The release 0.44 version of µCL fixes some problems that were found along the way. Bill is working the bugs out of the LED10 RoboBrick.

2001−May−10

At long last the Threshold4 RoboBrick is 100% done. Most of the RoboBrick module ppages have been reorganized to leave the artwork out. This makes the resulting PDF files smaller. Also, the PIC12C509 programmer code was the µCL programming environment. The Parallel Port Server that Wayne uses to run his PIC Programmer got some modifications as well.

2001−Apr−23

The specification for Stepper1 is done. Now only the code needs to be written. (Heh−heh ;−)

2001−Apr−22

Worked on software for AnalogIn4 and InOut4. Now there is only Stepper1 left to be done.

2001−Apr−21

Worked on software for BIROD2, In8, LED10, Motor2, Out10, Shaft2, and Switch8. Only three modules left to finish up −− AnalogIn4 (easy), InOut4 (easy), and Stepper1 (very hard). In addition, the 0.36 release of the µCL compiler improves code generation for the PIC16C505 along with improved array indexing with constants.

2001−Apr−9

Rearranged the web pages into an introduction, news (i.e. this document), specifications, and modules. All of the underlying module directories now generate PDF files. The top level directory has two PDF files −− robobricks.pdf and rebobricks_all.pdf.

2001−Apr−2

Rewrote the RoboBrick Interrupt protocol stuff. There are now some shared commands for supporting interrupts. Improved string handling and fixed another register bank switching bug in µCL. Upgraded Threshold4 to use the new interrupt protocol stuff. Also, there is now a test program for testing Threshold4.

2001−Mar−4

Updated the led4.ucl code to be a complete implementation of the LED4 specification. Renamed Activity to be Activity4. Wrote the code for activity4.ucl.

2001−Mar−3

Updated the servo4.ucl code to be a complete implementation of the Servo4 specification. Better comments too. This version needs the 0.30 version of the µCL compiler.

2001−Mar−1

Fixed output to GPIO2 for PIC509's in µCL (release 0.29.) Also, added the assembly directive. The servo4.ucl code is working inside of a PIC12C509.

2001−Feb−14

Improved code generation for switch statements in µCL.

2001−Feb−13

Updated the µCL compiler to contain random number generation, oscillator calibration initialization, A/D converter initialization, and fixed array and string constant access from different code and data banks. Updated Threshold4 to contain a very complete implementation of the code.

2001−Feb−5

Updated the programming specifications for AnalogIn4, In8, Shaft2, Switch8, Threshold4, and Activity4 RoboBricks.

2001−Jan−31

Showed CDBot following a line of black electrical tape using RoboBricks at the Home Brew Robotics Club meeting. Some folks at the Tech Museum showed up and were quite interested in RoboBricks. Apparently there is some sort of similar technology called Stackable Core Modules being developed over at Twin Cities Robotics Group (TCRG).

2001−Jan−21

Added links to CDBot.

2001−Jan−17

Updated Activity4, Harness, PIC16F876, and Tether to get the directions of SIN and SOUT properly oriented. The programming specifications for the Motor2 RoboBrick have been updated. There is still a bug in µCL that causes the delay routine to have a non−uniform delay.

2001−Jan−16

The boot loader for PIC16F876 is almost working with the [download] button in the µCL graphical user interface. The boot loader is residing in code bank 3 (0x1800) and uses register bank 3 (0x180).

2000−Dec−30

Updated programming specification of In8 RoboBrick.

2000−Dec−21

Added the last remaining pictures for Threshold4 and PIC16F876. The µCL compiler now has support to directly program a Microchip microcontroller.

2000−Dec−11

Added most of the remaining pictures (Activity4, AnalogIn4, Bench, Hub8, LED4, Motor2, ProtoPIC8Pin, Stepper1, and Switch8.) We're only missing PIC16F876 and Threshold4 pictures now. We've got LED4 working with a UV erasable PIC12CE674. Motor2 is starting to work. There are some command transmission reliability problems being worked on. Sometimes the RoboBricks do not reset properly on power up. Our short term goal is to get a Line following robot working using a battery and the Hub8, PIC16F876, Threshold4, and Motor2 RoboBricks.

2000−Nov−30

Added a whole bunch of pictures of individual RoboBricks (Birod2, Harness, In8, InOut4, LED10, out10, Servo4, Shaft2, and Tether). We're still missing a picture of LED4. Updated the source files for harness and LED4. Continued bug fixing in µCL. LED4 code is now working using the PIC16F876 emulator. Stand−alone execution using a PIC12CE764 UV erasable part should occur soon. Starting to add PIC programmer support to µCL development environment.

2000−Nov−15

Rearranged the µCL language specification to be in its own file. Documented the emerging µCL programing environment. Added a whole bunch of issue sections to the revision A RoboBricks as they get built out. There is now an over−arching RoboBrick Software Protocol. Also, because Bill wired up a cable backwards, I added a Cable Mechanical Specification.

2000−Nov−9

The following RoboBricks are starting to work −− Tether (100% done), Harness (100% done), Bench (100% done), Hub8 (100% done), PIC16F876 (Needs lots of software), Emulate (100% done), and LED10 (50% done; more software needed). There is still a bunch of software development to do, but the hardware seems to be working fairly well. The Revision B boards are going to switch from a 4−wire bus to a 6−wire RoboBrick interconnect standard. We had a heck of a time finding a 4−wire crimper; we figure most people will have a much easier time finding a 6−wire crimper. Lastly, the latest version of µCL now has the beginnings of an integrated development environment (sorry, no documenation yet.)

This is the specification portion of the RoboBricks Projects. It is currently work in progress.

RoboBricks Specifications

Table of Contents

1 Introduction

There are three components to the RoboBrick specifications −− the software protocol, electrical protocol, and the mechanical connector specification.

2 Software Protocol

The RoboBrick protocol is very simple. The controlling processor sends out one or more command bytes and the selected Robobrick responds with one or more response bytes. The RoboBrick protocol is asynchronous serial in 8N1 format (i.e. 1 start bit, 8 data bits, no parity, and 1 stop bit.) The protocol speed is at 2400 baud.

All of the slave RoboBricks share some common commands to help with glitches, RoboBrick identication, and clock drift management. These are discussed briefly below:

Glitches
: A glitch occurs when a spurrious signal manages to cross−couple onto a RoboBrick signal wire. There a few commands to help combat glitches.

Identification
: Each RoboBrick has a bunch of identification information in it. This identification information contains the major and minor version numbers of the RoboBrick protocol, the major and minor version numbers for the RoboBrick itself and a 128−bit random number.

Clock Drift
: RoboBricks are currently implemented using low cost 8−pin PIC processors running off of an internal 4MHz RC oscillator. While this reduces costs, RC oscillators are notoriously sensitive to temperature variations. While most RoboBrick applications will choose to ignore this issue, there are a variety of commands that can be used to adjust the RC osciallator frequency up and down as needed.

The shared commands are summarized textually below:

Glitch
: Sometimes a strong current pulse from elsewhere in the robot will cross couple with a RoboBrick signal wire and cause a spurrious start bit. The rest of the bits will be read as all ones. We call such a command the glitch command and all it does is bump a counter that can be read back via the Glitch read comand.

Glitch Read
: This command returns the current value of the glitch counter and then resets the counter to zero.

ID Reset

This command will reset the ID pointer register.

ID Next

This command will return the next byte of identifier information. The ID pointer register is incremented.

Clock Pulse

This command cause the system to send a null character back. This pulse width can be measured by the master system to determine if the RC oscillator is running fast or slow.

Clock Read

This command returns the current value of the clock adjust register.

Clock Increment

This command increments the clock adjust register.

Clock Decrement

This command decrements the clock adjust register.

The shared command protocol is defined in the table below:

Shared RoboBrick Commands										
Command	Bit Number								Send/Receive	Description
	7	6	5	4	3	2	1	0		
Glitch	1	1	1	1	1	1	1	1	Send	Glitch Command
Glitch Read	1	1	1	1	1	1	1	0	Send	Glitch Read Command
	g	*g*	*g*	*g*	*g*	*g*	*g*	*g*	Receive	Returns 8−bit *gggggggg* glitch counter value
ID Reset	1	1	1	1	1	1	0	1	Send	ID Reset Command
ID Next	1	1	1	1	1	1	0	0	Send	ID Next Command
	i	*i*	*i*	*i*	*i*	*i*	*i*	*i*	Receive	Returns next 8−bit *iiiiiiii* identification byte value
Clock Pulse	1	1	1	1	1	0	1	1	Send	Clock Pulse Command
	0	0	0	0	0	0	0	0	Receive	Returns a null byte that can be timed for clock drift
Clock Read	1	1	1	1	1	0	1	0	Send	Clock Read Command
	c	*c*	*c*	*c*	*c*	*c*	*c*	*c*	Receive	Returns the 8−bit *cccccccc* clock adjust register value
Clock Increment	1	1	1	1	1	0	0	1	Send	Clock Increment Command
Clock Decrement	1	1	1	1	1	0	0	0	Send	Clock Decrement Command

The identification bytes in each RoboBrick are arranged as follows:

Offset	Name	Description
0	RBMajor	Major Version Number for identification stream (currently 1)
1	RBMinor	Minor Version Number for identification stream (currently 0)
2	BrickID	BrickID for common Bricks (see table below)
3	BrickRev	Brick Revision (0=A, 1=B, 2=C, 3=D, 4=E 5=F,

		6=G 7=H, etc.)
4	BrickFlags	8 RoboBrick Specific Flags
5	Reserved0 (use 0)	Reserved for future use
6	Reserved1 (use 0)	Reserved for future use
7	Reserved2 (use 0)	Reserved for future use
8−23	UID0−15	128−bit Unique Identifier (Randomly Generated)
24	NameLength	RoboBrick Name Length
Next NameLength Bytes	BrickName	Name of RoboBrick in ASCII
Next Byte	VendorLength	Vendor Name Length
Next VendorLength Bytes	VendorName	Vendor Name in ASCII
Next Byte	OptionsLength	Options Length (optional)
Next OptionLength Bytes	Options	Option Bytes (optional)

The BrickFlags are currently defined as follows:

Bit								**BrickFlags Description**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
							c	*c=1* => clock adjust supported
						i		*i=1* => interrupt protocol supported
					o			*o=1* => optional bytes follow vendor name
				b				*b=1* => Baud rate change is allowed

The RoboBricks are given for *BrickID* identifiers on a first come first serve basis. The following identifiers have already been allocated:

ID	RoboBrick Name
0−7	*Reserved for experimenters*
8	LED4 (obsolete)
9	LED10 (obsolete)
10	In8 (obsolete)
11	BIROD2 (abandoned)
12	AnalogIn4
13	Out10 (obsolete)
14	Motor2
15	Servo4
16	Shaft2
17	Stepper1

18	Switch8 (obsolete)
19	Threshold4 (obsolete)
20	AIROD2 (abandoned)
21	Compass360 (obsolete)
22	Compass8 (obsolete)
23	InOut10
24	Laser1
25	Light4
26	Sonar1 (abandoned)
27	AIROD4
28	BIROD5 (abandoned)
29	SONARDT1
30	Bill Hubbard's RC4
31	IRProximity2
32	Digital8
33	DualMotor1Amp
34	IREdge4

Each brick is assigned a 128−bit random number. The probability of two bricks being assigned the same random number is $1/(2^{128})$ which is a pretty small number. On Linux, the random numbers can be read from `/dev/random` (or `/dev/urandom`.)

3 Interrupts

At 2400 baud, it can take a while to poll several input RoboBricks to see if anything interesting has occured. Sometimes RoboBricks are sensing inputs that need a response that is faster than strict polling can provide. For example, bumper detectors. To support low latency, many RoboBricks support the RoboBrick Interrupt Protocol.

The RoboBrick Interrupt Protocol is very simple. Each RoboBrick that supports the protocol has two bits −− the interrupt pending bit and the interrupt enable bit. The interrupt pending bit is set by the RoboBrick when a prespecified user event has occured. The interrupt enable bit is set to allow the interrupt to occur.

The following steps occur when using interrupts:

1. The user sends some RoboBrick specific commands to set up the conditions for setting the interrupt pending bit.
2. The user sends an enable interrupt command.
3. When the interrupt condition occurs, the interrupt pending bit is set and an interrupt is triggered. The interrupt is signaled by dropping the output line from the RoboBrick to a low.
4. The master processor detects that the interrupt has occured.
5. One or more commands are sent to the Robobrick to figure out what happened. When the first bit of the first command is received, the RoboBrick clears both the interrupt enable bit and restores its transmit line high.
6. Depending upon the RoboBrick, the interrupt pending bit may need to be cleared by sending pending

bit clear command. For some other RoboBricks, the condition that sets the interrupt pending bit may automatically clear.

If the user needs to query the RoboBrick before the interrupt occurs, any command will clear the interrupt enable bit. In order to get another interrupt, another interrupt enable command must be sent.

Since many RoboBricks will implement the RoboBrick Interrupt Protocol, there are some common commands defined to support the protocol:

Command	**Send/ Receive**	**Byte Value**								**Discussion**
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
Read Interrupt Bits	Send	1	1	1	0	1	1	1	1	Return the interrupt enable bit *e* and pending bit *p*.
	Receive	0	0	0	0	0	0	*e*	*p*	
Set Interrupt Bits	Send	1	1	1	1	0	0	*e*	*p*	Set interrupt enable bit to *e* and pending bit to *p*.
Set Interrupt Pending	Send	1	1	1	1	0	1	0	*p*	Set interrupt pending bit to *p*.
Set Interrupt Enable	Send	1	1	1	1	0	1	1	*e*	Set interrupt enable bit to *e*.

4 Baud Rate Control

As of the version 1.1 of the RoboBricks protocol, the ability to change baud rate has been added. All RoboBrick modules start out communicating at 2400 baud using an 8N1 (1 start bit, 8 data bits, No parity, and 1 stop bit) asynchronous serial protocol. A RoboBrick indicates that it can support increases in its baud rate by seting bit 3 in the BrickFlags byte (5th byte = offset 4) of the RoboBrick identificiation string.

There are three RoboBrick baud rate control commands:

Read Available Baud Rates
: This command will return a bit mask of the baud rates supported by the RoboBrick.

Read Current Baud Rate
: This command will return a code that specifies what the current baud rate is.

Set New Baud Rate
: This command will set the new baud rate.

The available baud rates are in the table below:

Baud Rate	**Code**	**Mask (binary)**
2400	0	0000 0001
4800	1	0000 0010
9600	2	0000 0100
19200	3	0000 1000
38400	4	0001 0000
57600	5	0010 0000
115200	6	0100 0000
230400	7	1000 0000

The detailed commands are:

Command	Send/ Receive	Byte Value 7	6	5	4	3	2	1	0	Discussion
Read Available Baud Rates	Send	1	1	1	0	1	1	1	0	Return the available baud rates as a mask *abcdefg* where *a*=230400, *b*=115200, ..., *h*=2400
	Receive	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	
Read Current Baud Rate	Send	1	1	1	0	1	1	0	1	Return the current baud rate as *rrr* where *rrr*=000 => 2400, *rrr*=001 => 4800, ..., *rrr*=111 => 230400
	Receive	0	0	0	0	0	*r*	*r*	*r*	
Set New Baud Rate	Send	1	1	1	0	1	1	0	0	Set the new baud rate to *rrr* where *rrr*=000 =>2400, *rrr*=001 => 4800, ... *rrr*=111 =>230400. The first two bytes are sent at the old baud rate. The next two bytes are sent/received at the new baud rate. If the RoboBrick does not receive the last byte correctly at the new baud rate, this command will fail and the baud rate will remain unchanged.
	Send	0	*r*	*r*	*r*	0	*r*	*r*	*r*	
	Receive	0	1	0	1	0	1	0	1	
	Send	0	1	0	1	0	1	0	1	

The Set New Baud Rate command is a little tricky and merits additional discussion. Changing baud rates is potentially risky. If the host attempts to change the baud rate, and the target RoboBrick sets the baud rate incorrectly, the host will no longer be able to successfully communicate with the RoboBrick. The only way to recover is to reset power to the RoboBrick to get it back to 2400 baud. For this reason, the command to set the new baud rate requires positive acknowledgement that the baud rate has changed. The first two bytes of the command are sent at the old baud rate, where the second byte specifies the desired new baud rate. The next two bytes of the command are performed at the new baud rate. If the host does not get a '0101 0101', the knows that something has gone wrong. If the RoboBrick does not get a '0101 0101' from the host, the RoboBrick knows that something has gone wrong. If anything goes wrong, the baud rate reverts back to the original value.

After the baud rate for a RoboBrick has been set, it probably makes sense to run the clock adjust algorithm to make sure the RoboBrick clock is as close as possible to the host clock.

5 Electrical Specification

The RoboBrick electrical protocol is based around a 4 wires using standard 5−pin straight headers with .100 inch between the pins. The 4 wires are:

Ground (GND)
: Ground return

Power (PWR)
: +5 Volts of regulated DC power

Serial Down (MOUT => SIN)
: Serial bit stream down using 8N1 (1 start bit, 8 data bits, no parity, and 1 stop bit) asynchronous signaling at 2400 baud. The signal levels swing between .2 volts and +4.8 volts. A 1 is indicated by 4.8 volts and a zero is indicated by .2 volts. The start bit is a zero and the stop bit is a one.

Serial Up (MIN <= SOUT)

Serial bit stream up using 8N1 asynchronous signaling at 2400 baud. The signal levels swing between ground and +5 volts. A 1 is indicated by 4.8 volts and a zero is indicated by .2 volts. The start bit is a zero and the stop bit is a one.

The printed circuit boards use standard .100 straight male headers. These are usually purchased in lengths of 30−40 pins (e.g. Jameco 160881), and are snipped to a length of 5 pins. The cables are manufactured using 5 pin female cable headers with .100 centers (e.g. Jameco 163686).

The pin outs for master boards are:

Pin 1 (GND)
GND stands for GrouND return.
Pin 2 (NC)
NC stands for No Connection. This pin is snipped off for polarization purposes.
Pin 3 (PWR)
PWR stand sfor PoWeR and corresponds to +5 volts of regulated DC power.
Pin 4 (MOUT)
MOUT stands for Master OUT and corresponds to the serial down connection for sending serial data from the master RoboBrick to the slave RoboBrick.
Pin 5 (MIN)
MIN stands for Master IN and corresponds to the serial up connection for sending serial data from the slave RoboBrick to the master RoboBrick.

The pin outs for the slave boards are:

Pin 1 (GND)
GND stands for GrouND return.
Pin 2 (NC)
NC stands for No Connection. This pin is snipped off for polarization purposes.
Pin 3 (PWR)
PWR stands for PoWeR and corresponds to +5 volts of regulated DC power.
Pin 4 (SIN)
SIN stands for Slave IN and corresponds to the serial down connection for sending serial data from the master RoboBrick to the slave RoboBrick.
Pin 5 (SOUT)
SOUT stands for Slave OUT and corresponds to the serial up connection for sending serial data from the slave RoboBrick to the master RoboBrick.

The cables are wired straight through with pin 2 left unconnected (i.e. pin 1 to pin 1, pin 3 to pin 3, pin 4 to pin 4 and pin 5 to pin 5.) 22 AWG stranded wire must be used for the cable wires. There is no offical color code for the cable wires.

Pin 2 is used to polarize the cable. A male pin (Jameco 145357) is jammed into pin 2 and the male pin that sticks out is snipped off For a properly polarized cable and RoboBrick boards, it is not possible to plug the cable into the board either backwards or off by one. It is possible to plug a master to a master and a slave to slave, but no harm results.

6 Mechanical Specification

RoboBricks are compatible with the Lego®, MegaBloks®, and RokenBok® plastic toys. The standard pitch between studs on these toys is approximately 5/16 inches (or 4mm.) This means that a 4 by 4 square is 1.25 inches. The RoboBrick boards are always in units of 1.25 inch squares. All RoboBricks are 2.5 inches high by some multiple of 1.25 inches wide. Thus, the smallest RoboBrick is 1.25 by 2.5 inches, the next size up is 2.5 by 2.5, and the one after that is 2.5 by 3.75, etc.

The top and bottom of each RoboBrick has a row of holes that fit over the studs on plastic bricks. Thus, the holes are at least .195 inches in diameter. Since most RoboBrick printed circuit boards are double sided with plated through holes, the holes should probably be drilled with at least a .210 inch drill. The formula for determining the offset for stud N (where N starts at 0) is:

Offset = U/2 + N × U

where U is 5/16 of an inch. The expanded formula is:

Offset = .15625 + N × .31250

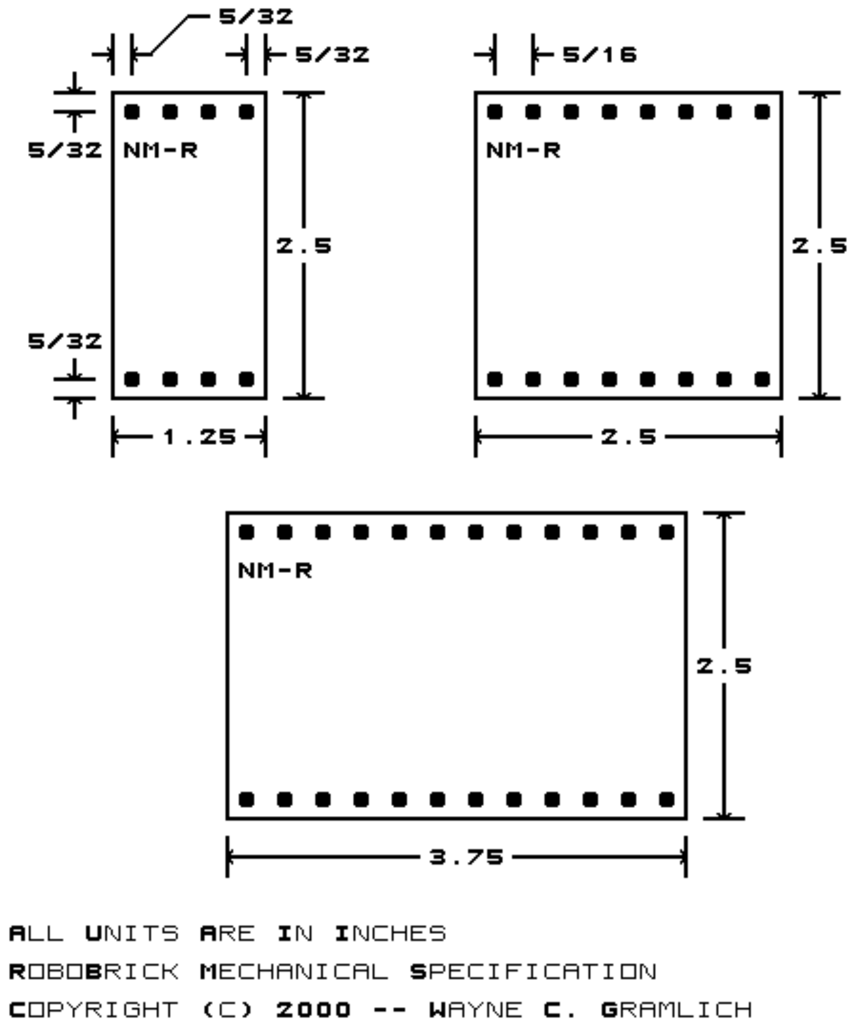
The first 8 values for this formula are shown below:

Count	**Offset (in.)**	**N×.05+/−offset**
0	0.15625	3×.05+.00625
1	0.46875	9×.05+.01875
2	0.78125	16×.05−.01875
3	1.09375	22×.05−.00625
4	1.40625	28×.05+.00625
5	1.71875	34×.05+.01875
6	2.03125	41×.05−.01875
7	2.34375	47×.05−.00625
Repeats on 2.5 inch grid		

After 8 entries, the numbers repeat offset by 2.5 inches.

Somewhere on each RoboBrick, must be name of the RoboBrick. The standard naming convention is `{name}−{revision}'. For example, 'Digital8−A', `DualMotor1Amp−B', etc. Please note that the revision corresponds to *both* hardware revision and the software revision inside the microcontroller.

A diagram of the mechanical specification is shown below:



This is the modules component of the RoboBricks project. It is currently work in progress.

RoboBricks Modules

Table of Contents

Robobricks Catagories

Robobricks are partioned into four catagories:

Master Robobricks
: A master module contains some sort of processor and a bunch of connectors for connecting to and controlling 1 or more slave Robobricks (see definition below.) Many master modules have some sort of power regulator for producing 5 volts from the battery voltage.

Slave Robobricks
: A slave module performs some sort of input or output function. There are usually many slave Modules per robot.

Debug Robobricks
: A debug module provides some sort of debugging function. These modules are only during robot development After a robot has been debugged, the debug modules can be removed.

Obsolete Robobricks
: An obsolete module is one that no longer makes any sense to build. Typically they have been replaced by something better. These are listed in a separate section below.

A robot consists of one master Module and one or more slave Modules. Debug Modules are added and removed as needed for debugging.

Master Robobricks

The following master Robobricks are under active development:

MicroBrain8
: The MicroBrain8 module provides a master module controlled by any processor that is pin with the Basic Stamp II® from Parallax®. This module has a battery conntection, power switch, and 5 volt linear voltage regulator. It has hub connections for controlling up to 8 modules.

PICBrain11

The PICBrain11 module provides a master module controlled by a PIC16F876 from Microchip®. This module has a battery connection, power switch, and 5 volt linear voltage regulator with fuse. It has hub connections for controlling up to 11 modules. It can be directly connected to an RS−232 port on a host computer.

Eventually, there should be master Modules for each of the more popular microcontrollers out there (e. g. Basic Stamp II, HC11, AVR, 8051, Rabbit, etc.)

Slave Robobricks

The following slave Robobricks are being actively developed (in alphabetical order):

AnalogIn8

The AnalogIn8 module allows for the input of up to 8 analog voltages between 0 and 5 volts with a resolution of up to 10 bits. There are 6 trim pots on board that can be jumpered to the first 6 analog inputs.

Compass8

The Compass8 module uses the 1490 digital compass module from Dinsmore Instrument Company. This module provides a 8 directions N, NE, E, SE, S, SW, W, and NW. This module can prevent a robot from getting totally turned around.

CompassDT1

The CompassDT1 module uses the CMPS01 compass module from Devantech to provide a compass bearing between 0.00 and 359.00 degrees.

Digital8

The Digital8 module has 8 I/O lines that can be used for input or output. A line can be changed from input to output and back under program control. This module replaces InOut10, Out10, In8, and InOut4 Modules.

DualMotor1Amp

The Robobricks DualMotor1Amp module an control up to two small DC motors. The motor voltage input can range from 5 volts to 24 volts. It is capable of accelleration ramping and electronic breaking. Lastly, it has an optional watchdog feature that will turn the motors off if a command has not been received in a while.

DualMotor2Amp

The Robobricks DualMotor1Amp module an control up to two small DC motors with a current of up to 2 amps. The motor voltage can be as high as 48 volts. The two internal H−Bridges can be tied together to provide a current capacity of 3.5 amps to a single motor.

IRBeacon8

The IRBeacon8 module is used to provide an IR beacon that Modules can home in on. It is designed for both stand alone operation and to work in a Module setting.

IRDistance4

The IRDistance4 module is used measure distances using up to 4 Sharp GP2D12 (InfraRed Optical Distance) modules. The modules are typically attached to IRDistanceHolder modules.

IRDistance8

The IRDistance8 module is used measure distances using up to 8 Sharp GP2D12 (InfraRed Optical Distance) modules. The modules are typically attached to IRDistanceHolder modules.

IRDistanceHolder

The IRDistanceHolder module is used carry 1 4 Sharp GP2D12 (InfraRed Optical Distance) module.

IREdge4

The IREdge4 module provides a way to use inexpensive IR emitter/detector pairs to sense changes in surface reflectivity.

IRProximity2
: The IRProximity2 module is used detect objects via reflection of an InfraRed (IR) light. There are two light sources and one light receiver along one edge of the board.

IRRemote1
: The IRRemote1 module is used to send and receive IR signals. Currently, only signals from Sony style IR Remotes are supported.

Keypad12
: The Keypad12 Module has 12 push buttons for user control inputs and 12 LED's for direct output.

LCD32
: The LCD32 module displays 2 lines of 16 characters each using an LCD display.

LCD32Holder
: The LCD32 module holdes a 2×16 LCD module that is plugged into the LCD32 module.

Laser1
: The Laser1 Module is able to detect when an inexpensive laser pointer is reflecting off of a passive reflector beacon. In conjunction with 3 reflector beacons placed in known locations, it is possible for a robot to triangulate its position accurately.

LaserHead1
: The LaserHead1 Module is a board that be used to mount a laser pointer and some photo detectors on. It is meant to work in conjunction with the Laser1 Module.

LCD32
: The LCD32 Module provides a way to output up to 32 characters (2 lines of 16 characters each) to a Liquid Crystal Display.

LED10
: The LED10 Module provides the ability to output 10 bits to 10 on board LED's.

Line3
: The Line3 Module provides the ability to sense lines on flat surfaces for building line/maze followers.

PIC876Hub10
: The PIC876Hub10 module provides a master Module controlled by PIC16F876 from Microchip®. This module has a battery connection, power switch, and 5 volt linear voltage regulator with fuse. It has hub connections for controlling up to 10 Modules. Lastly, it has the ability to sense the battery voltage. This module has morphed into the PICBrain11.

ProtoPIC
: The ProtoPIC Module is just a prototyping board for the 8−pin PIC's (e.g. PIC12C509 and PIC12C672) and the 14−pin PIC's (e.g. PIC16C505.)

RCInput8
: The RCInput8 module reads up to 8 RC servo pulse widths from a standard RC server receiver.

Reckon2
: The Reckon2 module is used to manuver a robot. It can contol two motors in "differential steering" mode. Each motor needs to have a shaft encoder with a quadrature output. If there is enough resolution on the shaft encoder and the wheels are not too "squishy", it is possible to keep pretty accurate track of a robot's location and bearing using deduced reckoning.

Rotation2
: The Rotation2 module can keep track of up to two quadrature shaft encoders.

SpeechQV1
: The SpeechQV1 Module is used to perform speech synthesis to allow a robot to talk.

Sense3
: The Sense3 module contains a infrared distance, sonar and laser bearing sensor that is meant to be scanned using a hobby servo.

Servo4
: The Servo4 Module is used to connect to up to 4 standard servo motors.

SonarDT1
The Sonar1 Module is used to provide a Module interface to the SRF04 sonar range finder from Devantech.

Sonar8
The Sonar8 module can drive up to 8 SonarSR modules.

SonarSR
The SonarSR module provide an ultra−sonic send/receive functionality.

SRFHolder
The SRFHolder holds a Robot Electronics SRF04 sonar ranging module.

Stepper1
The Stepper1 Module can control one small unipolar or bipolar stepper motor.

Switch8
The Switch8 Module will read in 8 bits of data from on−board switches.

Below is a list of slave Robobricks that are under consideration for future development:

Analog Output Module
An anilog output module can output a single 5−bit output voltage.

Tilt Module
This module detects what its current inclination is.

IR Remote Module
This module detects signals from an IR Remote control.

Microphone Module
This module detects the current sound level. inclination is. It does not provide way to record sound.

FM Synthesis Module
This module produces sounds using FM synthesis.

Temperature Module
This module measures the current temperature.

Light Module
This module measures the current amount of ambient light.

Miscellaneous Robobricks

The following Miscellaneous Modules are being worked on:

IRDistance Holder
A board for holding a Sharp GP2D12 infrared distance sensor.

LaserHolder1
A board mecahnically supporting a small laser pointer for the Sense3 module.

Scan Base
A board for electrically connecting to a Scan Panel.

Scan Panel
A board for mounting on top of a servo horn. This typically used to mount other sensors, such as sonar or IR distance sensors, to be swept back and forth. From revision B on, this board is used to electrically connect to a Sense3 module.

Servor Adaptor 0.4
This board is used for adapting servos with .4 inch mounting hole on systems that "Lego" stud spacing.

Shaft Sense 2
This module is meant to pick up a quadrature single from shaft mounted optical encoder wheel.

SRF Holder
: This board is used to hold a SRF04 module for sonar distance sensing.

Strut 1x2
: This is just a small piece of PCB with two holes for Lego studs.

Strut 1x4
: This is just a small piece of PCB with four holes for Lego studs.

Strut 1x8
: This is just a small piece of PCB with eight holes for Lego studs.

Twin Gear Sensor Left
: This module is designed to fit into the left side of a Tamiya Twin Gear motor box and extract a quadrature signal off of one of the gears.

Twin Gear Sensor Right
: This module is designed to fit into the right side of a Tamiya Twin Gear motor box and extract a quadrature signal off of one of the gears.

Debug Robobricks

The following Debugging Modules are under active development:

Activity9
: The Activity4 Module is used to detect communication activity between two Modules.

Debug16
: The Debug16 Module is used to view up to 16 8−bit data values inside of many Module modules.

Emulate
: The Emulate board uses an 28−pin PIC16F876 with flash memory to emulate a PIC12C519, a PIC12C672, or a PIC16C505. The PIC16F876 has flash memory so it is easier to erase than the other parts which require a UV light.

Harness
: The Harness Module is used as a testing harness for testing other Modules. The Harness Module has an RS−232 connection and a connection to a single slave Module.

Tether
: The Tether Module provides a wire connection between a master Module and a computer via a standard telephone extension cord. The connection to the computer is via a standard 9−pin RS−232 connector.

Obsolete Modules

The obsolete Modules are listed below:

Activity4 (Use Activity9 instead!)
: The Activity4 Module is used to detect communication activity between two Modules.

AnalogIn4
: The AnalogIn4 Module allows for the input of up to 4 analog voltages between 0 and 5 volts with a resolution of 8 bits.

AIROD2 (Use AIROD4 instead!)
: The AIROD2 Module is used to measure distances using up to 2 the Sharp® GPD2D12 analog infrared distance measurement units.

AIROD4
: The AIROD4 Module uses the Sharp® GP2D12 analog infrared distance measurement device to measure distances between 3 and 30 centimeters. Currently, the GP2D12 seems to cost about half

what the GP2D05 used in the BIROD2 Module.

AIROD5

The AIROD4 Module uses the Sharp® GP2D12 analog infrared distance measurement device to measure distances between 3 and 30 centimeters. Up to five GP2D12's can be supported.

BIROD2 (Use BIROD5 instead!)

The BIROD2 Module is used to connect to up to 2 of the Sharp® GP2D05 IROD (InfraRed Optical Distance) measuring sensors. This version of the Sharp chip provides a single bit of information for when the sensor is within a fixed distance an object.

BIROD5

The BIROD2 Module is used to connect to up to 5 of the Sharp® GP2D05 IROD (InfraRed Optical Distance) measuring sensors. This version of the Sharp chip provides a single bit of information for when the sensor is within a fixed distance an object.

Bench (Use a master Module instead)

The Bench Module provides a way to provide power to a bunch of Modules via a standard 5 volt bench supply. It has two banana plugs to provide the connection.

BS2Hub8

The BS2Hub8 module provides a master Module controlled by the Basic Stamp II® from Parallax®. This module has a battery conntection, power switch, and 5 volt linear voltage regulator. It has hub connections for controlling up to 8 Modules. This module has morphed into the MicroBrain8 module.

Compass360

The Compass360 Module uses the 1655 analog compass module from Dinsmore Instrument Company. It can provide a resolution that is good to about 1 in 256 (1.4 degree.) The magnetic environment that a robot operates in can generate deviations of 10's of degrees however.

Hub8 (Use a master Module instead)

The Hub8 Module can connect up to 8 slave Modules.

In8 (Use InOut10 instead!)

The In8 Module will read in 8 bits of data.

InOut4 (Use InOut10 instead!)

The InOut4 Module allows for the bi−directional input or output of up to 4 signals. The direction of input or output can be changed dyamically. This Module can be used to talk to a serial bus such as I^2C.

InOut10

The InOut10 Module has 10 I/O lines that can be used for input or output. A line can be changed from input to output and back under program control. This module replaces Out10, In8, and InOut4 Modules.

IRSense2

The IRSense2 Module is used seek out IR Beacons and do simple proximity detection.

IRSense3

The IRSense3 module is used to do simple IR proximity detection in three directions.

LED4 (Use LED10 instead)

The LED4 Module provides the ability to output 4 bits to 4 on board LED's.

Light4

The Light4 Module provides a way to use inexpensive IR emitter/detector pairs to sense changes in surface reflectivity. The input level is renal flexibility in control.ad as an analog value to provide additional flexibility in control.

Motor2

The Motor2 Module can control up to two small DC motors. The motor voltage input can range from 5 volts to 24 volts. The Motor2 Module is capable of electronic breaking.

Motor3

The Motor3 Module allows for control of up to three small DC motors via pulse width modulation. The motor voltage input can range from 1 volt to 24 volts. There is no electronic breaking for the

Motor3 Module.

MotorScan

The MotorScan Module is used to provide horizontal rotational scan platform based on the Tamiya Universal Gear Box. A combination of laser head, sonar, and IR sensors can be placed on the vertical shaft and rotated around.

Out10 (Use InOut10 instead!)

The Out10 Module provides the ability to output 10 digital bits to a terminal strip.

OOPicHub10

The OOPicHub15 is an adaptor board for the OOPic by Savage Innovations. The newer OOPIC module that is pin compatible with the Parallax Basic Stamp II is now the preferred way to go

PIC16F876 (Use PIC876Hub10 instead!)

The PIC16F876 master Module is based around the PIC16F876 microcontroller from MicroChip®. This microcontroller has the ability to write into its own program memory without requiring any additional voltages or hardware.

ProtoPIC8Pin (Use ProtoPIC instead!)

The ProtoPIC8Pin Module is a prototype board for building Modules using an 8−pin PIC. ProtoPIC works with 8 and 14−pin PIC's.

Shaft2

The Shaft2 Module can keep track of the quadrature encoding of 2 shaft encoders.

Sonar1

The Sonar1 Module provides an active sonar range finder that can measure distances 5 centimeters to 3 meters. It uses some inexpensive ultrasound transducers (~$6US).

Threshold4 (Use Light4 instead!)

The Threshold4 Module consists of 4 analog voltage comparators. Each comparator compares an input voltage against a fixed voltage that is set a small potentiometer. There is one potentiometer per comparator. The resulting 4 binary bits of data are avaiable for querying.
