The RoboBricks Project

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

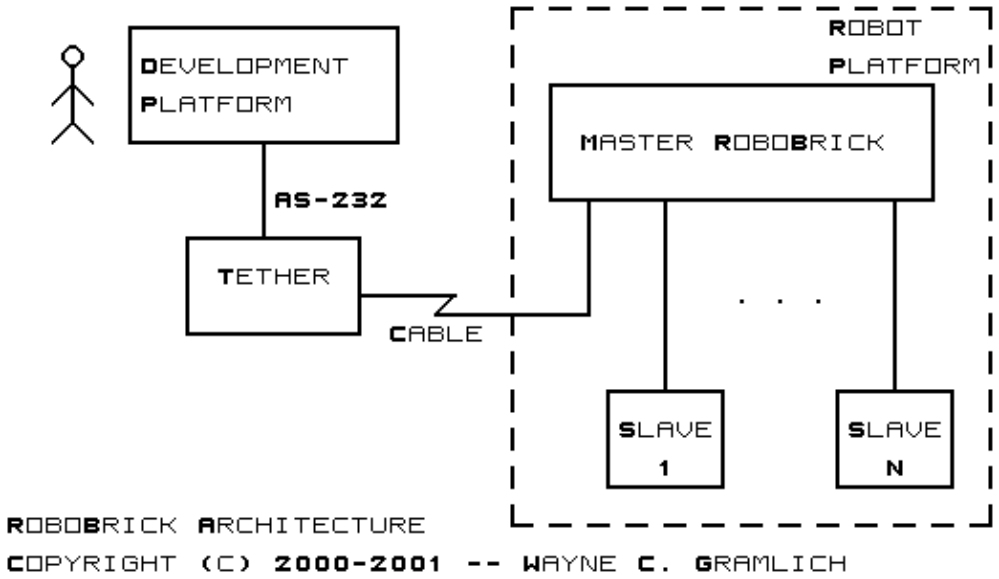
RoboBricks Introduction

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								Discussion
		7	6	5	4	3	2	1	0	
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

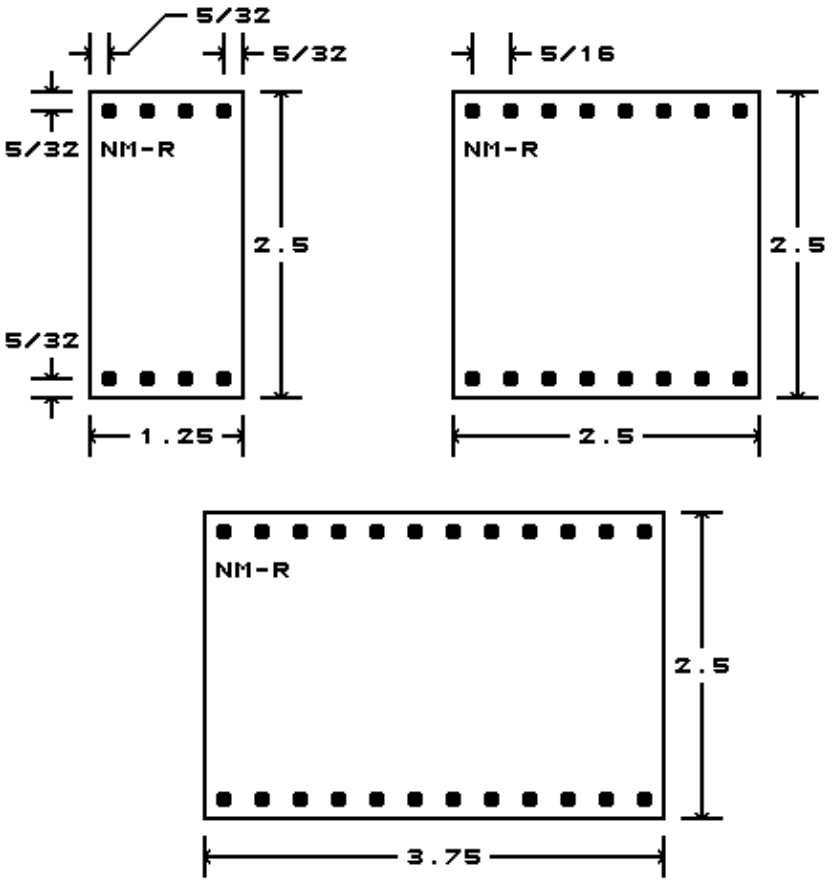
The first 8 values for this formula are shown below:

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.

This is the Revision A verion of the AnalogIn8 RoboBrick. The status of this project is work in progress.

AnalogIn8 Robobrick (Revision C)

Table of Contents

This document is also available in PDF format.

1. Introduction

The AnalogIn8 RoboBrick allows for the input of up to 4 analog voltages between 0 and 5 volts with a resolution of 8 bits.

2. Programming

The AnalogIn8 RoboBrick is continuously reading the analog inputs from its four A/D pins. The controlling program can just read the results of the digital conversion, or it can have the result down converted into a single binary bit. Each pin has has a threshold high and threshold low register that is used for the down conversion. Whenever the digital conversion exceeds the high threshold register, the down coversion results in a 1. Whenever the digital conversion is lower than the low threshold register, the down conversion results in a 0. A hysterisis effect can be introduced by having some spread between the high and low threshold values.

There AnalogIn8 RoboBrick operates in either regular mode or Vref mode. In regular mode, all four inputs are A/D converted between 0 and 5 volts. In Vref mode, input 1 is used as Vref, the highest expected input voltage, and there remaining three inputs are A/D converts betweeen 0 and Vref.

After the down coversions to binary bits, the result is 4−bits of binary data. A complement mask can be used to selectively invert individual bits in the 4−bit data.

The AnalogIn8 RoboBrick supports RoboBrick Interrupt Protocol for those lines that are being used as inputs. The interrupt pending bit is set whenever the the formula:

L&(~I) | H&I | R&(~P)&I | F&P&(~I)

is non−zero, where:

- I is the current input bits XOR'ed with the complement mask (C)
- P is the previous value of I
- L is the low mask
- H is the high mask
- R is the raising mask

- F is the falling mask

and

- ~ is bit−wise complement
- | is bit−wise OR
- & is bit−wise AND

Once the interrupt pending bit is set, it must be explicitly cleared by the user.

In addition to the common shared commands and the shared interrupt commands, the AnalogIn8 RoboBrick supports following commands:

Command	Send/ Receive	Byte Value								Discussion
		7	6	5	4	3	2	1	0	
Read Pin	Send	0	0	0	0	0	0	*b*	*b*	Read pin *bb* and respond with 8−bit value *vvvvvvvv*
	Send	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	
Read Binary Values	Send	0	0	0	0	0	1	0	0	Return the binary values *abcd* (after XOR'ing with complement mask)
	Receive	0	0	0	0	*a*	*b*	*c*	*d*	
Read Raw Binary	Send	0	0	0	0	0	1	0	1	Return the raw binary values *abcd* (no XOR with complement mask)
	Receive	0	0	0	0	*a*	*b*	*c*	*d*	
Reset	Send	0	0	0	0	0	1	1	0	Reset everything to zero
Read Complement Mask	Send	0	0	0	0	1	0	0	0	Return and return the complement mask *cccc*
	Receive	0	0	0	0	*c*	*c*	*c*	*c*	
Read High Mask	Send	0	0	0	0	1	0	0	1	Return and return the high mask *hhhh*
	Receive	0	0	0	0	*h*	*h*	*h*	*h*	
Read Low Mask	Send	0	0	0	0	1	0	1	0	Return and return the high mask *llll*
	Receive	0	0	0	0	*l*	*l*	*l*	*l*	
Read Raising Mask	Send	0	0	0	0	1	0	1	1	Return and return the raising mask *rrrr*
	Receive	0	0	0	0	*r*	*r*	*r*	*r*	
Read Falling Mask	Send	0	0	0	0	1	1	0	0	Return and return the falling mask *ffff*
	Receive	0	0	0	0	*f*	*f*	*f*	*f*	
Read Vref Mode	Send	0	0	0	0	1	1	0	1	Read and return the Vref mode bit *v*
	Receive	0	0	0	0	0	0	0	*f*	
Set Vref Mode	Send	0	0	0	0	1	1	1	*v*	Set the Vref mode to *v* (0=regular 1=Vref Mode)
Read High Threshold	Send	0	0	0	1	0	0	*b*	*b*	Read and return high threshold for pin *bb* of *hhhhhhhh*
	Receive	*h*	*h*	*h*	*h*	*h*	*h*	*h*	*h*	
Read Low Threshold	Send	0	0	0	1	0	1	*b*	*b*	Read and return low threshold for pin *bb* of *llllllll*
	Receive	*l*	*l*	*l*	*l*	*l*	*l*	*l*	*l*	
Set High Threshold	Send	0	0	0	1	1	0	*b*	*b*	Set high threshold for pin *bb* to *hhhhhhhh*
	Send	*h*	*h*	*h*	*h*	*h*	*h*	*h*	*h*	
Set Low Threshold	Send	0	0	0	1	1	1	*b*	*b*	Set low threshold for pin *bb* to *llllllll*
	Send	*l*	*l*	*l*	*l*	*l*	*l*	*l*	*l*	

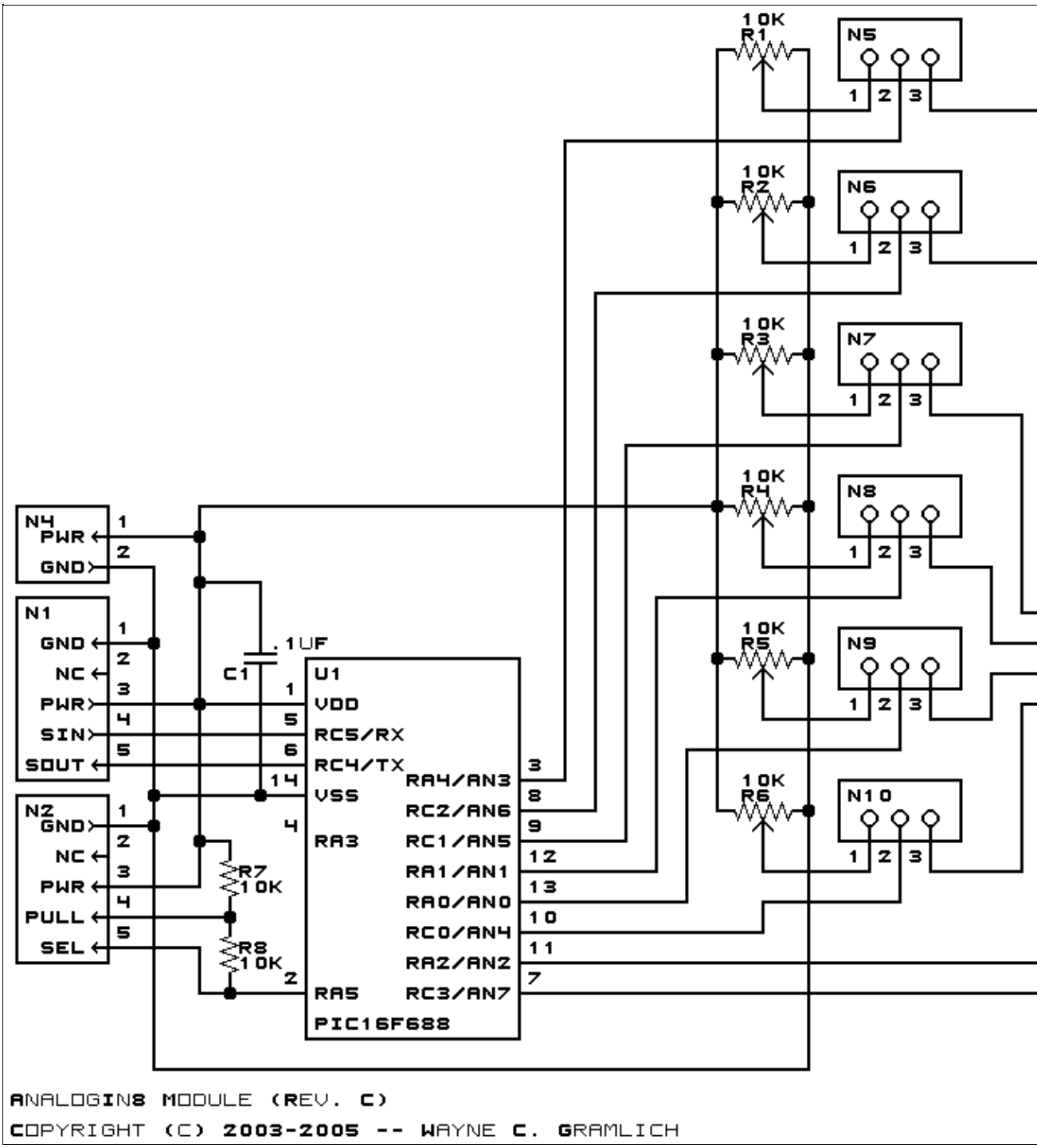
Set Complement Mask	Send	0	0	1	0	*c*	*c*	*c*	*c*	Set complement mask to *cccc*
Set High Mask	Send	0	1	0	0	*h*	*h*	*h*	*h*	Set high mask to *hhhh*
Set Low Mask	Send	0	1	0	1	*l*	*l*	*l*	*l*	Set low mask to *llll*
Set Raising Mask	Send	0	1	1	0	*r*	*r*	*r*	*r*	Set raising mask to *rrrr*
Set Falling Mask	Send	0	1	1	1	*f*	*f*	*f*	*f*	Set falling mask to *ffff*
Read Interrupt Bits	Send	1	1	1	0	1	1	1	1	Return the interrupt pending bit *p* and the interrupt enable bit *e*.
	Receive	0	0	0	0	0	0	*e*	*p*	
Set Interrupt Commands	Send	1	1	1	1	0	*c*	*c*	*c*	Set Interrupt Command *ccc*.
Shared Commands	Send	1	1	1	1	1	*c*	*c*	*c*	Execute common shared command *ccc*

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the AnalogIn8 RoboBrick is shown below:



The parts list kept in a separate file −− analogin8.ptl.

3.2 Printed Circuit Board

The printed circuit board files are listed below:

analogin8_back.png
: The solder side layer.

analogin8_front.png

The component side layer.

analogin8_artwork.png
The artwork layer.

analogin8.gbl
The RS−274X "Gerber" back (solder side) layer.

analogin8.gtl
The RS−274X "Gerber" top (component side) layer.

analogin8.gal
The RS−274X "Gerber" artwork layer.

analogin8.drl
The "Excellon" NC drill file.

analogin8.tol
The "Excellon" tool rack file.

3.3 Construction Instructions

The construction instructions are in separate file to be a little more printer friendly.

4. Software

The AnalogIn8 software is available as one of:

analogin8.ucl
The µCL source file.

analogin8.asm
The resulting human readable PIC assembly file.

analogin8.lst
The resulting human readable PIC listing file.

analogin8.hex
The resulting Intel® Hex file that can be fed into a PIC12C5xx programmer.

5. Issues

Any fabrication issues that come up will be discussed here.

This is the Revision E verion of the Compass8 module. The status of this project is finished.

Compass8 Module (Revision E)

Table of Contents

This document is also available in PDF format.

1. Introduction

The Compass8 module is used to connect to a Dinsmore Instrument Company digital compass. The compass can report the 8 bearings (N, NE, E, SE, S, SW, W, NW.)

I currently only have one of the digital compass modules and it has less than ideal behavior. My compass module works best in an envionment which has some vibration, otherwise the compass is prone to sticking and can be off by as much as 90 degrees. There is some significant hysterisis as rotation is changed; trying to steer a robot straight by aligning the robot with a bearing boundary (e.g. between N and NE) will not yield a very straight course due to this observed hysterisis effect. Of course, I may have a `lemon' module and other people might have different experiences with their moduels. However, if all you want is basic compass bearing, the Dinsmore digital compass module does seem to meet that requirement at a very reasoable cost.

If you want a more accurate compass module, you might want to try the CMPS01 magnetic compass module available at: Robot Electronics.

2. Programming

The basic operation is to send a query to the Compass8 Module to read the 3 bits of bearing data.

The Compass8 Module supports Module Interrupt Protocol. The interrupt pending bit is set whenever the the formula:

B&M

is non−zero, where:

- B is the bearing expanded out into an 8−bit vector with only 1 bit turned on,
- M is the interrupt mask, and
- & is bit−wise AND

Once the interrupt pending bit is set, it must be explicitly cleared by the user.

The Compass8 Module supports both the standard shared commands and the shared interrupt commands in

addition to the following commands:

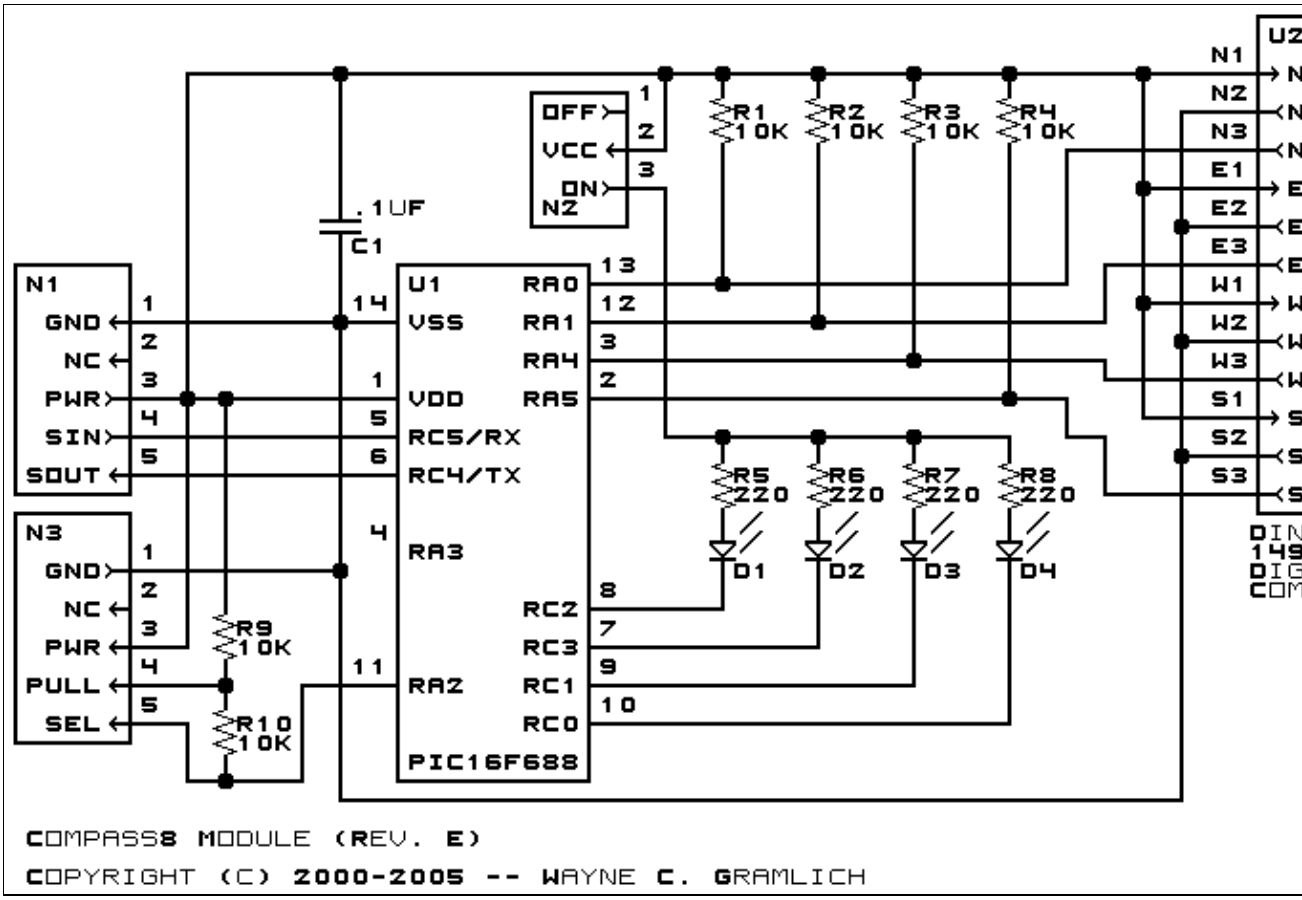
Command	Send/ Receive	Byte Value								Discussion
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
Read Bearing	Send	0	0	0	0	0	0	0	0	Return bearing *bbb* (N=000, NE=001, E=010, SE=011, S=100, SW=101, W=110, NW=111)
	Receive	0	0	0	0	0	*b*	*b*	*b*	
Read Interrupt Mask	Send	0	0	0	0	0	0	0	1	Return interrupt mask *mmmmmmmm* (N, NE, E, SE, S, SW, W, NW)
	Receive	*m*	*m*	*m*	*m*	*m*	*m*	*m*	*m*	
Read Raw	Send	0	0	0	0	0	0	1	0	Return raw data *abcd*
	Receive	0	0	0	0	*a*	*b*	*c*	*d*	
Set Interrupt Mask	Send	0	0	0	0	0	0	1	1	Set interrupt mask to *mmmmmmmm* (N, NE, E, SE, S, SW, W, NW)
	Send	*m*	*m*	*m*	*m*	*m*	*m*	*m*	*m*	
Read Interrupt Bits	Send	1	1	1	0	1	1	1	1	Return the interrupt pending bit *p* and the interrupt enable bit *e*.
	Receive	0	0	0	0	0	0	*e*	*p*	
Set Interrupt Commands	Send	1	1	1	1	0	*c*	*c*	*c*	Set Interrupt Command *ccc*.
Shared Commands	Send	1	1	1	1	1	*c*	*c*	*c*	Execute Shared Command *ccc*.

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the Compass8 Module is shown below:



The parts list kept in a separate file −− compass8.ptl.

3.2 Printed Circuit Board

The printed circuit board files are listed below:

compass8_back.png
: The solder side layer.

compass8_front.png
: The component side layer.

compass8_artwork.png
: The artwork layer.

compass8.gbl
: The RS−272X "Gerber" back (solder side) layer.

compass8.gtl
: The RS−272X "Gerber" top (component side) layer.

compass8.gal
: The RS−272X "Gerber" artwork layer.

compass8.drl
: The "Excellon" NC drill file.

compass8.tol
: The "Excellon" tool rack file.

4. Software

The Compass8 software is available as one of:

compass8.ucl
: The µCL source file.

compass8.asm
: The resulting human readable PIC assembly file.

compass8.lst
: The resulting human readable PIC listing file.

compass8.hex
: The resulting Intel® Hex file.

5. Issues

Any fabrication issues that come up are listed here.

This is the Revision D verion of the Digital8 module. The status of this project is finished.

Digital8 Module (Revision D)

Table of Contents

This document is also available in PDF format.

1. Introduction

The Digital8 module provides the ability to input and output 8 bits of digital data. The direction of each bit can be changed under program control.

2. Programming

The programmer can download a complement mask to cause any of the bits to be complemented prior to reading.

The Digital8 module supports the Interrupt Protocol. The interrupt pending bit is set whenever the the formula:

L&(~I) | H&I | R&(~P)&I | F&P&(~I)]

is non−zero, where:

- I is the current input bits XOR'ed with the complement mask (C)
- P is the previous value of I
- L is the low mask
- H is the high mask
- R is the raising mask
- F is the falling mask

and

- ~ is bit−wise complement
- | is bit−wise OR
- & is bit−wise AND

Once the interrupt pending bit is set, it must be explicitly cleared by the user.

The Digital8 module supports both the standard shared commands and the shared interrupt commands in addition to the following commands:

Command	Send/ Receive	Byte Value								Discussion
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
Read Inputs	Send	0	0	0	0	0	0	0	0	Return 8−bits of input *iiii iiii* (after XOR'ing with complement mask)
	Receive	*i*	*i*	*i*	*i*	*i*	*i*	*i*	*i*	
Read Outputs	Send	0	0	0	0	0	0	0	1	Return 8−bits of the outputs *oooo oooo* (after XOR'ing with complement mask.)
	Receive	*o*	*o*	*o*	*o*	*o*	*o*	*o*	*o*	
Read Complement Mask	Send	0	0	0	0	0	0	1	0	Return 8−bits of complement mask *cccc cccc*
	Receive	*c*	*c*	*c*	*c*	*c*	*c*	*c*	*c*	
Read Direction Mask	Send	0	0	0	0	0	0	1	1	Return 8−bits of direction mask *dddd dddd*
	Receive	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	
Read Low Mask	Send	0	0	0	0	0	1	0	0	Return 8−bits of low mask *llll llll*
	Receive	*l*	*l*	*l*	*l*	*l*	*l*	*l*	*l*	
Read High Mask	Send	0	0	0	0	0	1	0	1	Return 8−bits of the high mask *hhhh hhhh*
	Receive	*h*	*h*	*h*	*h*	*h*	*h*	*h*	*h*	
Read Rising Mask	Send	0	0	0	0	0	1	1	0	Return 8−bits of the rising mask *rrrr rrrr*
	Receive	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	
Read Falling Mask	Send	0	0	0	0	0	1	1	1	Return 8−bits of the falling mask *ffff ffff*
	Receive	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	
Read Raw	Send	0	0	0	0	1	0	0	0	Return 8−bits of raw input data *rrrr rrrr* (without XOR'ing with complement mask)
	Receive	r	r	r	*r*	*r*	*r*	*r*	*r*	
Reset Outputs	Send	0	0	0	1	0	0	0	0	Set all 8 bits of outputs to 0 (then XOR with complement mask).
Set Outputs	Send	0	0	0	1	0	0	0	1	Set output bits to *oooo oooo*.
	Send	*o*	*o*	*o*	*o*	*o*	*o*	*o*	*o*	
Set Complement Mask	Send	0	0	0	1	0	0	1	0	Set 8−bits of complement mask to *cccc cccc*
	Send	*c*	*c*	*c*	*c*	*c*	*c*	*c*	*c*	
Set Direction Mask	Send	0	0	0	1	0	0	1	1	Set 8−bits of direction mask to *dddd dddd* 1=input; 0=output
	Send	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	
Set Low Mask	Send	0	0	0	1	0	1	0	0	Set 8−bits of low mask to *llll llll*
	Send	*l*	*l*	*l*	*l*	*l*	*l*	*l*	*l*	
Set High Mask	Send	0	0	0	1	0	1	0	1	Set 8−bits of the high mask to *hhhh hhhh*
	Send	*h*	*h*	*h*	*h*	*h*	*h*	*h*	*h*	
Set Rising Mask	Send	0	0	0	1	0	1	1	0	Set 8−bits of the rising mask to *rrrr rrrr*
	Send	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	
Set Falling Mask	Send	0	0	0	1	0	1	1	1	Set 8−bits of the falling mask to *ffff ffff*
	Send	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	
Set Outputs Raw	Send	0	0	0	1	1	0	0	0	Set 8−bits to *oooo oooo* with no complement mask.
	Send	*o*	*o*	*o*	*o*	*o*	*o*	*o*	*o*	
Reset Everything	Send	0	0	0	1	1	0	0	1	Reset all registers to 0 and set direction bits to 1 (input)
Set Output Bit	Send	0	0	1	0	*v*	*b*	*b*	*b*	Set output bit *bbbb* to *v*

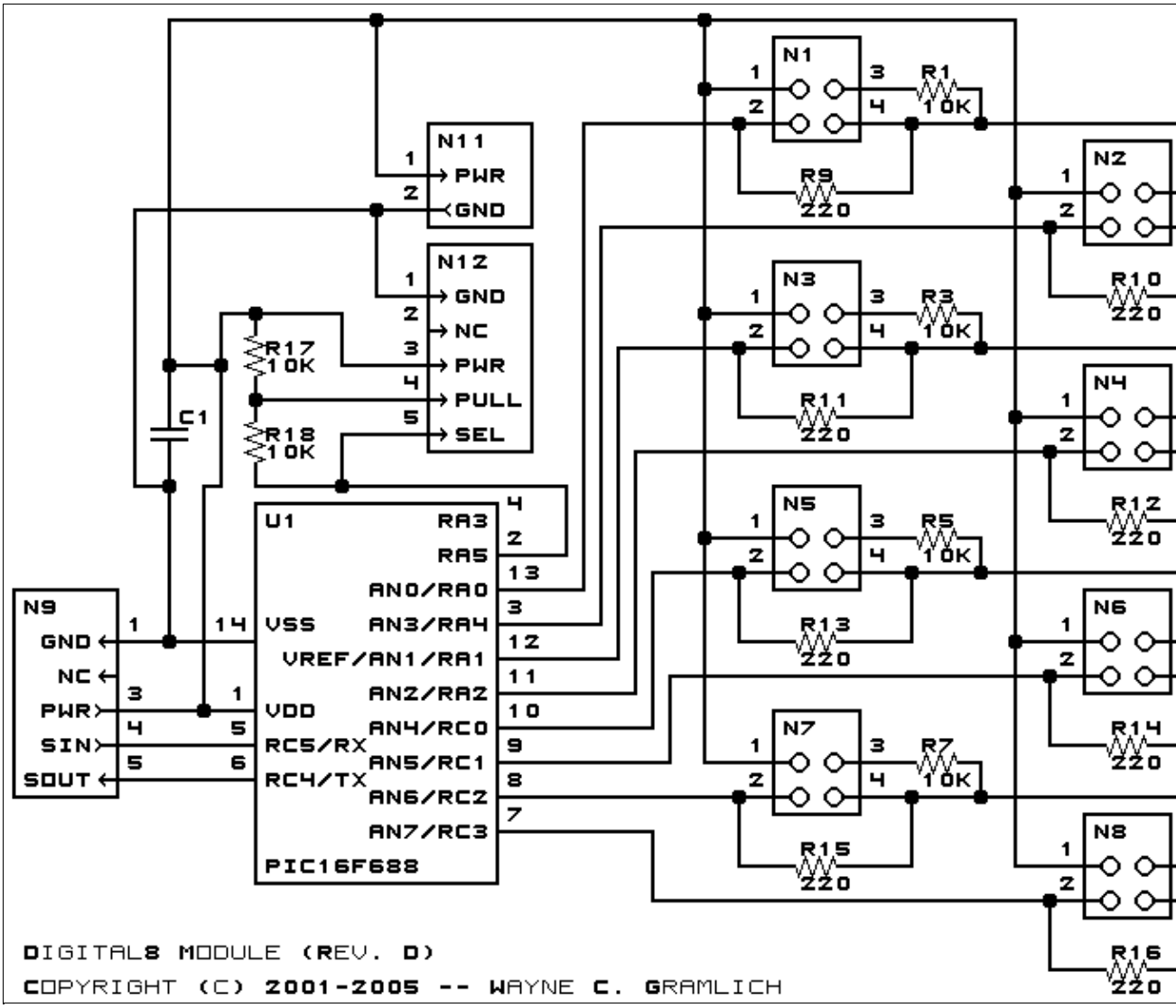
Set Outputs Low	Send	0	1	0	0	*l*	*l*	*l*	*l*	Set low order 4−bits of Outputs to *llll* and then XOR complement mask
Set Outputs High	Send	0	1	0	1	*h*	*h*	*h*	*h*	Set high order 4−bits of Outputs to *hhhh* and and then XOR complement mask
Set Direction Low	Send	0	1	1	0	*l*	*l*	*l*	*l*	Set low order 4−bits of direction to *llll*.
Set Direction High	Send	0	1	1	1	*h*	*h*	*h*	*h*	Set high order 4−bits of direction to *hhhh*.
Set Interrupt Commands	Send	1	1	1	1	0	*c*	*c*	*c*	Set Interrupt Command *ccc*.
Shared Commands	Send	1	1	1	1	1	*c*	*c*	*c*	Execute Shared Command *ccc*

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the Digital8 module is shown below:



The parts list kept in a separate file −− digital8.ptl.

3.2 Printed Circuit Board

The printed circuit files are listed below:

digital8_back.png
: The solder side layer.

digital8_front.png
: The component side layer.

digital8_artwork.png
: The artwork layer.

digital8.gbl
: The RS−274X "Gerber" back (solder side) layer.

digital8.gtl
: The RS−274X "Gerber" top (component side) layer.

digital8.gal
: The RS−274X "Gerber" artwork layer.

digital8.drl
: The "Excellon" NC drill file.

digital8.tol
: The "Excellon" tool rack file.

3.3 Construction Instructions

The construction Instructions are located in a separate file to be a little more printer friendly.

4. Software

The Digital8 software is available as one of:

digital8.ucl
: The µCL source file.

digital8.asm
: The resulting human readable PIC assembly file.

digital8.lst
: The resulting human readable PIC listing file.

digital8.hex
: The resulting Intel® Hex file.

5. Issues

Any fabrication issues will be listed here.

This is revision E of the DualMotor1Amp module. The status of this project is work in progress.

DualMotor1Amp Module (Revision E)

Table of Contents

This document is also available as a PDF document.

1. Introduction

The DualMotor1Amp module allows for control of up to two small DC motors via pulse width modulation. Power is provided via an external terminal block. For low voltage motors, an on−board 5 volt voltage regulator can be optionally used to limit the motor voltage to 5 volts.

2. Programming

The DualMotor1Amp module can control up to two motors called 0 and 1 respectively. Each motor has a condition, power mode, direction, and speed. The condition is either on or off (i.e. freewheel.) The two power modes are pulsed and continous. The two directions are forward and backward. The speed is a number between 0 and 255 inclusive. There is one additional variable associated with each motor called ramp and a few additional variables that are shared between the two motors.

Pulsed mode is standard motor control via pulse width modulation (PWM.) When the speed is 0, no pulses are sent to the motor. When the speed is 255, the motor is full on. When the speed is 128, 50% duty cycle pulses are sent to the motor. The direction bit, controls what direction current is pulsed into the motor.

In continuous mode, power is continuously applied either forward or backward through the motor. In continuous mode, when the speed is 128, 50% duty cycle pulses are sent to the motor, where half the cycle is sends current forward through the motor and the other half is sends current backward through the motor (thereby cancelling out and resulting in a rotational speed of 0.) While continous mode consumes more power than pulsed mode, it sometimes provides better motor speed control at slow speeds.

The ramp variable is used to slow down the rate at which motor speeds are changed. When the ramp variable is non−zero, it specifies the rate at which motor speed changes (i.e. the speed ramp.) The ramp rate is measured in ticks (1/3 of a bit time at 2400 baud, or 1/7200, or 138µS. A ramp rate of 1, means the pulse widths will be changed every 138µS. A ramp rate of 100 means the pulse widths will be changed every 100 × 138µS or every 13.8mS. This allows the user to slowly speed up and slow down the motor. Please note, that ramp only applies to speed, changing the motor direction is immediate. (Sorry!)

For safety reasons, you might want the motors to shut off if the controlling program crashes. This is accomplished with a variable called the failsafe delay variable which is shared between both motors. When the failsafe delay variable is set to a non−zero value, it causes another variable called the failsafe counter to be

initialized to the same value. Every 256 ticks (= 256 × 138µS = 35.5mS), the failsafe counter is decremented. If the failsafe counter ever decrements to 0, it immediately turns off both motors without any ramping. Every time a speed command is sent to the DualMotor1Amp module, the failsafe counter is reinitialed to contain the failsafe delay variable. Thus, by occasionally sending commands that set the speed of either motor, the failsafe counter can be kept non−zero. Alternatively, there is a command that just reinitializes the failsafe counter without affecting the speed. The maximum amount of time between commands that reset the failsafe counter is 255 × 35.5mS or approximately 9 seconds. If the controlling program crashes, it will stop sending commands to the DualMotor1Amp module and eventually, the failsafe counter will decrement to zero and stop both motors. There is yet a third variable called the failsafe error counter that is incremented each time a failsafe shut down occurs. The failsafe error counter can be read with yet another command. Lastly, both motors can be restarted by simply sending another command that sets the speed of either motor.

Finally, there is one other variable that is shared between the two motors called the prescaler. The prescaler is 3−bits wide and controls duty cycle width of the pulses are sent to the motor. The table below summarizes the prescaler to duty cycle relationship:

Prescaler	**Duty Cycle Width**
000	.5µS
001	1µS
010	2µS
011	4µS
100	8µS
101	16µS
110	32µS
111	64µS

The DualMotor1Amp commands are summarized in the table below:

Command	**Send/ Receive**	**Byte Value**								**Discussion**
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
Set Quick	Send	0	0	*h*	*h*	*h*	*h*	*d*	*m*	Set motor *m* speed to *hhhh hhhh* and direction to *d* (0=forward, 1=backward).
Set Low	Send	0	1	*l*	*l*	*l*	*l*	*d*	*m*	Set low order 4 bits of motor *m* speed to *ll* and direction to *d* (0=forward, 1=backward).
Set Ramp	Send	1	0	0	0	0	0	0	*m*	Set the ramp for motor *m* to *rrrrrrrr* (00000000=no ramp (default)).
	Send	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	
Set Failsafe	Send	1	0	0	0	0	0	1	0	Set the failsafe delay variable to *ffffffff* (00000000=off (default)).
	Send	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	
Reset Failsafe	Send	1	0	0	0	0	0	1	1	Reset the failsafe counter to the failsafe delay variable.
Set Speed	Send	1	0	0	0	0	1	*d*	*m*	Set motor *m* to speed *ssssssss* and direction to *d*.
	Send	*s*	*s*	*s*	*s*	*s*	*s*	*s*	*s*	
Set Mode	Send	1	0	0	0	1	0	*x*	*m*	Set motor *m* mode to *x* (0=pulsed

										(default), 1=continuous).
Set Direction	Send	1	0	0	0	1	1	*d*	*m*	Set motor *m* direction to *d* (0=forward (default), 1=reverse).
Set Prescaler	Send	1	0	0	1	0	*p*	*p*	*p*	Set prescaler to *ppp* (000=fast, 111=slow (default)).
Read Failsafe	Send	1	0	0	1	1	0	0	0	Read the return the failsafe delay variable *ffffffff*.
	Receive	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	
Read Prescaler	Send	1	0	0	1	1	0	0	1	Read the return the prescaler *ppp*.
	Receive	0	0	0	0	0	*p*	*p*	*p*	
Read Speed	Send	1	0	0	1	1	0	1	*m*	Read the return the speed *ssssssss* for motor *m*.
	Receive	*s*	*s*	*s*	*s*	*s*	*s*	*s*	*s*	
Read Mode/Direction	Send	1	0	0	1	1	1	0	*m*	Read the mode *x* (0=pulsed, 1=continuous) and direction *d* (0=forward, 1=reverse) for motor *m*.
	Receive	0	0	0	0	0	0	*x*	*d*	
Read Ramp	Send	1	0	0	1	1	1	1	*m*	Read and return the ramp *rrrrrrrr* for motor *m*.
	Receive	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	
Read Failsafe Errors	Send	1	0	1	0	0	0	0	0	Read and return the failsafe error counter *eeeeeeee*. Reset the counter.
	Receive	*e*	*e*	*e*	*e*	*e*	*e*	*e*	*e*	
Read Failsafe Counter	Send	1	0	1	0	0	0	0	1	Read and return the failsafe counter *cccccccc*.
	Receive	*c*	*c*	*c*	*c*	*c*	*c*	*c*	*c*	
Read Actual Speed	Send	1	0	1	0	0	0	1	*m*	Read and return the actual speed for motor *m*
	Receive	*e*	*e*	*e*	*e*	*e*	*e*	*e*	*e*	
On/Off	Send	1	0	1	0	0	1	*o*	*m*	Set motor *m* to condition *o* (0=off 1=on)
Reset	Send	1	0	1	0	1	0	0	0	Reset the entire motor controller
Shared Commands	Send	1	1	1	1	1	*c*	*c*	*c*	Execute shared command *ccc*.

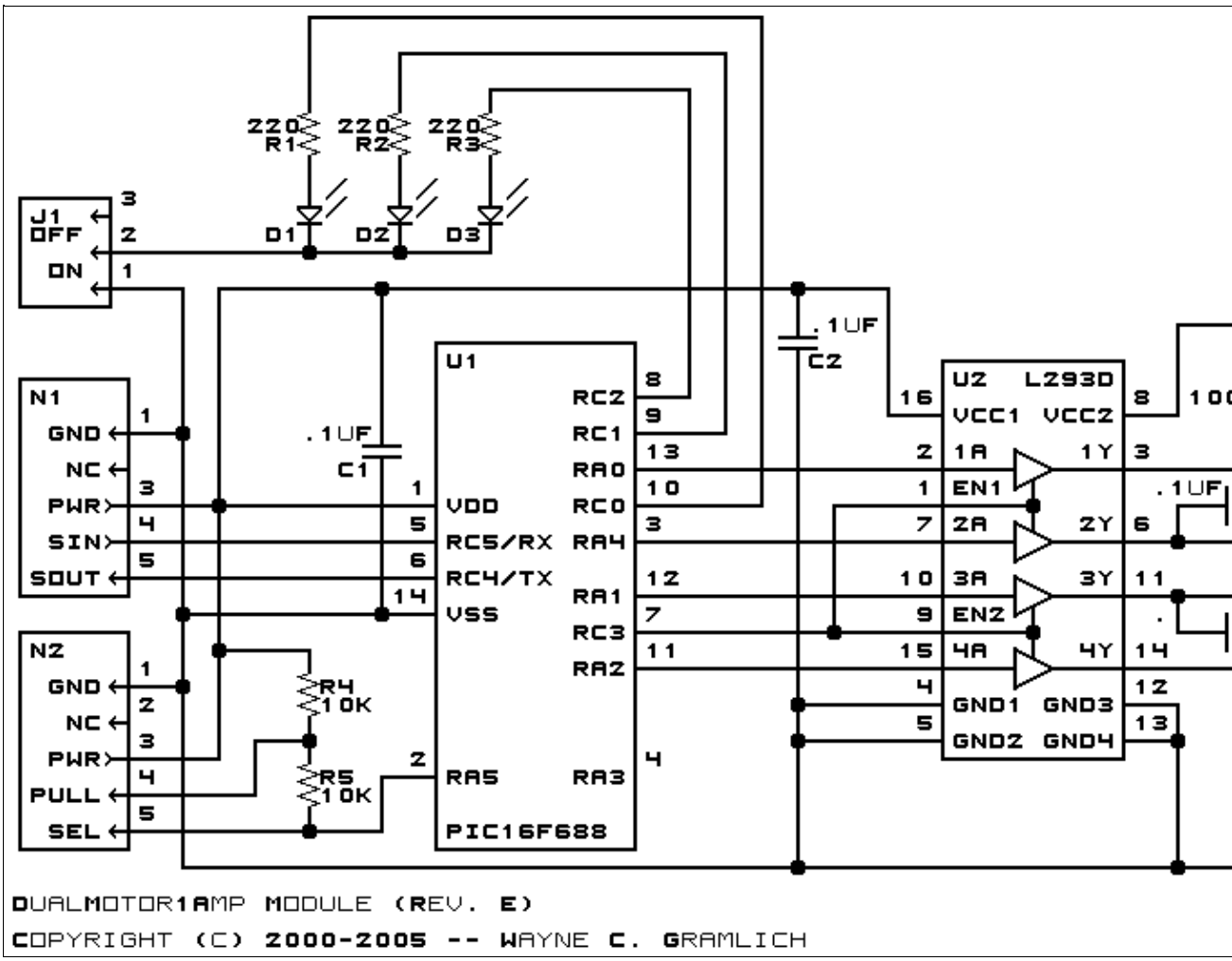
On power up, the DualMotor1Amp module sets all variables to zero. The motor modes default to pulsed forward.

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the DualMotor1Amp module is shown below:



The parts list kept in a separate file −− dualmotor1amp.ptl.

3.2 Printed Circuit Board

The printed circuit files are listed below:

dualmotor1amp_back.png
: The solder side layer.

dualmotor1amp_front.png
: The component side layer.

dualmotor1amp_artwork.png
: The artwork layer.

dualmotor1amp.gbl
: The RS−274X "Gerber" back (solder side) layer.

dualmotor1amp.gtl
: The RS−274X "Gerber" top (component side) layer.

dualmotor1amp.gal
: The RS−274X "Gerber" artwork layer.

dualmotor1amp.drl
: The "Excellon" NC drill file.

dualmotor1amp.tol

The "Excellon" NC drill rack file.

4. Software

The DualMotor1Amp software is available as one of:

dualmotor1amp.ucl
: The µCL source file.

dualmotor1amp.asm
: The resulting human readable PIC assembly file.

dualmotor1amp.lst
: The resulting human readable PIC listing file.

dualmotor1amp.hex
: The resulting Intel® Hex file.

5. Issues

The following issues need to be considered:

- Consider using the TI 754410 instead of the LM293.

This is the Revision A verion of the IRDistance8 Module. The status of this project is finished.

IRDistance8 Module (Revision A)

Table of Contents

This document is also available in PDF format.

1. Introduction

The IRDistance8 Module is used to connect up to 5 Sharp® GP2D12 IROD (InfraRed Optical Distance) measuring sensors. The GP2D12 module provides an analog voltage that is proportional to the distance (although not linearly.)

2. Programming

The IRDistance8 Module can enable zero, one or more of the AIROD's. For the ones that are enabled, it continuously reads the distance values. To conserve power, only one AIROD is powered up at a time.

The IRDistance8 Module supports Module Interrupt Protocol for those lines that are being used as inputs. The interrupt pending bit is set whenever the the formula:

L&(~I) | H&I | R&(~P)&I | F&P&(~I)

is non−zero, where:

- I is the current input bits XOR'ed with the complement mask (C)
- P is the previous value of I
- L is the low mask
- H is the high mask
- R is the raising mask
- F is the falling mask

and

- ~ is bit−wise complement
- | is bit−wise OR
- & is bit−wise AND

Once the interrupt pending bit is set, it must be explicitly cleared by the user.

In addition to the common shared commands and the shared interrupt commands, the AnalogIn4 Module

supports following commands:

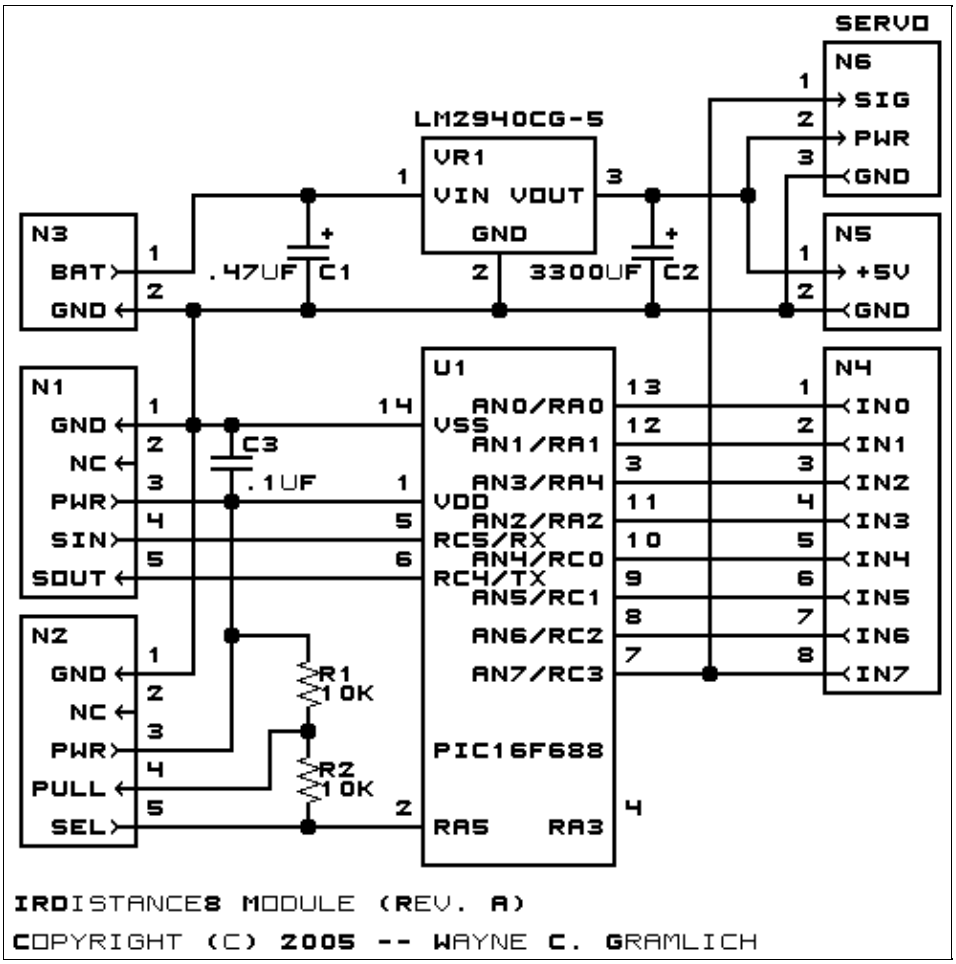
Command	Send/ Receive	Byte Value								Discussion
		7	6	5	4	3	2	1	0	
Read Distance	Send	0	0	0	0	0	0	0	*b*	Read IROD *b* and respond with 8−bit value *dddddddd*
	Send	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	
Read Binary Values	Send	0	0	0	0	0	0	1	0	Return the binary values *ab* (after XOR'ing with complement mask)
	Receive	0	0	0	0	0	0	*a*	*b*	
Read Raw Binary	Send	0	0	0	0	0	0	1	1	Return the raw binary values *ab* (no XOR with complement mask)
	Receive	0	0	0	0	0	0	*a*	*b*	
Reset	Send	0	0	0	0	0	1	0	0	Reset everything to zero
Read Enable Bit	Send	0	0	0	0	0	1	0	1	Read and return the enable bit *e*
	Receive	0	0	0	0	0	0	0	*e*	
Set Enable Bit	Send	0	0	0	0	0	1	1	*e*	Set enable bit to *e*
Read Complement Mask	Send	0	0	0	0	1	0	0	0	Return and return the complement mask *cccc*
	Receive	0	0	0	0	0	0	*c*	*c*	
Read High Mask	Send	0	0	0	0	1	0	0	1	Return and return the high mask *hh*
	Receive	0	0	0	0	0	0	*h*	*h*	
Read Low Mask	Send	0	0	0	0	1	0	1	0	Return and return the low mask *ll*
	Receive	0	0	0	0	0	0	*l*	*l*	
Read Raising Mask	Send	0	0	0	0	1	0	1	1	Return and return the raising mask *rr*
	Receive	0	0	0	0	0	0	*r*	*r*	
Read Falling Mask	Send	0	0	0	0	1	1	0	0	Return and return the falling mask *ff*
	Receive	0	0	0	0	0	0	*f*	*f*	
Read High Threshold	Send	0	0	0	1	0	0	0	*b*	Read and return high threshold for pin *b* of *hhhhhhhh*
	Receive	*h*	*h*	*h*	*h*	*h*	*h*	*h*	*h*	
Read Low Threshold	Send	0	0	0	1	0	0	1	*b*	Read and return low threshold for pin *bb* of *lllllll*
	Receive	*l*	*l*	*l*	*l*	*l*	*l*	*l*	*l*	
Set High Threshold	Send	0	0	0	1	0	1	0	*b*	Set high threshold for pin *b* to *hhhhhhhh*
	Send	*h*	*h*	*h*	*h*	*h*	*h*	*h*	*h*	
Set Low Threshold	Send	0	0	0	1	0	1	1	*b*	Set low threshold for pin *b* to *lllllll*
	Send	*l*	*l*	*l*	*l*	*l*	*l*	*l*	*l*	
Set Complement Mask	Send	0	0	1	0	0	0	*c*	*c*	Set complement mask to *cc*
Set High Mask	Send	0	0	1	0	0	1	*h*	*h*	Set high mask to *hh*
Set Low Mask	Send	0	0	1	0	1	0	*l*	*l*	Set low mask to *ll*
Set Raising Mask	Send	0	0	1	0	1	1	*r*	*r*	Set raising mask to *rr*
Set Falling Mask	Send	0	0	1	1	0	0	*f*	*f*	Set falling mask to *ff*
Read Interrupt Bits	Send	1	1	1	0	1	1	1	1	Return the interrupt pending bit *p* and the interrupt enable bit *e*.
	Receive	0	0	0	0	0	0	*e*	*p*	
Set Interrupt Commands	Send	1	1	1	1	0	*c*	*c*	*c*	Set Interrupt Command *ccc*.
Shared Commands	Send	1	1	1	1	1	*c*	*c*	*c*	Execute common shared command *ccc*

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the IRDistance8 Module is shown below:



The parts list kept in a separate file −− irdistance8.ptl.

3.2 Printed Circuit Board

The printed circuit board files are listed below:

irdistance8_back.png
: The solder side layer.

irdistance8_front.png
: The component side layer.

irdistance8_artwork.png
: The artwork layer.

irdistance8.gbl

The RS−272X "Gerber" back (solder side) layer.

irdistance8.gtl
The RS−272X "Gerber" top (component side) layer.

irdistance8.gal
The RS−272X "Gerber" artwork layer.

irdistance8.drl
The "Excellon" NC drill file.

irdistance8.tol
The "Excellon" tool rack file.

4. Software

The software for the IRDistance8 is listed below:

irdistance8.ucl
The µCL file for IRDistance8.

irdistance8.asm
The assembly file for IRDistance8.

irdistance8.hex
The Intel® Hex file.

irdistance8.lst
The listing file for IRDistance8.

5. Issues

The following issues need to be addressed:

- U1 needs to be moved up by at least .05", preferably .10". The chip interferres with VR1.
- Contemplate rotating VR1 by 90 degrees to provide more space.
- Contemplate moving C2 up between N4 and U1 to provide more space for VR1. C2 can be moved right by .05".
- Think about moving R1 and R2 up a little.

This is the Revision D verion of the IREdge4 module. The status of this project is finished.

IREdge4 Module (Revision D)

Table of Contents

This document is also available in PDF format.

1. Introduction

The IREdge4 module can connect to up to 4 Photo Sensors (combined light emitter with photodetector.) The inputs are done using analog to digital converters rather than just binary inputs. There are 4 potentiometers to control the current throught the light emitters and 4 pententionmeters to control the gain of the returned signal.

2. Programming

The IREdge4 module is continuously reading the analog inputs from its four A/D pins. The controlling program can just read the results of the digital conversion, or it can have the result down converted into a single binary bit. Each pin has has a threshold high and threshold low register that is used for the down conversion. Whenever the digital conversion exceeds the high threshold register, the down coversion results in a 1. Whenever the digital conversion is lower than the low threshold register, the down conversion results in a 0. A hysterisis effect can be introduced by having some spread between the high and low threshold values.

After the down coversions to binary bits, the result is 4−bits of binary data. A complement mask can be used to selectively invert individual bits in the 4−bit data.

The IREdge4 module supports the Interrupt Protocol for those lines that are being used as inputs. The interrupt pending bit is set whenever the the formula:

L&(~I) | H&I | R&(~P)&I | F&P&(~I)

is non−zero, where:

- I is the current input bits XOR'ed with the complement mask (C)
- P is the previous value of I
- L is the low mask
- H is the high mask
- R is the rising mask
- F is the falling mask

and

- ~ is bit−wise complement
- | is bit−wise OR
- & is bit−wise AND

Once the interrupt pending bit is set, it must be explicitly cleared by the user.

In addition to the common shared commands and the shared interrupt commands, the IREdge4 RoboBrix supports following commands:

Command	Send/ Receive	Byte Value								Discussion
		7	6	5	4	3	2	1	0	
Read Pin	Send	0	0	0	0	0	0	*b*	*b*	Read pin *bb* and respond with 8−bit value *vvvvvvvv*
	Receive	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	
Read Binary Values	Send	0	0	0	0	0	1	0	0	Return the binary values *abcd* (after XOR'ing with complement mask)
	Receive	0	0	0	0	*a*	*b*	*c*	*d*	
Read Raw Binary	Send	0	0	0	0	0	1	0	1	Return the raw binary values *abcd* (no XOR with complement mask)
	Receive	0	0	0	0	*a*	*b*	*c*	*d*	
Reset	Send	0	0	0	0	0	1	1	0	Reset everything to zero
Read Complement Mask	Send	0	0	0	0	1	0	0	0	Return the complement mask *cccc*
	Receive	0	0	0	0	*c*	*c*	*c*	*c*	
Read High Mask	Send	0	0	0	0	1	0	0	1	Return the high mask *hhhh*
	Receive	0	0	0	0	*h*	*h*	*h*	*h*	
Read Low Mask	Send	0	0	0	0	1	0	1	0	Return the high mask *llll*
	Receive	0	0	0	0	*l*	*l*	*l*	*l*	
Read Rising Mask	Send	0	0	0	0	1	0	1	1	Return the rising mask *rrrr*
	Receive	0	0	0	0	*r*	*r*	*r*	*r*	
Read Falling Mask	Send	0	0	0	0	1	1	0	0	Return the falling mask *ffff*
	Receive	0	0	0	0	*f*	*f*	*f*	*f*	
Read High Threshold	Send	0	0	0	1	0	0	*b*	*b*	Return high threshold for pin *bb* of *hhhhhhhh*
	Receive	*h*	*h*	*h*	*h*	*h*	*h*	*h*	*h*	
Read Low Threshold	Send	0	0	0	1	0	1	*b*	*b*	Return low threshold for pin *bb* of *llllllll*
	Receive	*l*	*l*	*l*	*l*	*l*	*l*	*l*	*l*	
Set High Threshold	Send	0	0	0	1	1	0	*b*	*b*	Set high threshold for pin *bb* to *hhhhhhhh*
	Send	*h*	*h*	*h*	*h*	*h*	*h*	*h*	*h*	
Set Low Threshold	Send	0	0	0	1	1	1	*b*	*b*	Set low threshold for pin *bb* to *llllllll*
	Send	*l*	*l*	*l*	*l*	*l*	*l*	*l*	*l*	
Set Complement Mask	Send	0	0	1	0	*c*	*c*	*c*	*c*	Set complement mask to *cccc*
Set High Mask	Send	0	1	0	0	*h*	*h*	*h*	*h*	Set high mask to *hhhh*
Set Low Mask	Send	0	1	0	1	*l*	*l*	*l*	*l*	Set low mask to *llll*
Set Rising Mask	Send	0	1	1	0	*r*	*r*	*r*	*r*	Set rising mask to *rrrr*
Set Falling Mask	Send	0	1	1	1	*f*	*f*	*f*	*f*	Set falling mask to *ffff*
Read Interrupt Bits	Send	1	1	1	0	1	1	1	1	Return the interrupt pending bit *p* and the interrupt enable bit *e*.
	Receive	0	0	0	0	0	0	*e*	*p*	

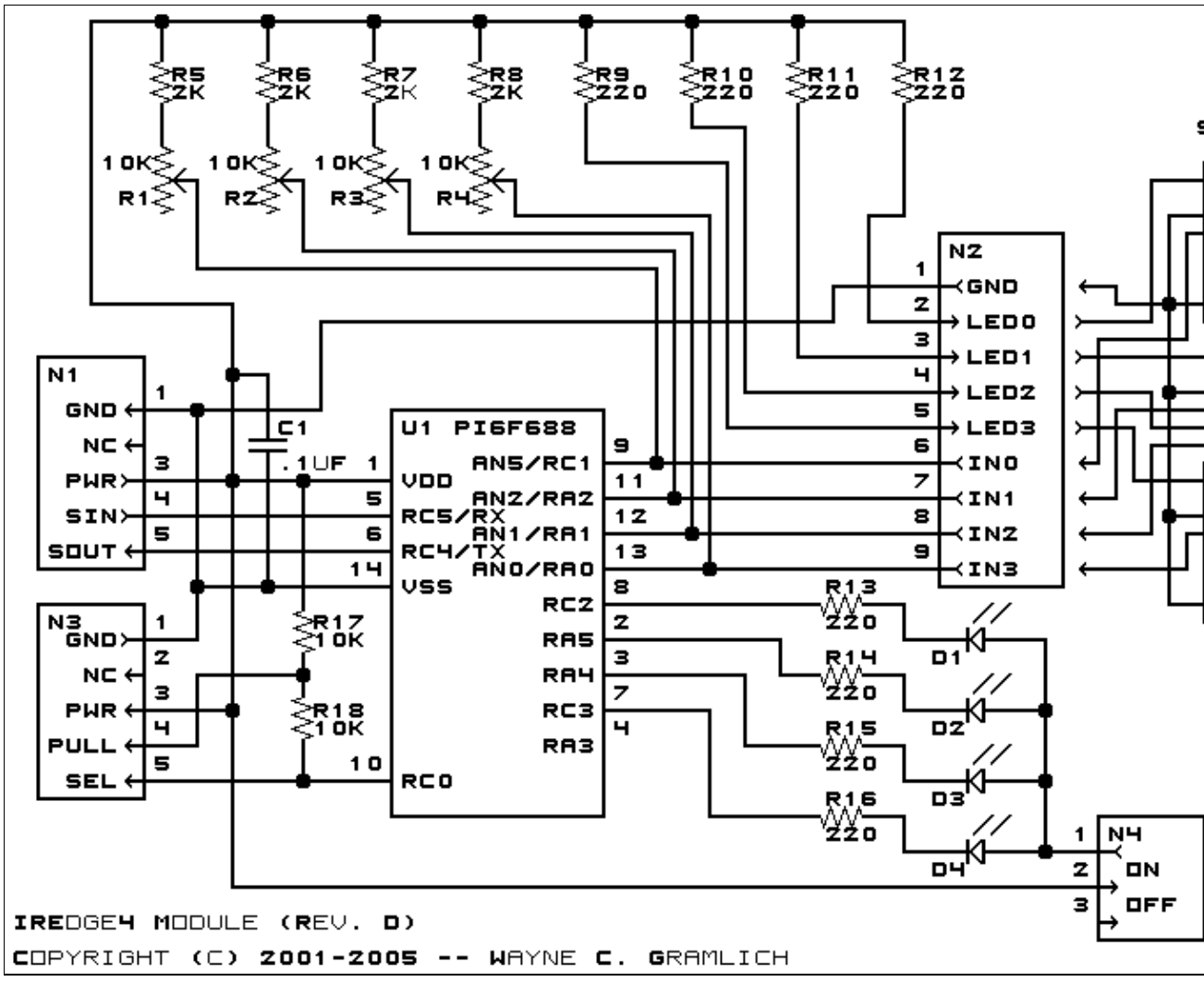
Set Interrupt Commands	Send	1	1	1	1	0	*c*	*c*	*c*	Set Interrupt Command *ccc*.
Shared Commands	Send	1	1	1	1	1	*c*	*c*	*c*	Execute common shared command *ccc*

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the IREdge4 RoboBrix is shown below:



The parts list kept in a separate file −− iredge4.ptl.

3.2 Printed Circuit Board

The printed circuit board files are listed below:

iredge4_back.png
: The solder side layer.

iredge4_front.png
: The component side layer.

iredge4_artwork.png
: The artwork layer.

iredge4.gbl
: The RS−272X "Gerber" back (solder side) layer.

iredge4.gtl
: The RS−272X "Gerber" top (component side) layer.

iredge4.gal
: The RS−272X "Gerber" artwork layer.

iredge4.drl
: The "Excellon" NC drill file.

iredge4.tol
: The "Excellon" tool rack file.

4. Software

The IREdge4 software is available as one of:

iredge4.ucl
: The µCL source file.

iredge4.asm
: The resulting human readable PIC assembly file.

iredge4.lst
: The resulting human readable PIC listing file.

iredge4.hex
: The resulting Intel® Hex file.

5. Issues

Any fabrication iusses will be listed here.

This is the Revision A verion of the IRremote1 module. The status of this project is work in progress.

IRRemote1 Robobrick (Revision C)

Table of Contents

This document is also available as a PDF document.

1. Introduction

The IRRemote1 module is used to send and received IR signals. It currently takes signals from Sony IR remotes. The transmisssion facility is a little underdeveloped (i.e. non−existant) at the moment. The IR Receiver is the Sharp GP1U26X.

2. Programming

The basic operation is to send a query to the IRRemote1 RoboBrick to return the last two bytes of IR remote command.

The IRRemote1 module supports RoboBrick Interrupt Protocol. The interrupt pending bit is set whenever a command has been received. Once the interrupt pending bit is set, it must be explicitly cleared by the user.

The IRRemote1 RoboBrick supports both the standard shared commands and the shared interrupt commands in addition to the following commands:

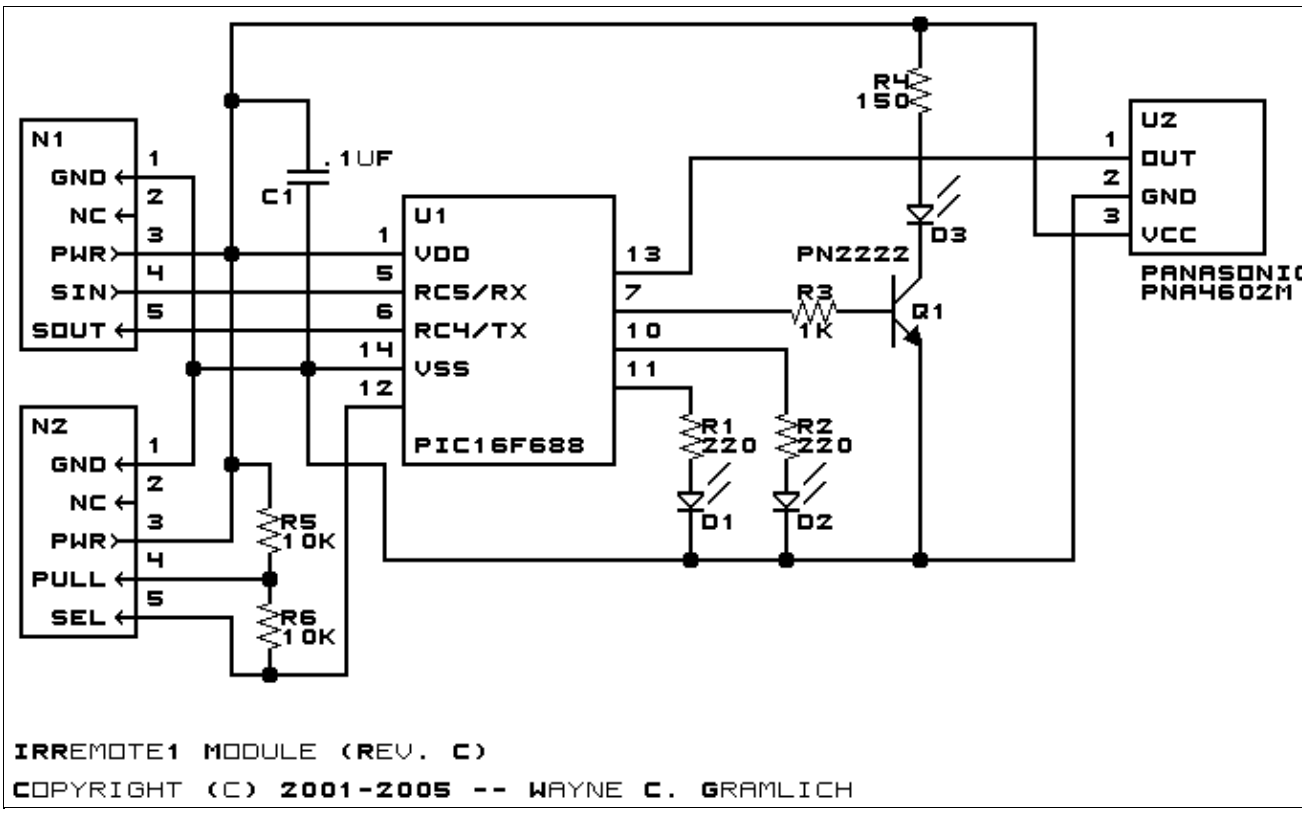
Command	Send/ Receive	Byte Value								Discussion
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
Read Inputs	Send	0	0	0	0	0	0	0	0	Return input values *abcdefghijk*
	Receive	0	0	0	0	*a*	*b*	*c*	*d*	
	Receive	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	
Read Interrupt Bits	Send	1	1	1	0	1	1	1	1	Return the interrupt pending bit *p* and the interrupt enable bit *e*.
	Receive	0	0	0	0	0	0	*e*	*p*	
Set Interrupt Bit Commands	Send	1	1	1	1	0	*c*	*c*	*c*	Execute shared set interrupt command *ccc*.
Shared Commands	Send	1	1	1	1	1	*c*	*c*	*c*	Execute shared command *ccc*.

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The IRRemote1 RoboBrick schematic is shown below:



The parts list kept in a separate file −− irremote1.ptl.

3.2 Printed Circuit Board

The available printed circuit boards are listed below:

irremote1_back.png
: The solder side layer.

irremote1_front.png
: The component side layer.

irremote1_artwork.png
: The artwork layer.

irremote1.gbl
: The RS−274X "Gerber" back (solder side) layer.

irremote1.gtl
: The RS−274X "Gerber" top (component side) layer.

irremote1.gal
: The RS−274X "Gerber" artwork layer.

irremote1.drl
: The "Excellon" NC drill file.

irremote1.tol
: The "Excellon" NC drill rack file.

4. Software

The IRREMOTE1 software is available as one of:

irremote1.ucl
: The µCL source file.

irremote1.asm
: The resulting human readable PIC assembly file.

irremote1.lst
: The resulting human readable PIC listing file.

irremote1.hex
: The resulting Intel® Hex file that can be fed into a PIC12C5xx programmer.

5. Issues

Any fabrication issues are listed here.

This is the Revision A verion of the IO8 Module. The status of this project is finished.

IO8 Module (Revision A)

Table of Contents

This document is also available in PDF format.

1. Introduction

The IO8 Module is used to connect up to 5 Sharp® GP2D12 IROD (InfraRed Optical Distance) measuring sensors. The GP2D12 module provides an analog voltage that is proportional to the distance (although not linearly.)

2. Programming

The IO8 Module provides up to 8 lines of analog input, digital input, and/or digital output under user control. For analog input, 10−bits of analog to digital conversion are permitted.

Module Interrupt Protocol for those lines that are being used as inputs. The interrupt pending bit is set whenever the the formula:

L&(~I) | H&I | R&(~P)&I | F&P&(~I)

is non−zero, where:

- I is the current input bits XOR'ed with the complement mask (C)
- P is the previous value of I
- L is the low mask
- H is the high mask
- R is the raising mask
- F is the falling mask

and

- ~ is bit−wise complement
- | is bit−wise OR
- & is bit−wise AND

Once the interrupt pending bit is set, it must be explicitly cleared by the user.

In addition to the common shared commands and the shared interrupt commands, the AnalogIn4 Module

supports following commands:

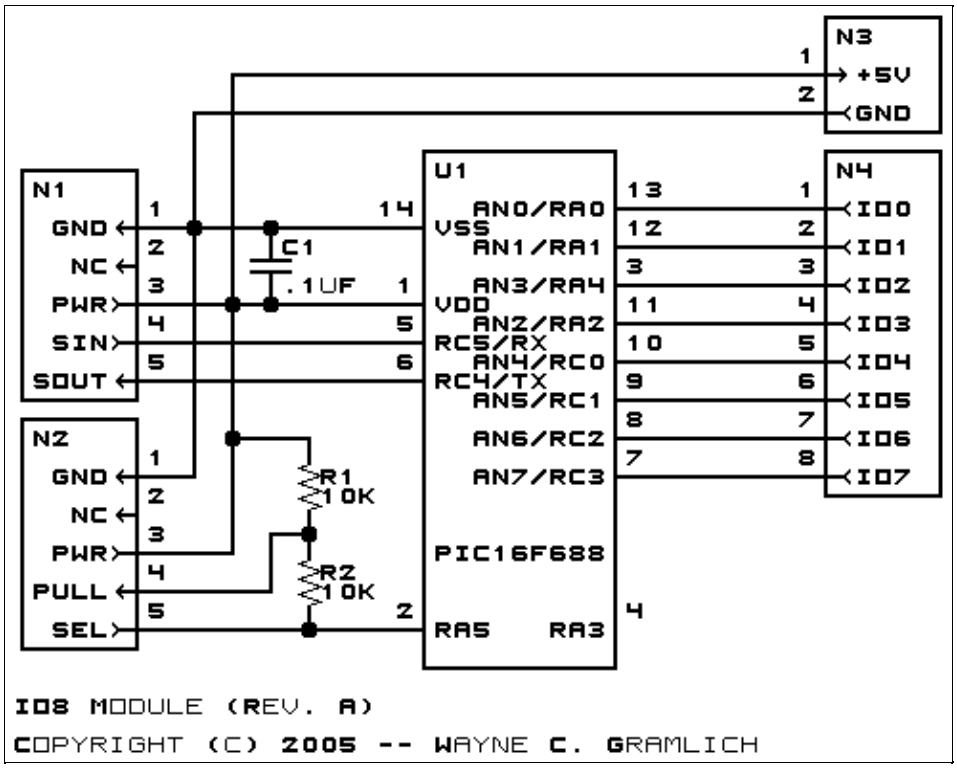
Command	Send/ Receive	Byte Value								Discussion
		7	6	5	4	3	2	1	0	
Read Distance	Send	0	0	0	0	0	0	0	*b*	Read IROD *b* and respond with 8−bit value *dddddddd*
	Send	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	
Read Binary Values	Send	0	0	0	0	0	0	1	0	Return the binary values *ab* (after XOR'ing with complement mask)
	Receive	0	0	0	0	0	0	*a*	*b*	
Read Raw Binary	Send	0	0	0	0	0	0	1	1	Return the raw binary values *ab* (no XOR with complement mask)
	Receive	0	0	0	0	0	0	*a*	*b*	
Reset	Send	0	0	0	0	0	1	0	0	Reset everything to zero
Read Enable Bit	Send	0	0	0	0	0	1	0	1	Read and return the enable bit *e*
	Receive	0	0	0	0	0	0	0	*e*	
Set Enable Bit	Send	0	0	0	0	0	1	1	*e*	Set enable bit to *e*
Read Complement Mask	Send	0	0	0	0	1	0	0	0	Return and return the complement mask *cccc*
	Receive	0	0	0	0	0	0	*c*	*c*	
Read High Mask	Send	0	0	0	0	1	0	0	1	Return and return the high mask *hh*
	Receive	0	0	0	0	0	0	*h*	*h*	
Read Low Mask	Send	0	0	0	0	1	0	1	0	Return and return the low mask *ll*
	Receive	0	0	0	0	0	0	*l*	*l*	
Read Raising Mask	Send	0	0	0	0	1	0	1	1	Return and return the raising mask *rr*
	Receive	0	0	0	0	0	0	*r*	*r*	
Read Falling Mask	Send	0	0	0	0	1	1	0	0	Return and return the falling mask *ff*
	Receive	0	0	0	0	0	0	*f*	*f*	
Read High Threshold	Send	0	0	0	1	0	0	0	*b*	Read and return high threshold for pin *b* of *hhhhhhhh*
	Receive	*h*	*h*	*h*	*h*	*h*	*h*	*h*	*h*	
Read Low Threshold	Send	0	0	0	1	0	0	1	*b*	Read and return low threshold for pin *bb* of *lllllll*
	Receive	*l*	*l*	*l*	*l*	*l*	*l*	*l*	*l*	
Set High Threshold	Send	0	0	0	1	0	1	0	*b*	Set high threshold for pin *b* to *hhhhhhhh*
	Send	*h*	*h*	*h*	*h*	*h*	*h*	*h*	*h*	
Set Low Threshold	Send	0	0	0	1	0	1	1	*b*	Set low threshold for pin *b* to *lllllll*
	Send	*l*	*l*	*l*	*l*	*l*	*l*	*l*	*l*	
Set Complement Mask	Send	0	0	1	0	0	0	*c*	*c*	Set complement mask to *cc*
Set High Mask	Send	0	0	1	0	0	1	*h*	*h*	Set high mask to *hh*
Set Low Mask	Send	0	0	1	0	1	0	*l*	*l*	Set low mask to *ll*
Set Raising Mask	Send	0	0	1	0	1	1	*r*	*r*	Set raising mask to *rr*
Set Falling Mask	Send	0	0	1	1	0	0	*f*	*f*	Set falling mask to *ff*
Read Interrupt Bits	Send	1	1	1	0	1	1	1	1	Return the interrupt pending bit *p* and the interrupt enable bit *e*.
	Receive	0	0	0	0	0	0	*e*	*p*	
Set Interrupt Commands	Send	1	1	1	1	0	*c*	*c*	*c*	Set Interrupt Command *ccc*.
Shared Commands	Send	1	1	1	1	1	*c*	*c*	*c*	Execute common shared command *ccc*

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the IO8 Module is shown below:



The parts list kept in a separate file −− io8.ptl.

3.2 Printed Circuit Board

The printed circuit board files are listed below:

io8_back.png
: The solder side layer.

io8_front.png
: The component side layer.

io8_artwork.png
: The artwork layer.

io8.gbl
: The RS−272X "Gerber" back (solder side) layer.

io8.gtl
: The RS−272X "Gerber" top (component side) layer.

io8.gal
: The RS−272X "Gerber" artwork layer.

io8.drl

The "Excellon" NC drill file.

io8.tol

The "Excellon" tool rack file.

4. Software

The software for the IO8 is listed below:

io8.ucl

The µCL file for IO8.

io8.asm

The assembly file for IO8.

io8.hex

The Intel® Hex file.

io8.lst

The listing file for IO8.

5. Issues

Any fabrication issues will be listed here.

This is the Revision A version of the LaserHolder1 Module. The status of this project is work in progress.

LaserHolder1 Module (Revision A)

Table of Contents

This document is also available as a PDF document.

1. Introduction

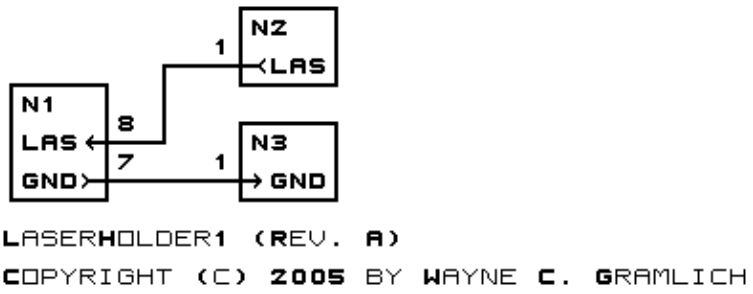
The LaserHolder1 module is used to mechanically support a small laser pointer for the Sense3 module. Two of these modules are needed to accomplish the task. Both modules have the center hole enlarged just enough to accept the desired small laser pointer. The four outside corner holes of the LaserHolder1 align with the four holes on the Sense3 module. Some all thread, nuts, washers and lock washers are used to stack two LaserHolder1 modules behind the Sense3 module to hold the laser pointer. Finally, the LaserHolder1 is used to electically connect the Sense3 through the ScanPanel module, which, in turn, is mounted on top of a small hobby servo.

2. Hardware

...

2.1 Schematic

The schematic is shown below:



The parts list is kept in a separate file.

2.2 Printed Circuit_Board

The printed circuit files are listed below:

laserholder1_back.png
: The solder side layer.

laserholder1_front.png
: The component side layer.

laserholder1_artwork.png
: The artwork layer.

laserholder1.gbl
: The RS−274X "Gerber" back (solder side) layer.

laserholder1.gtl
: The RS−274X "Gerber" top (component side) layer.

laserholder1.gal
: The RS−274X "Gerber" artwork layer.

laserholder1.drl
: The "Excellon" NC drill file.

laserholder1.tol
: The "Excellon" tool rack file.

3. Issues

Any fabrication issues will be listed here.

This is the Revision A version of the LCD32 RoboBrick. The status of this project is work in progress.

LCD32 Module (Revision E)

Table of Contents

This document is also available in PDF format.

1. Introduction

The LCD32 module can display a total 4 lines of 16 characters each, of which only 2 lines are visible at a time. The characters are displayed using a 5×7 dot matrix. There is a mechancal switch labeled LINES on the LCD32 module that switches between displaying lines 1−2 and lines 3−4. The LCD32 module is based upon the inexpensive Lumex® LCM−S01602DTR/M 2×16 liquid crystal display (LCD) module available from both Digikey® and Mouser The LCD32 module has a small trim potentiometer that allows you adjust the display contrast.

The LCD32 can be used in two ways:

User Mode
: In user mode, the LCD32 is being used an output device under user control. All serial data communication is between the master connected on N1 and the LCD32 module. In this mode, connectors N2 and N3 are left disconnected. A pull−up on one of the N2 connector pins allows the LCD32 module to determine that it is in this mode.

Debug Mode
: In debug mode, the LCD32 module is inserted between a master brick and a slave brick. The master brick is connected on N1 and the slave brick is connected on N3. In addition, there is another connection from a "debug" port on the slave brick and connector N2 on the LCD32. In this mode, the master brick does not even know that the LCD32 is present. However, the slave brick can detect that LCD32 is present and output additional debugging information to the LCD32. There are two sub−modes of operation in this mode −− 1) slave sending data to the master and 2) slave sending data to the LCD32. The LCD32 module determines this by examining the "select" line on connector N2.

The various paths for serial data are summarized in the table below:

Description	Mode (S1)	Select (S0)	Master Out (N1−5)	Slave Out (N3−4)	LCD32 RX (U1−5)
Master to LCD32 (User Mode)	1	x	LCD32 TX (U1−6)	x (Master In) (N1−4)	Master In (N3−4)
Slave to LCD32 (Debug Mode)	0	1	idle (high)	Master In (N1−4)	Slave In (N3−5)
Slave to Master (Debug Mode)	0	0	Slave In (N3−5)	Master In (N1−4)	idle (high)

The LCD32 module is meant to be used in conjunction with the LCD32Holder (Rev. A) board which carries the actual LCM−201602DTR/M and plugs onto the top of the LCD32 module.

2. Programming

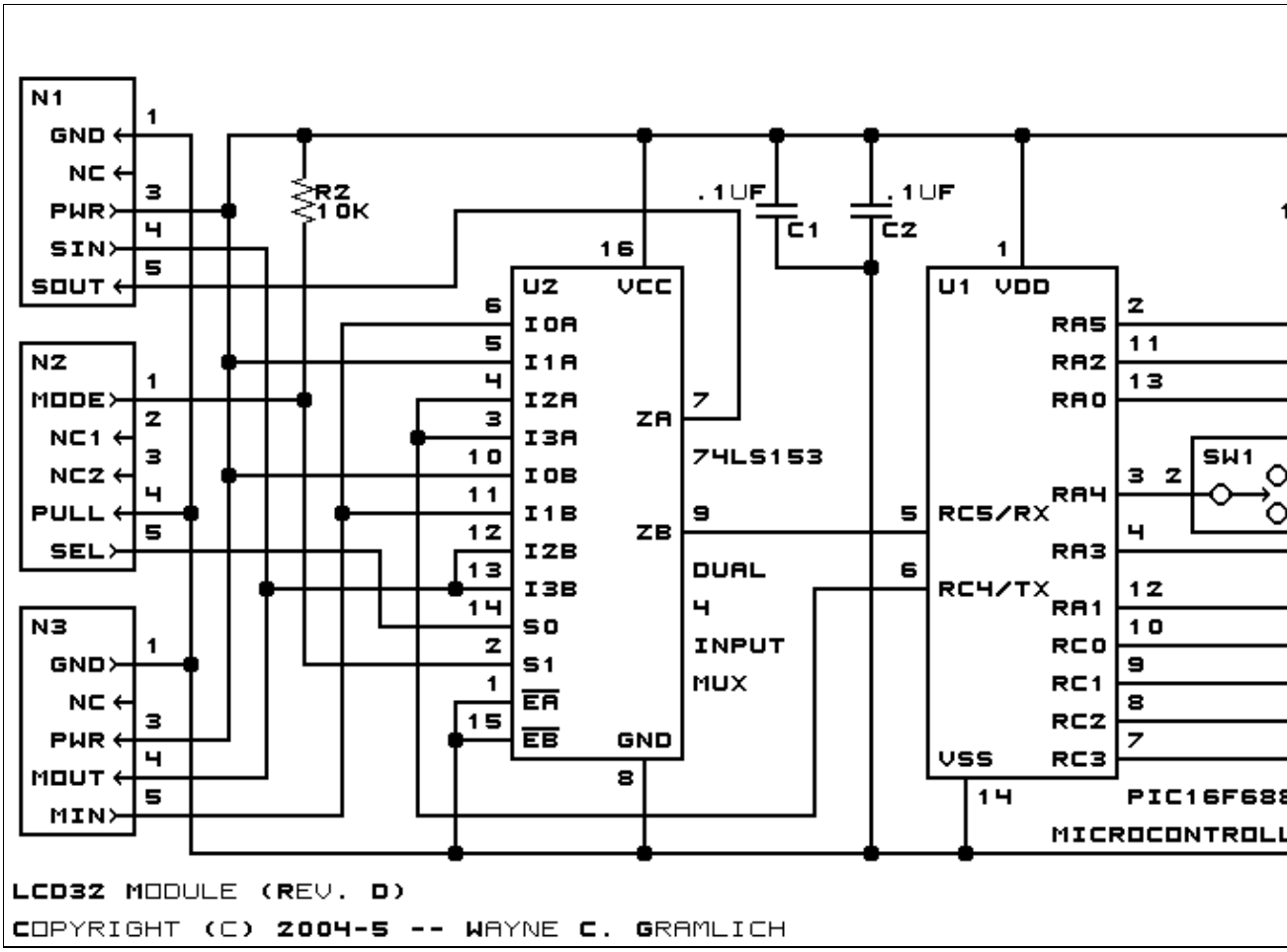
Command	Send/ Receive	Byte Value								Discussion
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
Back Space	Send	0	0	0	0	1	0	0	0	Move cursor to the left.
Line Feed	Send	0	0	0	0	1	0	1	0	Advance currsor to beginning of next line; clear the next line
Form Feed	Send	0	0	0	0	1	1	0	0	Clear entire display and place cursor at home
Carriage Return	Send	0	0	0	0	1	1	0	1	Return cursor to beginning of line
Character 32 to 63	Send	0	0	1	x	x	x	x	x	Enter the character on the display and advance cursor.
Character 64 to 127	Send	0	1	x	x	x	x	x	x	Enter the character on the display and advance cursor.
Line Set	Send	1	0	0	0	0	0	*l*	*l*	Move cursor to line *ll*
Line Clear	Send	1	0	0	0	0	1	*l*	*l*	Move cursor to line *ll* and clear it
Cursor Mode Set	Send	1	0	0	0	1	0	*v*	*b*	Cursor mode is set (*v*=1 visible cursor) (*b*=1 blinking cursor)
Cursor Mode Read	Send	1	0	0	0	1	1	0	0	Read cursor mode (*v*=1 visible cursor) (*b*=1 blinking cursor)
	Receive	0	0	0	0	0	0	*v*	*b*	
Character Read	Send	1	0	0	0	1	1	0	1	Read the current character *ccc cccc*; advance cursor
	Receive	0	*c*	*c*	*c*	*c*	*c*	*c*	*c*	
Line Read	Send	1	0	0	0	1	1	1	0	Read the current line *ll*
	Receive	0	0	0	0	0	0	*l*	*l*	
Position Read	Send	1	0	0	0	1	1	1	1	Read the current character position *pppp*
	Receive	0	0	0	0	*p*	*p*	*p*	*p*	
Position Set	Send	1	0	0	1	*p*	*p*	*p*	*p*	Move cursor to character position *pppp*
Position Set	Send	1	0	1	0	*p*	*p*	*p*	*p*	Move cursor to character position *pppp*; clear to end of line
Shared Commands	Send	1	1	1	1	1	*c*	*c*	*c*	Execute shared command *ccc*.

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the LCD32 RoboBrick is shown below:



The parts list kept in a separate file −− lcd32.ptl.

3.2 Printed Circuit Board

The printed circuit board files are listed below:

lcd32_back.png
: The solder side layer.

lcd32_front.png
: The component side layer.

lcd32_artwork.png
: The artwork layer.

lcd32.gbl
: The RS−272X "Gerber" back (solder side) layer.

lcd32.gtl
: The RS−272X "Gerber" top (component side) layer.

lcd32.gal
: The RS−272X "Gerber" artwork layer.

lcd32.drl
: The "Excellon" NC drill file.

lcd32.tol
: The "Excellon" tool rack file.

4. Software

The LCD32 software is available as one of:

lcd32.ucl
: The µCL source file.

lcd32.asm
: The resulting human readable PIC assembly file.

lcd32.lst
: The resulting human readable PIC listing file.

lcd32.hex
: The resulting Intel® Hex file.

5. Issues

The following issues came up:

- The connector N2 in not properly aligned with connectors N3 and N4 on LCD32Holder−C. Move N2 right .05" to get them aligned. This will also provide a little extra space on the left side of the board.

This is the Revision C version of the LCD32Holder module. The status of this project is work in progress.

LCD32Holder Module (Revision C)

Table of Contents

This document is also available in PDF format.

1. Introduction

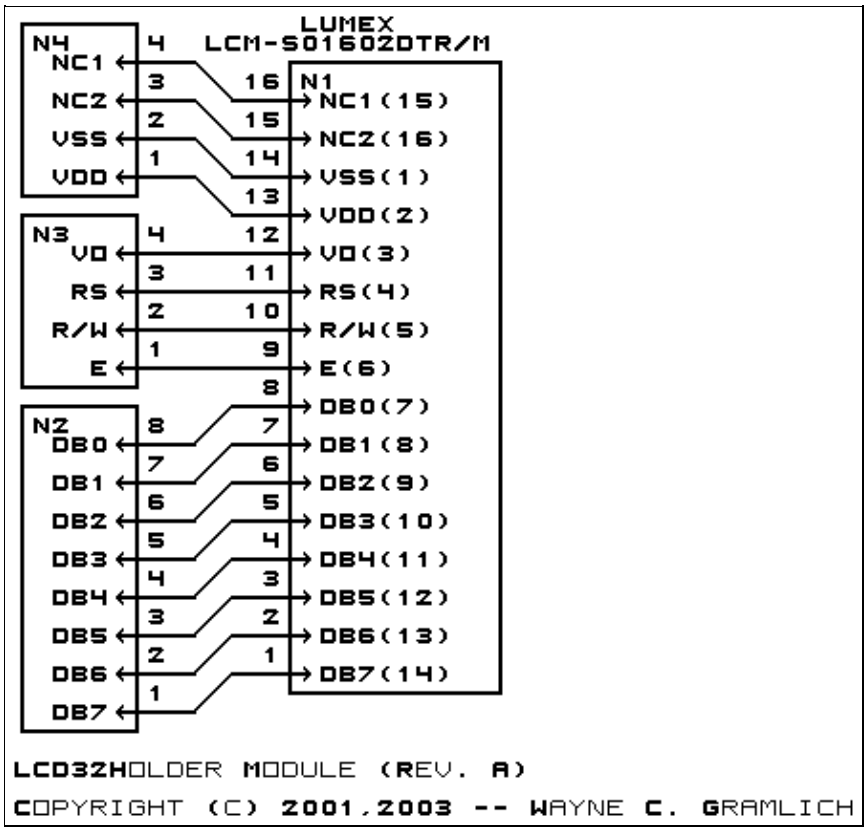
The LCD32HOLDERHolder module adapts the Lumex LCM−S01602DTR/M to the LCD32HOLDER module.

2. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

2.1 Circuit Schematic

The schematic for the LCD32HOLDERHolder module is shown below:



The parts list kept in a separate file −− lcd32holder.ptl.

2.2 Printed Circuit Board

The printed circuit board files are listed below:

lcd32holder_back.png
: The solder side layer.

lcd32holder_front.png
: The component side layer.

lcd32holder_artwork.png
: The artwork layer.

lcd32holder.gbl
: The RS−272X "Gerber" back (solder side) layer.

lcd32holder.gtl
: The RS−272X "Gerber" top (component side) layer.

lcd32holder.gal
: The RS−272X "Gerber" artwork layer.

lcd32holder.drl
: The "Excellon" NC drill file.

lcd32holder.tol
: The "Excellon" tool rack file.

3. Issues

The following fabrication issues came up:

- The stand−off next to N3 just happens to land on top of a surface mount resistor on the LCD board. Move it up by .15".

This is the Revision F verion of the LED10 module. The status of this project is work in progress.

Led10 Module (Revision F)

Table of Contents

This document is also available in PDF format.

1. Introduction

The LED10 module provides the ability to output 10 bits of data to 10 LED's on board.

2. Programming

The Led4 Module supports the standard shared commands in addition to the following commands:

Command	Send/Receive	Byte Value								Discussion
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
Write Lower	Send	0	0	0	*f*	*g*	*h*	*i*	*j*	Write *fghij* out to the lower 5 LED's.
Write Upper	Send	0	0	1	*a*	*b*	*c*	*d*	*e*	Write *abcde* out to the upper 5 LED's.
Bit Clear	Send	0	1	0	0	*b*	*b*	*b*	*b*	Turn LED *bbbb* off. MSB (*bbbb*=1001) LSB (*bbbb*=0000)
Bit Set	Send	0	1	0	1	*b*	*b*	*b*	*b*	Turn LED *bbbb* on.
Bit Toggle	Send	0	1	1	0	*b*	*b*	*b*	*b*	Toggle LED *bbbb*.
Bit Read	Send	0	1	1	1	*b*	*b*	*b*	*b*	Read status of LED *bb*.
	Receive	*r*	*r*	*r*	0	0	0	0	*b*	LED state is *b*. Blink rate is *rrr*
Read All	Send	1	0	0	0	0	0	0	0	Read all ten LED's.
	Receive	0	0	0	*a*	*b*	*c*	*d*	*e*	Upper five LED state is *abcde*
	Receive	0	0	0	*f*	*g*	*h*	*i*	*j*	Lower five LED state is *fghij*
Read Lower	Send	1	0	0	0	0	0	0	1	Read lower five LED's.
	Receive	0	0	0	*f*	*g*	*h*	*i*	*j*	Lower five LED state is *fghij*
Read Upper	Send	1	0	0	0	0	0	1	0	Read upper five LED's.
	Receive	0	0	0	*a*	*b*	*c*	*d*	*e*	Upper five LED state is *abcde*
Blink Rate Set	Send	1	0	0	0	0	0	1	1	Set Blink Rate
	Send	r	r	r	0	b	b	b	b	Set LED *bbbb* blink rate to *rrr*. On (*rrr*=000) Slow (*rrr*=001) Medium(*rrr*=100) Fast (*rrr*=111)
Increment LED's	Send	1	0	0	1	*b*	*b*	*b*	*b*	Increment LED's starting at bit *bbbb*

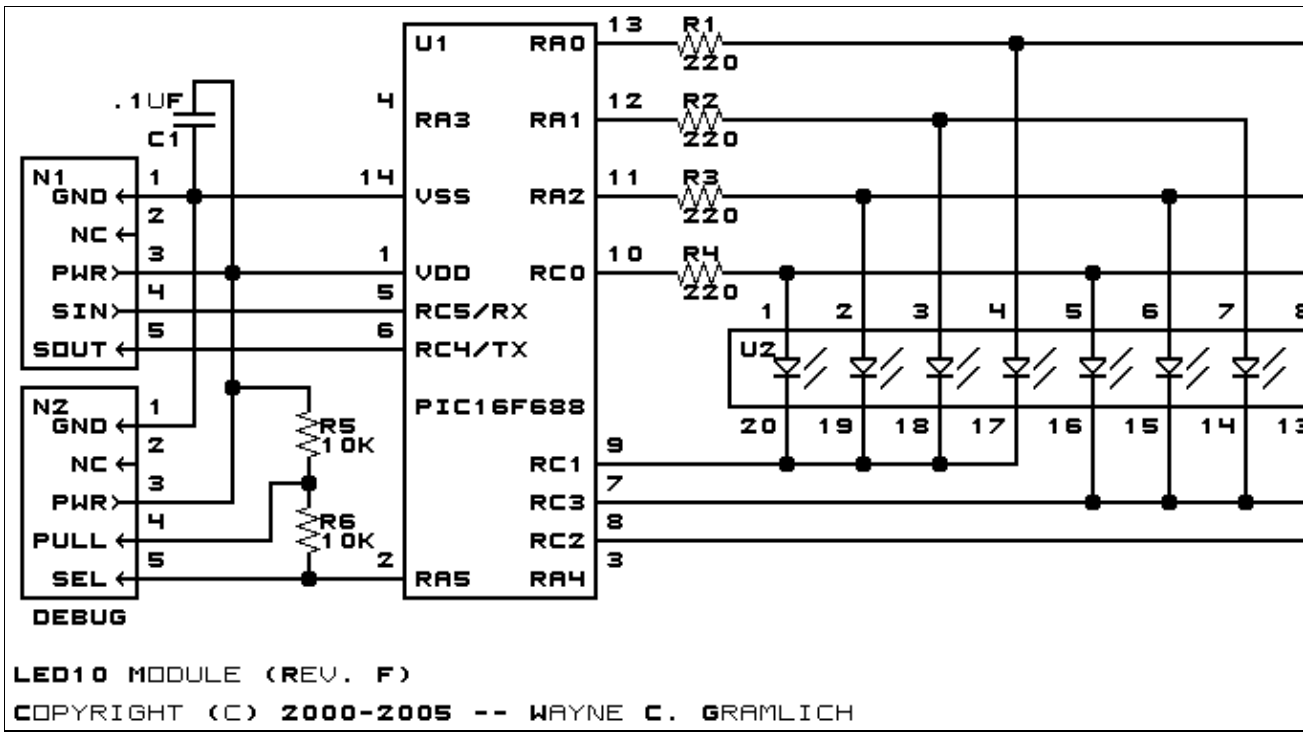
Decrement LED's	Send	1	0	1	0	*b*	*b*	*b*	*b*	Decrement LED's starting at bit *bbbb*
Power Level Mode	Send	1	0	1	1	*l*	*l*	*l*	*l*	Set LED's to power level *llll*; All off (*llll*=000), All on (*llll*>=1010)
Shared Commands	Send	1	1	1	1	1	*a*	*b*	*c*	Send shared command *abc* to Module.

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the Led10 Module is shown below:



The parts list kept in a separate file −− led10.ptl.

3.2 Printed Circuit Board

The printed circuit board files are listed below:

led10_back.png
: The solder side layer.

led10_front.png
: The component side layer.

led10_artwork.png
: The artwork layer.

led10.gbl
: The RS−274X "Gerber" back (solder side) layer.

led10.gtl
: The RS−274X "Gerber" top (component side) layer.

led10.gal
: The RS−274X "Gerber" artwork layer.

led10.drl
: The "Excellon" NC drill file.

led10.tol
: The "Excellon" tool rack file.

4. Software

The Led10 software is available as one of:

led10.ucl
: The µCL source file.

led10.asm
: The resulting human readable PIC assembly file.

led10.lst
: The resulting human readable PIC listing file.

led10.hex
: The resulting Intel® Hex file.

5. Issues

Any fabrication issues are listed here.

This is the Revision A verion of the Line3 module. The status of this project is finished.

Line3 Module (Revision A)

Table of Contents

This document is also available in PDF format.

1. Introduction

The Line3 module has 3 infrared (IR) sensors for sensing lines on flat surfaces. There are sensor that straddle the line and one that can detect when the line simply ends.

2. Programming

{more goes here.}

In addition to the common shared commands and the shared interrupt commands, the Line3 RoboBrix supports following commands:

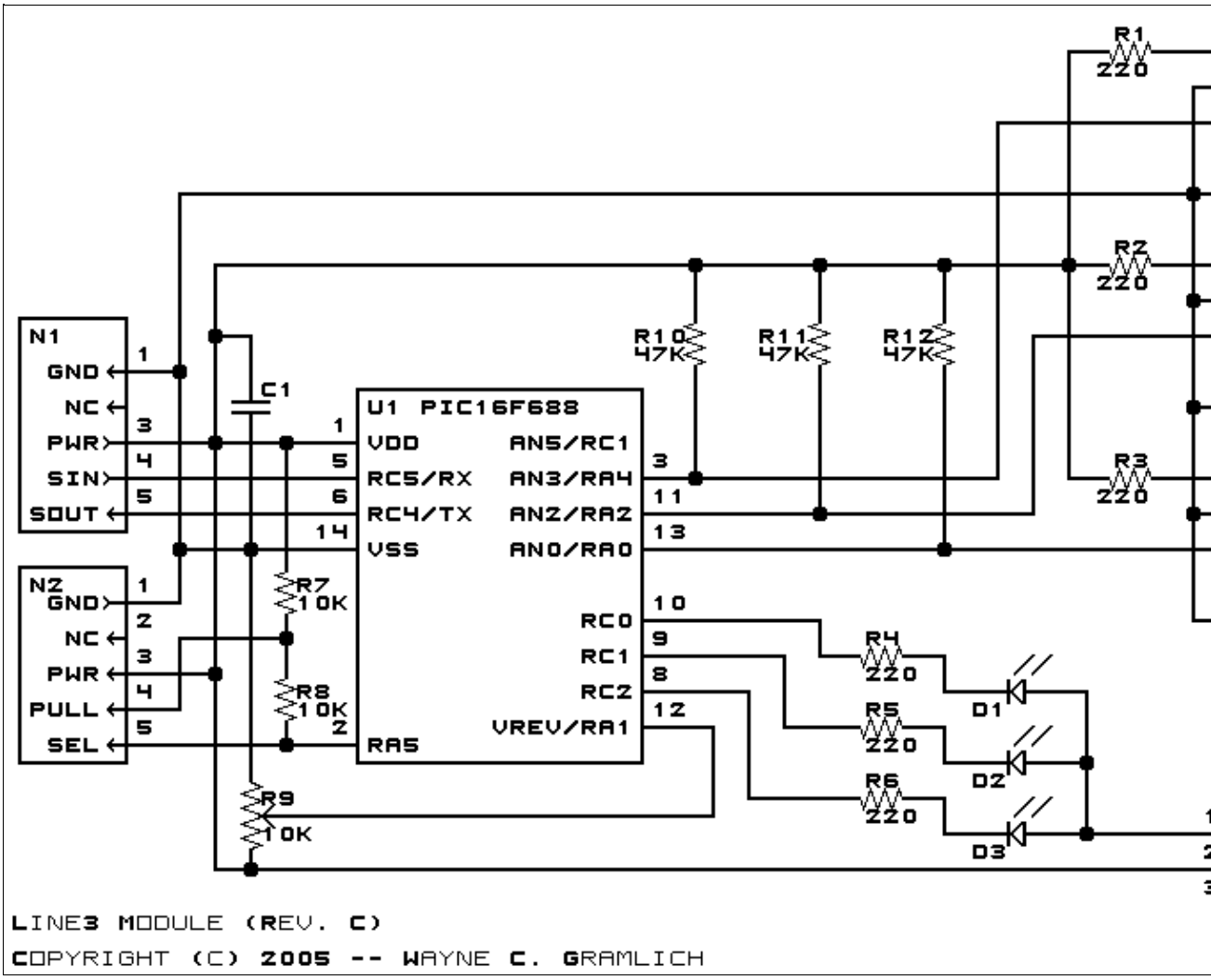
Command	Send/ Receive	Byte Value								Discussion
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
Read Interrupt Bits	Send	1	1	1	0	1	1	1	1	Return the interrupt pending bit *p* and the interrupt enable bit *e*.
	Receive	0	0	0	0	0	0	*e*	*p*	
Set Interrupt Commands	Send	1	1	1	1	0	*c*	*c*	*c*	Set Interrupt Command *ccc*.
Shared Commands	Send	1	1	1	1	1	*c*	*c*	*c*	Execute common shared command *ccc*

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the Line3 RoboBrix is shown below:



The parts list kept in a separate file −− line3.ptl.

3.2 Printed Circuit Board

The printed circuit board files are listed below:

line3_back.png
: The solder side layer.

line3_front.png
: The component side layer.

line3_artwork.png
: The artwork layer.

line3.gbl
: The RS−272X "Gerber" back (solder side) layer.

line3.gtl
: The RS−272X "Gerber" top (component side) layer.

line3.gal
: The RS−272X "Gerber" artwork layer.

line3.drl
: The "Excellon" NC drill file.

line3.tol
: The "Excellon" tool rack file.

4. Software

The Line3 software is available as one of:

line3.ucl
: The µCL source file.

line3.asm
: The resulting human readable PIC assembly file.

line3.lst
: The resulting human readable PIC listing file.

line3.hex
: The resulting Intel® Hex file.

5. Issues

Any fabrication iusses will be listed here.

This is the revision B version of the MicroBrain8 module. The status of this project is finished.

MicroBrain8 Module (Revision C)

Table of Contents

This document is also available as a PDF document.

1. Introduction

The MicroBrain8 module is a controller module that can control up to 8 sensor or actuator modules. It is controlled by any microcontroller that is pin compatible with the Parallax a Basic Stamp 2® It has two terminals that can be connect to a battery between 6 and 9 volts. It has an on board 5 volt voltage regulator to provide power to the other sensor and actuator modules. The is a connector that can be connected to a DB9 connector and used to communicate with a controlling PC via RS−232 voltage levels.

2. Programming

The MicroBrain11 works with any processor that is pin compatible with the Basic Stamp 2 from Parallax. This includes several products from Parallax (e.g. Basic Stamp 2, Basic Stamp2SX, Javalin) and some products from other companies such as (e.g. OOPIC−C from Savage Innovations, etc.)

The required pin−outs from the chip are:

Pin	Label	Description	Pin	Label	Description
1	SOUT	Serial Out	24	VIN	Unregulated > 6 Volts
2	SIN	Serial In	23	VSS	Ground (0 Volts)
3	ATN	Attention	22	RES#	Reset
4	VSS	Ground (0 Volts)	21	VDD	Regulated +5 Volts
5	P0	Data Pin 0	20	P15	Data Pin 15
6	P1	Data Pin 1	19	P14	Data Pin 14
7	P2	Data Pin 2	18	P13	Data Pin 13
8	P3	Data Pin 3	17	P12	Data Pin 12
9	P4	Data Pin 4	16	P11	Data Pin 11
10	P5	Data Pin 5	15	P10	Data Pin 10
11	P6	Data Pin 6	14	P9	Data Pin 9
12	P7	Data Pin 7	13	P8	Data Pin 8

All 16 pins from the microcontroller processor are brought out to the connectors N1−N8. You are free to do whatever you want with these pins; after all, you own the board. You are not required to exclusively talk to RoboBRiX modules.

Having said that, we expect that most people will want to use the MicroBrain8 to operate other RoboBRiX modules. The pins from the processor chip are routed to connectors N1 through N8 in pairs as shown in the table below:

Pin Out	Pin In	Connector (Pin 4)	Connector (Pin 5)
P0	P1	N8	N8
P2	P3	N7	N7
P4	P5	N6	N6
P6	P7	N5	N5
P8	P9	N1	N1
P10	P11	N2	N2
P12	P13	N3	N3
P14	P15	N4	N4

Notice that they are always organized in pairs of the form Pn=Output and Pn+1=Input. The same information, but presented from the point of view of the connector is listed below:

Connector	Input	Output
N1 (Top)	P8	P9
N2	P10	P11
N3	P12	P13
N4	P14	P15
N5	P6	P7
N6	P4	P5
N7	P2	P3
N8 (Bottom)	P0	P1

We may eventually put a few examples of programming the MicroBrain8 here. Basically, it is programmed using the Parallax Basic for the Basic Stamp 2.

```
' Even numbered pins inputs and odd number pins are outputs.
' (Remember for the BS2, 1=output and 0=input.)
dirs = $aaaa

' Set all outputs to high:
high 1
high 3
high 5
high 7
high 9
high 11
high 13
high 15

' To copy a Switch8−B (on N2) to LED10−B (on N1):
```

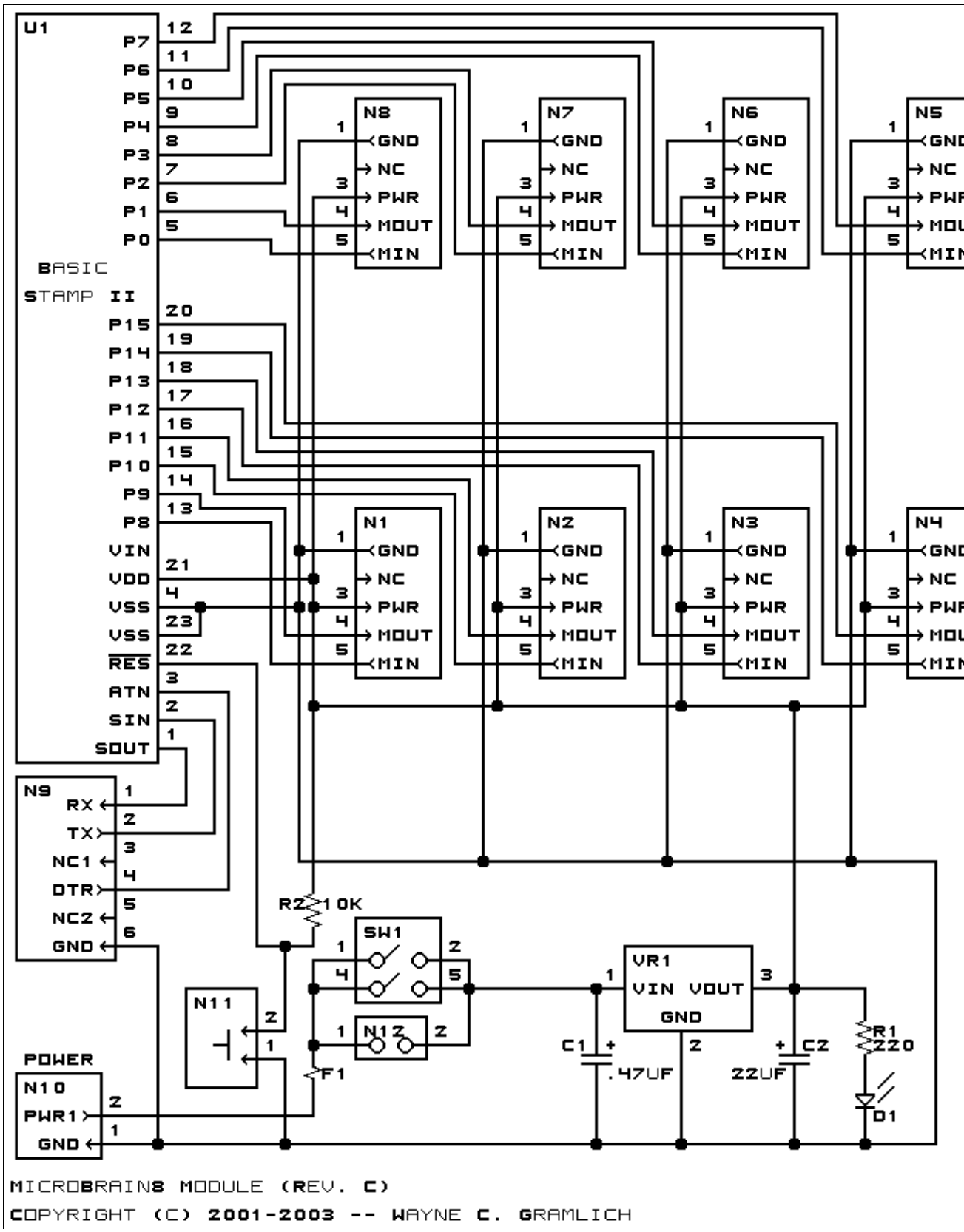
```
switches var byte
loop:
    ' Send command 0 (Read switches) to Switch8-B:
    serout 11, 396, [0]
    ' Receive the switch readings from Switch8-B:
    serin 10, 396, [switches]
    ' Send switch values to LED10-B:
    serout 9, 396, [switches]
    goto loop
```

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the MicroBrain8 RoboBrick is shown below:



The parts list kept in a separate file −− microbrain8.ptl.

3.2 Printed Circuit Board

The printed circuit board files are listed below:

microbrain8_back.png
: The solder side layer.

microbrain8_front.png
: The component side layer.

microbrain8_artwork.png
: The artwork layer.

microbrain8.gbl
: The RS−274X "Gerber" back (solder side) layer.

microbrain8.gtl
: The RS−274X "Gerber" top (component side) layer.

microbrain8.gal
: The RS−274X "Gerber" artwork layer.

microbrain8.gml
: The RS−274X "Gerber" mask layer.

microbrain8.drl
: The "Excellon" NC drill file.

microbrain8.tol
: The "Excellon" tool rack file.

3.2 Construction Instructions

The construction instructions are kept in a separate file document to be a little more printer friendly.

4. Software

The software for the MicroBrain8 is developed by the user.

5. Issues

Any fabrication issues that come up will be listed here.

This is the Revision A verion of the Multiplex8 Module. The status of this project is that it has been replaced by the PIC876Hub10 Module.

Multiplex8 Module (Revision A)

Table of Contents

This document is also available as a PDF document.

1. Introduction

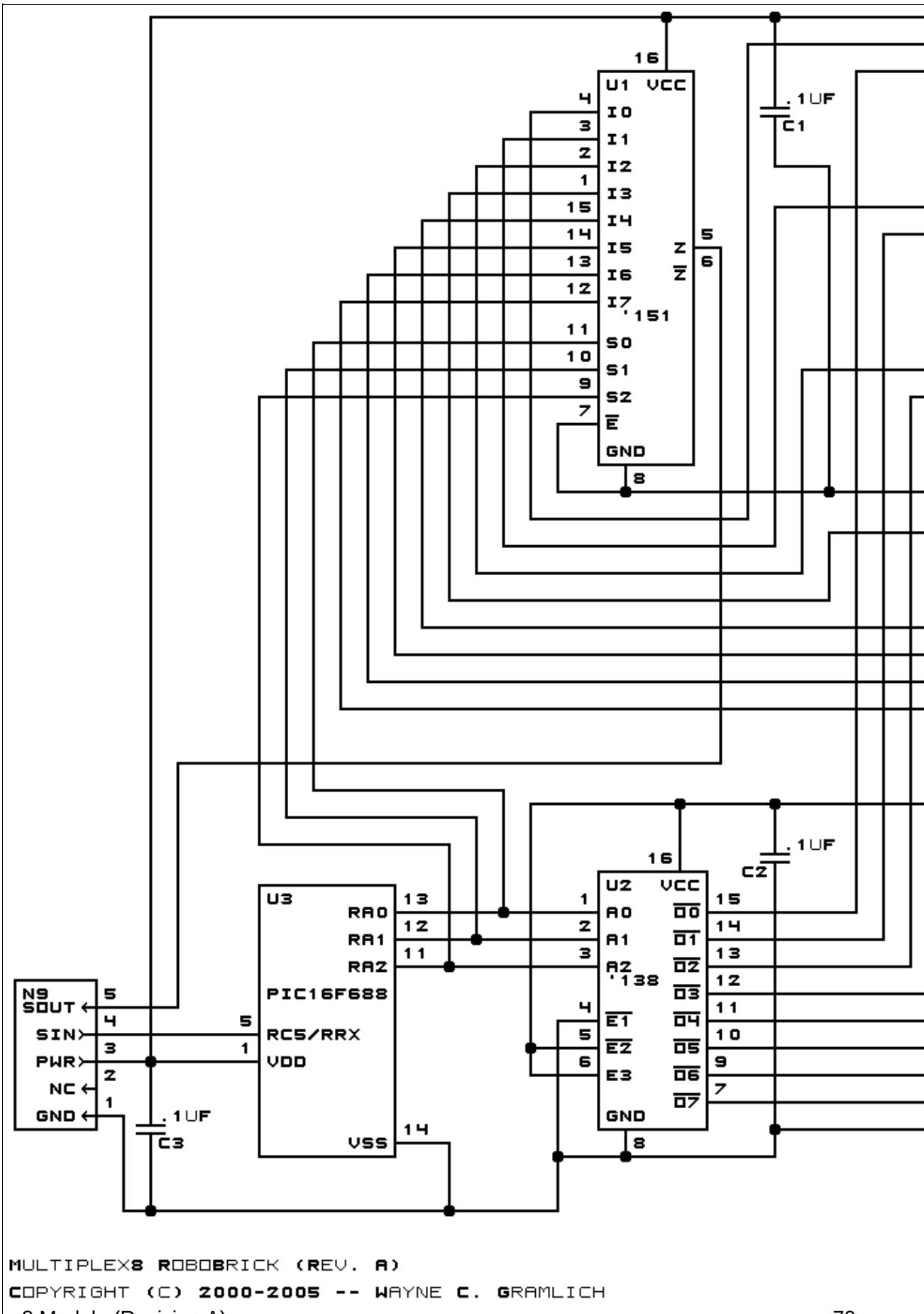
This module is used to multiplex the UART on a master module amongst 8 slave modules.

2. Hardware

The hardware consists of a circuit schematics and a printed circuit board.

2.1 Circuit Schematic

The schematic for the Multiplex8 Module is shown below:


U1 VCC
'151
GND
U2 VCC
'138
GND
U3
PIC16F688
RA0
RA1
RA2
RC5/RRX
VDD
VSS
N9
SOUT
SIN
PWR
NC
GND
.1UF
C1
C2
C3
MULTIPLEX8 ROBOBRICK (REV. A)
COPYRIGHT (C) 2000-2005 -- WAYNE C. GRAMLICH

The parts list kept in a separate file −− multiplex8.ptl.

2.2 Printed Circuit Board

multiplex8_back.png
: The solder side layer.

multiplex8_front.png
: The component side layer.

multiplex8_artwork.png
: The artwork layer.

multiplex8.gbl
: The RS−274X "Gerber" back (solder side) layer.

multiplex8.gtl
: The RS−274X "Gerber" top (component side) layer.

multiplex8.gal
: The RS−274X "Gerber" artwork layer.

multiplex8.drl
: The "Excellon" NC drill file.

multiplex8.tol
: The "Excellon" NC drill rack file.

3. Issues

Any construction issues are listed here:

This is the revision D version of the PICBrain11 module. The status of this project is finished.

PICBrain11 Module (Revision D)

Table of Contents

This document is also available as a PDF document.

1. Introduction

The PICBrain11 module can control up to 11 sensor and actuator modules. It uses a PIC16F876A microcontroller from MicroChip®. There is on board RS−232 level conversion circuitry so that the PICBrain11 can be directly connected to a PC serial port. There is an on−board 5 volt regulator for supplying regulated voltage to other sensor and actuator modules.

2. Programming

You are free to program the PICBrain11 in any fashion you so desire. There are many compilers and assemblers available for the PIC microcontroller family. A search of Google® for terms such as "PIC Basic Compiler" and "PIC C Compiler" will yield many possible compilers. The MPLab® software from Microchip® provides an excellent assembler that is free.

All 22 I/O pins of the the PICBRain11 are available on connectors N1 through N11. There is no requirement that you use the pins to only talk to the sensor and actuator modules. For example, RC4 and RC3 can be used to access an I^2C bus with the addition of external pull up resistors.

However, we do expect many people to want to connect to sensor/actuator modules. This is done by providing a routine that can bit bang (a technical term) serial protocol in and out on a pair of pins at 2400 baud using an 8N1 (1 start bit, 8 data bits, No parity, 1 stop bit).

The table below shows the mapping of I/O register bits to connector pins:

Pin	Location	Direction
RA0	N2 Pin 5	Input
RA1	N2 Pin 4	Output
RA2	N4 Pin 5	Input
RA3	N4 Pin 4	Output
RA4	N6 Pin 5	Input
RA5	N6 Pin 4	Output
RB0	N7 Pin 5	Input

RB1	N7 Pin 4	Output
RB2	N5 Pin 5	Input
RB3	N5 Pin 4	Output
RB4	N3 Pin 5	Input
RB5	N3 Pin 4	Output
RB6	N1 Pin 5	Input
RB7	N1 Pin 4	Output
RC0	N9 Pin 5	Input
RC1	N9 Pin 4	Output
RC2	N8 Pin 5	Input
RC3	N10 Pin 5	Input
RC4	N10 Pin 4	Output
RC5	N8 Pin 4	Output
RC6	N11Pin 4	Output
RC7	N11 Pin 5	Input

The table below is the inverse table that shows the the connector pins to register bit mapping:

Connector	**Pin 4**	**Pin 5**
N1	RB7	RB6
N2	RA1	RA0
N3	RB5	RB4
N4	RA3	RA2
N5	RB3	RB2
N6	RA5	RA4
N7	RB1	RB0
N8	RC5	RC2
N9	RC1	RC0
N10	RC4	RC3
N11	RC6	RC7

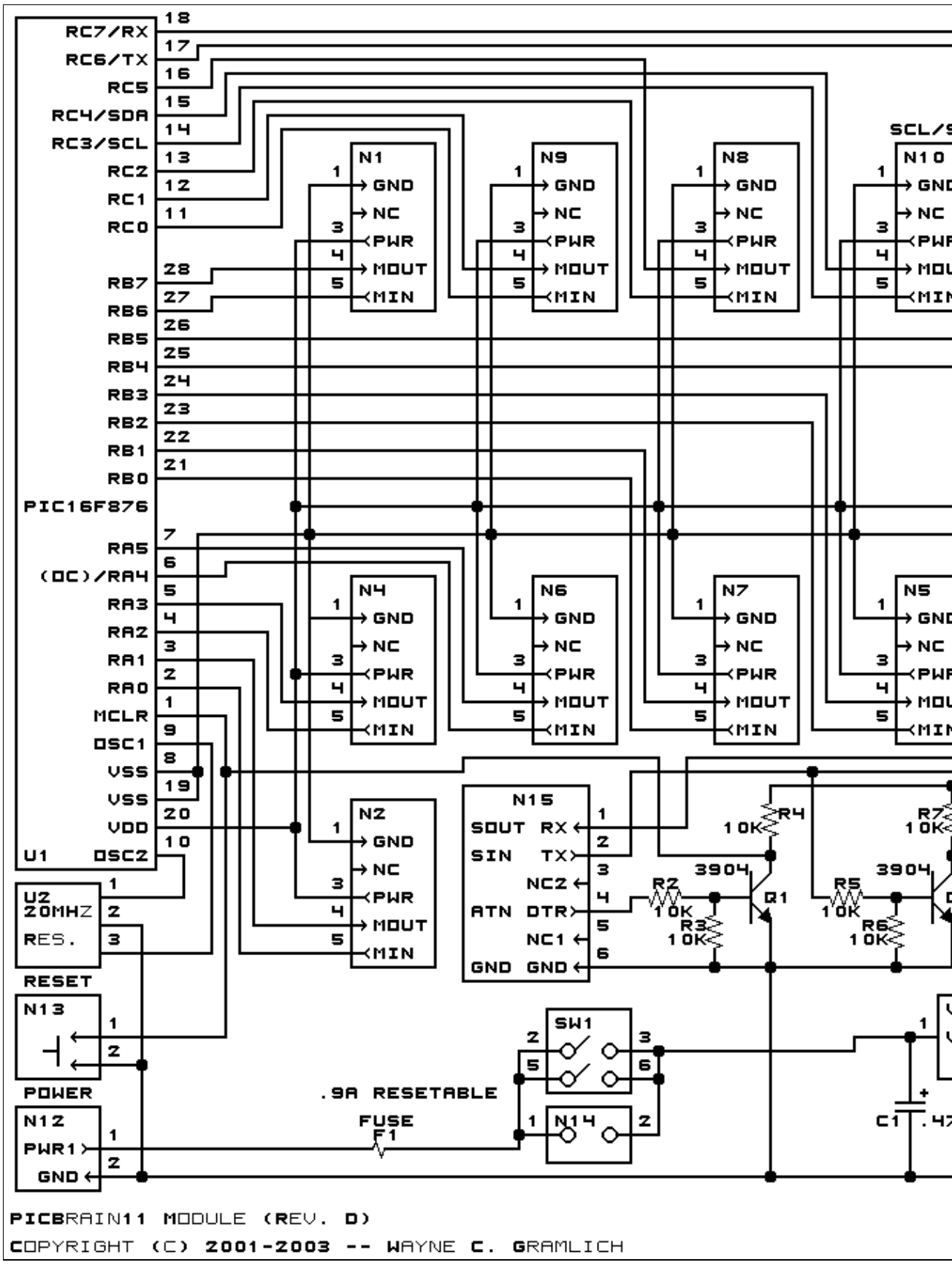
Note that most connectors are of the form where pin 5 is connected to PortN@BitN, where BitN is even and pin 4 is connected to PortN@BitN+1. Connects N8, N10, and N11 are the exceptions to this rule. N11 is connected to the TX and RX pins of the PIC16F876A hardware USART. N8 is connected to the SCA and SCL pins PIC16F876A I^2C support hardware.

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the PICBrain11 module is shown below:



The parts list kept in a separate file -- picbrain11.ptl.

3.2 Printed Circuit Board

The printed circuit board files are listed below:

picbrain11_back.png
: The solder side layer.

picbrain11_front.png
: The component side layer.

picbrain11_artwork.png
: The artwork layer.

picbrain11.gbl
: The RS−274X "Gerber" back (solder side) layer.

picbrain11.gtl
: The RS−274X "Gerber" top (component side) layer.

picbrain11.gal
: The RS−274X "Gerber" artwork layer.

picbrain11.drl
: The "Excellon" NC drill file.

picbrain11.tol
: The "Excellon" tool rack file.

3.3 Download Cable

More and more computers are shipping without serial ports. If your computer does not have a serial port, it will be necessary to purchase a USB to serial adaptor. Some converters appear to be better than others. In general, we have been pretty satisfied with adaptors based on FTDI chips. You may have to experiment until you find one that works for your system.

There are two download cables for the PICBrain11 −− the 4−wire cable and the 3−wire cable. The 4−wire cable is identical to the cable used for the MicroBrain8. The 3−wire cable is the same as the 4−wire cable, with the exception that the DTR signal is not hooked up. Both these cables end in a six pin 1×6 male header where pins 3 and 5 have been removed to provide polarization when plugged into N15. The two different cables are described in the two tables below:

4−wire Download Cable						
DB9		**Direction**	**N15**		**PIC**	
Label	**Pin**		**Label**	**Pin**	**Label**	**Pin**
RX	2	<==	SOUT	1	RC6/TX	17
TX	3	==>	SIN	2	RC7/RX	18
DTR	4	==>	ATN	4	MCLR	1
GND	5	<=>	GND	6	VSS	8,19
RTS	7	Short 7 to 8				
DSR	8	Short 8 to 7				

3−wire Download Cable						
DB9		**Direction**	**N15**		**PIC**	
Label	**Pin**		**Label**	**Pin**	**Label**	**Pin**

RX	2	<==	SOUT	1	RC6/TX	17
TX	3	==>	SIN	2	RC7/RX	18
GND	5	<=>	GND	6	VSS	8,19
RTS	7	Short 7 to 8				
DSR	8	Short 8 to 7				

Why are two different cables needed? The answer starts with the Parallax® Basic Stamp. Parallax decided to use DTR in a very non−standard fashion. They use DTR for reset. The MicroBrain8 ***must*** adhere to this Parallax "standard".

The goal for the PICBrain11 is to use the same identical cable as the Parallax cable. Thus, on the PICBrain11, DTR is also used for reset. Unfortunately, most terminal emulators assert DTR as part of their normal behavior. (DTR means Data Terminal Ready, which is properly asserted by a terminal emulator.) This causes the PICBrain11 to jam in the reset state. The more advanced terminal emulators allow you to independently set the DTR pin; alas, Microsoft's Hyperterminal is not one of them. My recommended solution is to disconnect DTR for these non DTR settable terminal emulators. Hence, the two cable flavors −− Parallax (4−wire) and Dumb Terminal Emulator (3−wire).

The next release of our µCL programming language has a full fledged IDE (Integrated Developement Environment) that talks to the PICBrain11. This developement environment uses the Parallax cable (4−wire) connected to the PICBrain11. Whenever the uCL IDE needs to reset the PICBrain11, it asserts DTR to force a reset. Thus, it would be wrong to remove the DTR circuitry on PICBrain11.

3.4 Using the HyperTerminal Terminal Emulator

MicroSoft Windows XP, ships with the HyperTerminal terminal emulator. This terminal emulator can be used to talk to the PICBrain11.

To start HyperTerminal, perform the following:

[Start] => [All Programs] => [Accessories] => [Communications] => [HyperTerminal]

Give the session a name like "RoboBRiX". COM1 is the usual port to talk to.

For Port Settings, select the following:

Port Settings	
Baud Rate	19200
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	None

Now go to the tool bar and select [File] => [Properties]. Select the [Settings] tab and click on the [ASCII Setup...] button. Set line delay to 50 milliseconds.

HyperTerminal Configuration is now complete.

Plug the 3−wire cable into PICBrain11 and the other end into a serial port of the PC running Windows XP. Power up the PICBrain11. If everything is working properly, you should see "PICBrain11−C" followed by "4 3 2 1" in the HyperTerminal window. Press the [Enter] key before the boot loader outputs the final "1" to get it to enter the command prompt.

The boot loader commands are described in the next section.

In order to download a program into the PICBrain11 using HyperTerminal you need to do the following. First, generate an Intel .hex file that contains the program to be downloaded. Next, get the PICBrain11 to the point where a ">" command prompt is showing. Finally use the [Transfer] => [Send Text File ...] command to force the .hex file to be downloaded into the program memory of the PICBrain11. See the ":" command for the boot loader for a little more detail.

4. Software

There is a boot loader preprogrammed into the PICBrain11.

You talk to the boot loader via the serial cable and some sort of terminal emulator running on your preferred host platform (e.g. HyperTerminal for various versions of Microsoft® Windows operating system, *minicom* for Linux, etc.) Configure your terminal emulator for 19200 baud, 8N1 (1 start bit, 8 data bits, No parity, 1 stop bit) and no hardware or software flow control.

When the PICBrain11 Boot loader is powered up it it prints an announcement message of "PICBrain11−C'. Then it counts down for 4 seconds "4 3 2 1". By typing an [Enter] key during the count down, the command prompt is entered. If no [Enter] is depressed, control is transferred to address 0008 after 4 seconds have elapsed.

The command program has the commands listed below. All numbers are in hexadecimal, upper and lower case is unimportant, and commands must be initiated by typing an [Enter] key.

V
: Print version number of software (i.e. "1.0")

K n
: Set the socket to *n*, where *n* must be between 1 and A inclusive.

I
: Talk to the module on the socket specified by the K command (see above.) Print out the module identifier information. The first 24 bytes are in hexadecimal followed by two strings −− the module name and module vendor.

C aa
: Perform a clock adjust on the selected socket (see K command above.) *aa* should be specified as 00. This command will ensure that the clock speed of the selected module is tweaked to be as close to optimal as possible.

P pp
: Show the page of program memory at *pp*00 to *pp*FF. For the PIC16F876, *pp* can range from 00 to 1F.

E rrrr cc
: Show the *cc* bytes of data from file registers starting at *rrrr*. For the PIC16F876A, *rrrr* can range from 0000 to 01FF. Not all register locations are active.

S rrrr vv
: Set register *rrrr* to *vv*. As with the E command (see above), *rrrr* can range from 0000 to 01FF.

G aaaa

Transfer control to address *aaaa*.

X

Transfer control to address 0008.

R cc

Send byte *cc* to the currently selected module (see K command above.) Print up to two bytes of response back. FC is returned for a timeout.

: ...

Program one line of Intel® Hex file format into program memory. All memory from 0004 through 1800 can be programmed. Any attempts to program outside the range are silently ignored. At the end of the download, the number of errors is printed. It should be Err=00. If not, the address where the problem occured is listed. Be sure that your terminal emulator is configured to delay about 50 milliseconds after each line is sent.

The boot loader is written in a programming language called µCL. It currently resides in code bank 3 of the PICBrain11 starting at address 1800. The following files are available:

picbrain11.ucl

The µCL source code for the PICBrain11 boot loader.

picbrain11.asm

The PICBrain11 boot loader assembly code listing

picbrain11.lst

The PICBrain11 boot loader listing file.

picbrain11.hex

The µCL PICBrain11 boot loader Intel® Hex file.

In addition, there is a program that provides some simple robototic behaviors:

robobrix_test.ucl

The µCL source code for the Robobrix_Test test program.

robobrix_test.asm

The Robobrix_Test test program assembly code listing.

robobrix_test.lst

The Robobrix_Test test program listing file.

robobrix_test.hex

The µCL Robobrix_Test test program Intel® Hex file.

The documentation for this program is follows:

There are 11 sockets labeled N1, N2, ..., N9, NA, NB.

There are three simple DualMotor1Amp behaviors:

DualMotor1Amp plugged into N2:

Turns on left motor in forward direction.

DualMotor1Amp plugged into N3:

Turns on right motor in forward direction.

DualMotor1Amp plugged into N4:

Turns on both left and right motors in forward direction.

The next set of behaviors requires the DualMotor1Amp to be plugged into N1:

Proximity2 plugged into N2:

Implements a follow behavior. It does nothing until it sees some object in front of it. Then it turns toward the object and trys to keep a fixed distance from the object.

IRProximity2 plugged into N3:

This is "attack" behavior. I find it to be pretty lame. It waits until an object shows up and runs after it until it runs into it. t is not smart enough to stop. Frankly, this is the lamest behavior in the program right now.

IRProximity2 plugged intO N4:

This is called push−pull behavior. The robot tries to to stay a fixed distance from an object. If object comes closer to the robot, it will back away from the object. Alas, its forward motion is a little too timid right now, so it is mostly a backup robot.

IRPRoximity2 plugged into N5:

This is an object avoider. The jittery nature of the sensor means that it tends to go into circles pretty easily.

IRProximity2 plugged into N6:

This is a right wall follower. The right IR LED needs to be canted about 45 degrees to the right. The trim pot should be cranked way up. It turns left when sees something in front with the left sensor, it turns left; otherwise, it tries to stay in a zone with the wall on the right using the right sensor.

With Servo4 plugged into NA (i.e. N10):

The gripper is plugged into servo0 and the wrist is plugged into servo1. The two trim pots a jumpered on the left. The first trim pot specifies the wrist position when gripping an object. The second trim pot specifies the wrist position when the gripper is searching for an object. While the robot is following a wall, it opens and closes the gripper. If the gripper closes on something, it stays closed until a "corner" is found; then it releases it.

5. Issues

The following fabrication issues came up:

◊ There is no D1 any more (the shorting diode.) D2 should be renamed to D1 everywhere.
◊ The boot loader is chewing up 2K; it should be much smaller.

This is the Revision B verion of the Reckon2 Module. The status of this project is work in progress.

Reckon2 Module (Revision B)

Table of Contents

This document is also available in PDF format.

1. Introduction

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. (Note: deduced reckoning is abbreiated as ded. reckoning and is now frequently refered to by the term "dead reckoning".)

2. Programming

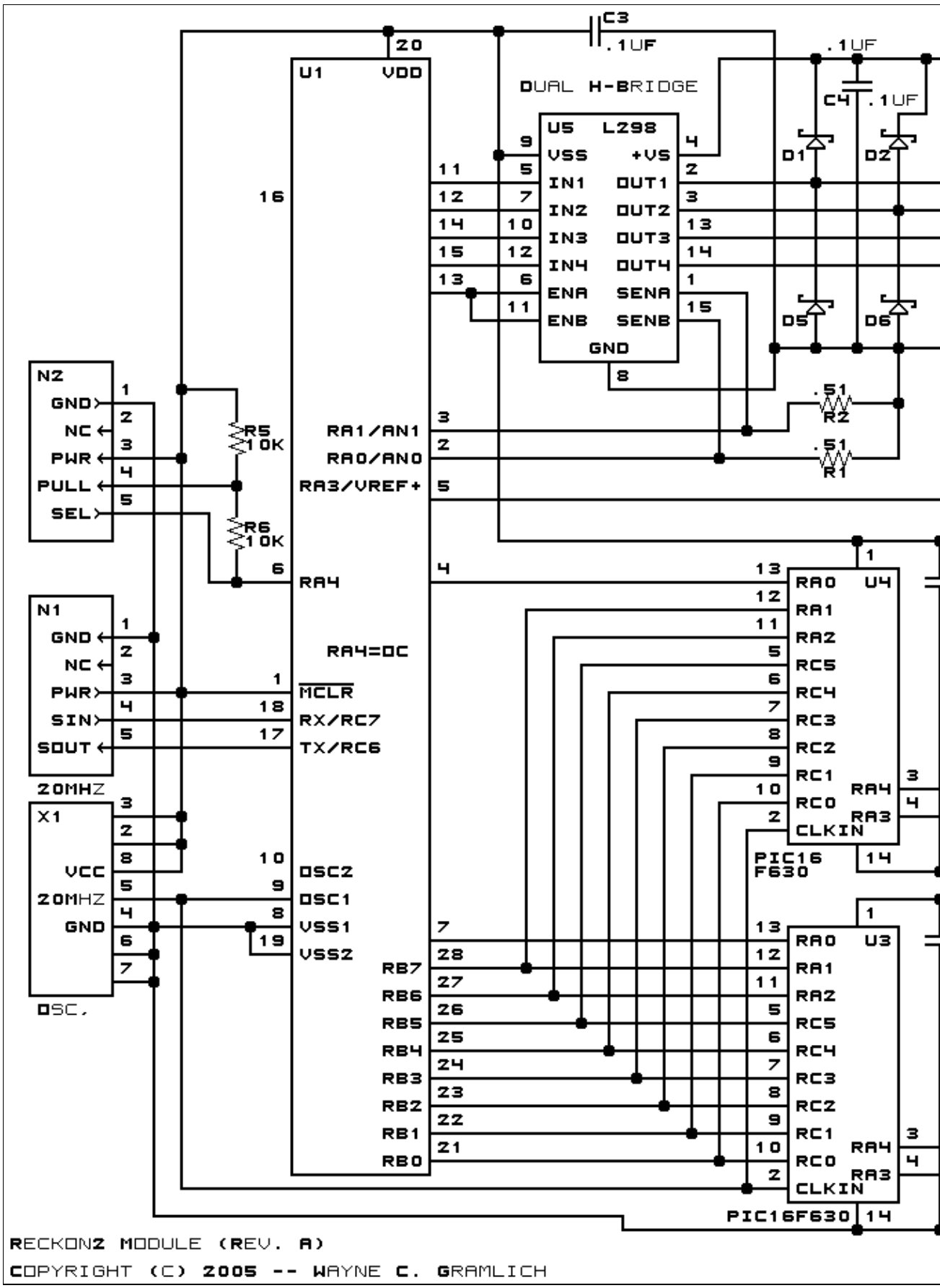
There is no programming yet.

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the Reckon2 Module is shown below:


C3
.1UF
.1UF
C4
.1UF
Z0
U1
VDD
DUAL H-BRIDGE
U5 L298
VSS
+VS
IN1
OUT1
IN2
OUT2
IN3
OUT3
IN4
OUT4
ENA
SENA
ENB
SENB
GND
D1
D2
D5
D6
.51
R2
.51
R1
N2
GND
NC
PWR
PULL
SEL
R5
10K
R6
10K
RA1/AN1
RA0/AN0
RA3/VREF+
RA4
RA4=OC
MCLR
RX/RC7
TX/RC6
N1
GND
NC
PWR
SIN
SOUT
20MHZ
X1
VCC
20MHZ
GND
OSC.
OSC2
OSC1
VSS1
VSS2
RB7
RB6
RB5
RB4
RB3
RB2
RB1
RB0
U4
RA0
RA1
RA2
RC5
RC4
RC3
RC2
RC1
RC0
CLKIN
RA4
RA3
PIC16
F630
U3
PIC16F630
RECKON2 MODULE (REV. A)
COPYRIGHT (C) 2005 -- WAYNE C. GRAMLICH

The parts list kept in a separate file −− reckon2.ptl.

3.2 Printed Circuit Board

The printed circuit board files are listed below:

reckon2_back.png
: The solder side layer.

reckon2_front.png
: The component side layer.

reckon2_artwork.png
: The artwork layer.

reckon2.gbl
: The RS−274X "Gerber" back (solder side) layer.

reckon2.gtl
: The RS−274X "Gerber" top (component side) layer.

reckon2.gal
: The RS−274X "Gerber" artwork layer.

reckon2.drl
: The "Excellon" NC drill file.

reckon2.tol
: The "Excellon" tool rack file.

4. Software

There is no software yet.

5. Issues

The following fabrication issues need to be addressed:

- Switch over to a resonator.
- There is too much interferance between the heat sink and diodes.
- Capacitor C7 does not fit.
- Capacitor C4 would be nicer if .2" lead spacing were used.

This is the Revision B version of the RCInput8 module. The status of this project is work in progress.

RCInput8 Module (Revision B)

Table of Contents

This document is also available as a PDF document.

1. Introduction

The RCInput8 module is used to read the outputs from a standard 8 channel Radio Control unit.

2. Programming

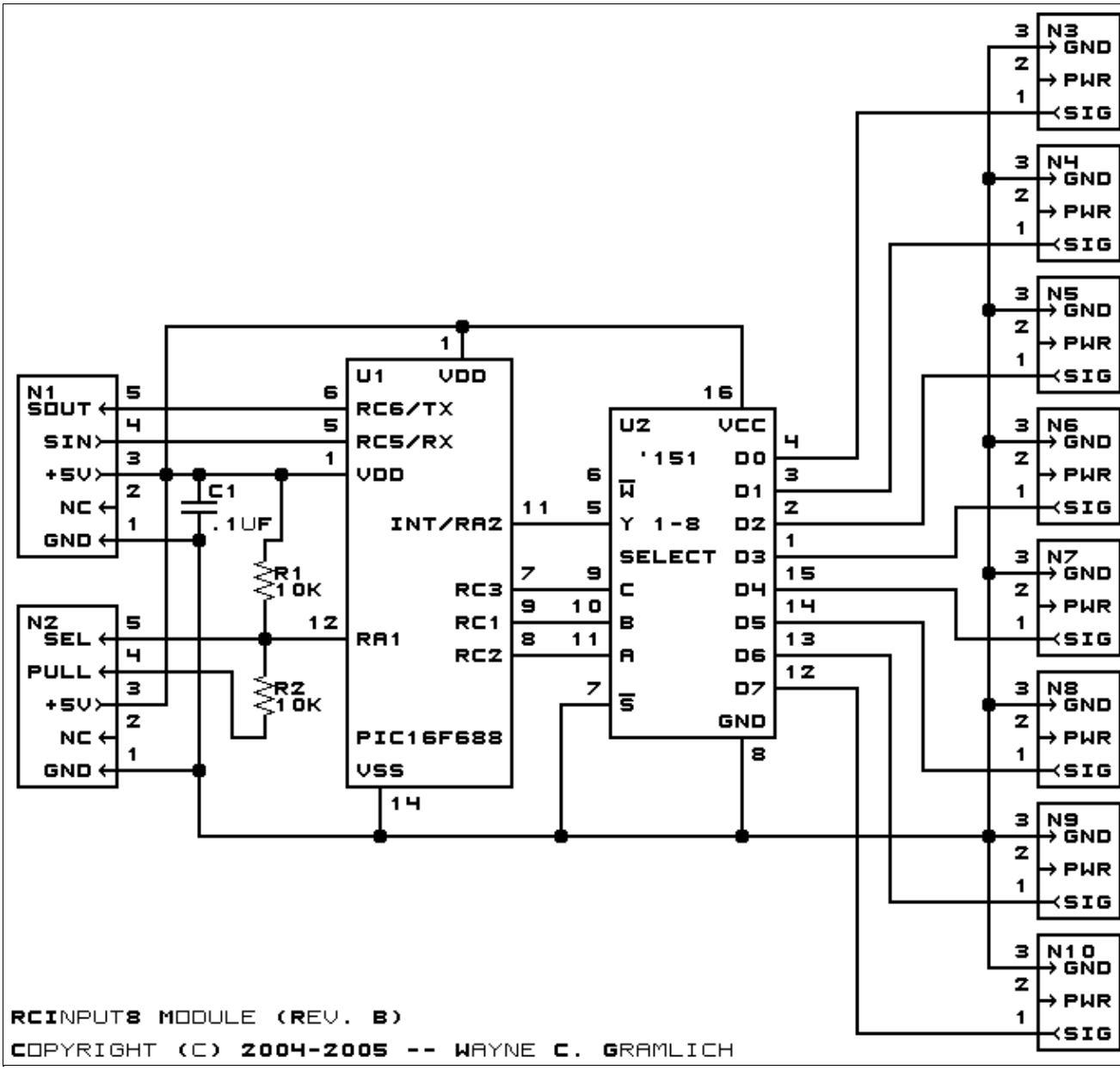
There is no programming specification for the Sonar1 RoboBrick yet.

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the Sonar1 RoboBrick is shown below:



The parts list kept in a separate file −− rcinput8.ptl.

3.2 Printed Circuit Board

The printed circuit board files are listed below:

rcinput8_back.png
: The solder side layer.

rcinput8_front.png
: The component side layer.

rcinput8_artwork.png
: The artwork layer.

rcinput8.gbl
: The RS−274X "Gerber" back (solder side) layer.

rcinput8.gtl
: The RS−274X "Gerber" top (component side) layer.

rcinput8.gal
: The RS−274X "Gerber" artwork layer.

rcinput8.drl
: The "Excellon" NC drill file.

rcinput8.tol
: The "Excellon" tool rack file.

4. Software

The RCInput8 software is available as one of:

rcinput8.ucl
: The µCL source file.

rcinput8.asm
: The resulting human readable PIC assembly file.

rcinput8.lst
: The resulting human readable PIC listing file.

rcinput8.hex
: The resulting Intel® Hex file that can be fed into a PIC12C5xx programmer.

5. Issues

The fabrication issues are listed here.

This is the Revision A version of the ScanBase Module. The status of this project is work in progress.

ScanBase Module (Revision A)

Table of Contents

This document is also available as a PDF document.

1. Introduction

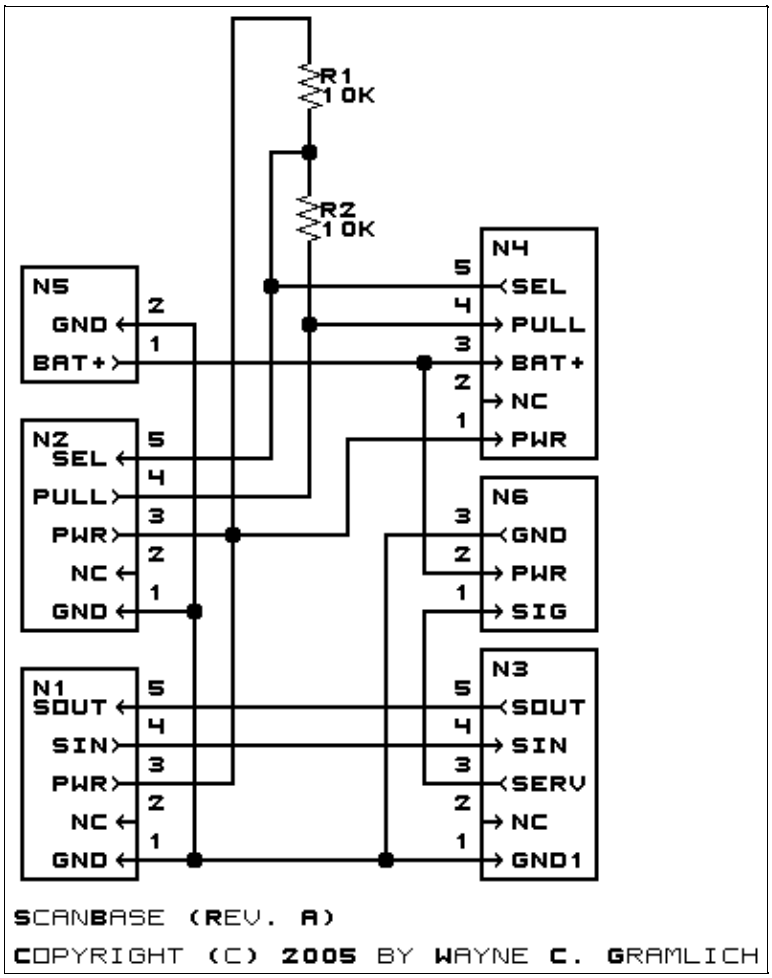
The ScanBase module provides the electrical connection to the ScanPanel module, which, in turn, provides the electrical connection to the Sense3 module.

2. Hardware

...

2.1 Schematic

The schematic is shown below:



The parts list is kept in a separate file.

2.2 Printed Circuit_Board

The printed circuit files are listed below:

scanbase_back.png
: The solder side layer.

scanbase_front.png
: The component side layer.

scanbase_artwork.png
: The artwork layer.

scanbase.gbl
: The RS−274X "Gerber" back (solder side) layer.

scanbase.gtl
: The RS−274X "Gerber" top (component side) layer.

scanbase.gal
: The RS−274X "Gerber" artwork layer.

scanbase.drl
: The "Excellon" NC drill file.

scanbase.tol
: The "Excellon" tool rack file.

3. Issues

Any fabrication issues will be listed here.

This is the Revision A version of the ScanPanel Module. The status of this project is work in progress.

ScanPanel Module (Revision B)

Table of Contents

This document is also available as a PDF document.

1. Introduction

The ScanPanel module is meant to be attached to a standard servo arm to provide a convenient location to attach other modules for scanning purposes. Currently, this module provides the specialized connections between Sense3 and the ScanBase.

Servo arms typically have 3 or 4 holes in a row on each arm. These holes are typically spaced 3 millimeters apart. The initial distance from the center of the servo arm is not standardized. In order to accomadate as many different servo arms as possible, the ScanPanel modules has three sets of holes as shown in the table below:

Angles (degrees)	Hole distances
30, 210	6, 9, 12, 15
90, 270	7, 10, 13, 16
150, 330	8, 11, 14, 17

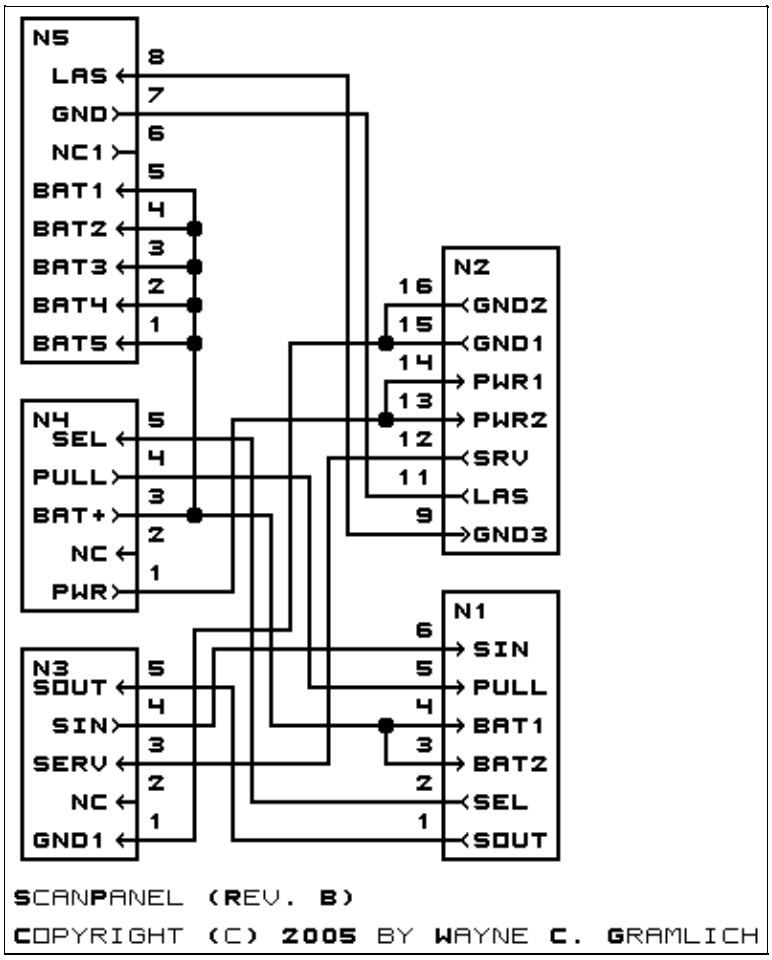
Using a cross style horn, the holes should line up one of the hole sets above.

2. Hardware

...

2.1 Schematic

The schematic is shown below:



The parts list is kept in a separate file.

2.2 Printed Circuit_Board

The printed circuit files are listed below:

scanpanel_back.png
: The solder side layer.

scanpanel_front.png
: The component side layer.

scanpanel_artwork.png
: The artwork layer.

scanpanel.gbl
: The RS−274X "Gerber" back (solder side) layer.

scanpanel.gtl
: The RS−274X "Gerber" top (component side) layer.

scanpanel.gal
: The RS−274X "Gerber" artwork layer.

scanpanel.drl
: The "Excellon" NC drill file.

scanpanel.tol
: The "Excellon" tool rack file.

3. Issues

Any fabrication issues will be listed here.

This is the Revision A verion of the Sense3 Module. The status of this project is work in progress.

Sense3 Module (Revision A)

Table of Contents

This document is also available in PDF format.

1. Introduction

The Sense3 module has three sensors −− an infrared (IR) distance sensor, a sonar distance sensor, and a laser bearing finder. The IR and sonar sensors need no external support, whereas the laser bearing finder needs pieces of reflective tape placed at the correct height at known locations in the environment. The Sense3 module is intended to be placed on top of a common hobby server to provide approximately 180 degrees of sensor sweeping.

This module uses a number of helper modules to accomplish its task. These modules are:

ScanBase
: This module is used to electically connect to the Sense3 module. This module is connected to the robot base and does not sweep back and forth. The hobby servo is electrically connected to this module.

ScanPanel
: This module is mecahnically mounted directly to the top of a hobby servo horn. This module provides electrical connections between ScanBase and Sense3 modules. In addition, one of the LaserHolder1 modules is plugged into this module to provide power for the Laser pointer.

LaserHolder1
: This module is used to mechanically mount and align the small laser pointer so that its laser beam comes out of the appropriate hole of the Sense3 module. Two of these modules. are required plus the Sense3 form a single unit that slips right into the appropriate female connectors on the ScanPanel.

Servo_Adaptor_0.4
: A couple of these modules provide a way to mechanically attach the hobby servo to the robot base.

2. Programming

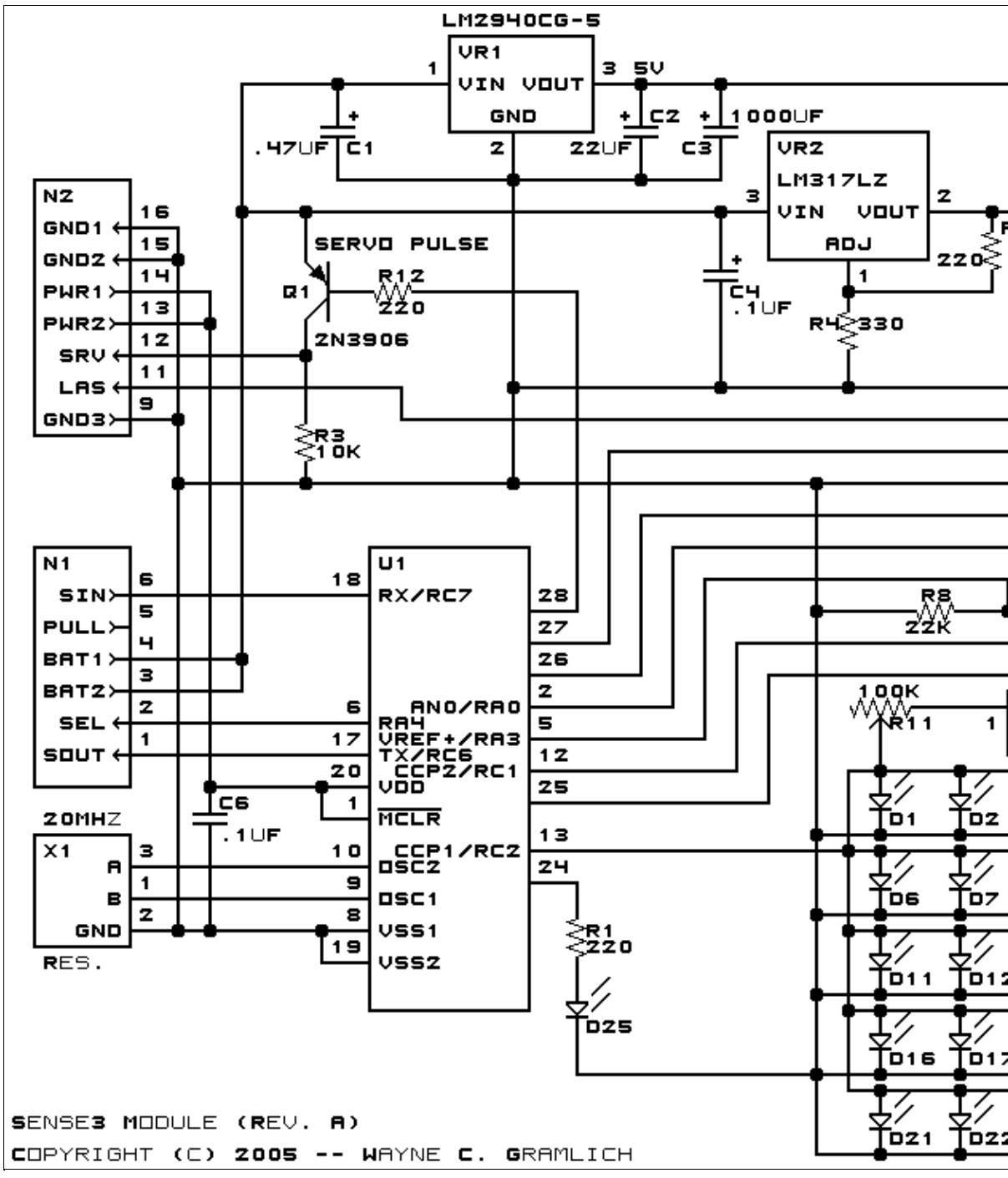
There is no programming yet.

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the Sense3 Module is shown below:



The parts list kept in a separate file -- sense3.ptl.

- Ground
- Regulated 5 Volts
- Unregulated +6 Volts
- Serial In

- Serial Out
- Debug Port(2)

3.2 Printed Circuit Board

The printed circuit board files are listed below:

sense3_back.png
: The solder side layer.

sense3_front.png
: The component side layer.

sense3_artwork.png
: The artwork layer.

sense3.gbl
: The RS−274X "Gerber" back (solder side) layer.

sense3.gtl
: The RS−274X "Gerber" top (component side) layer.

sense3.gal
: The RS−274X "Gerber" artwork layer.

sense3.drl
: The "Excellon" NC drill file.

sense3.tol
: The "Excellon" tool rack file.

4. Software

There is no software yet.

5. Issues

The following issues need to be addressed:

- C3 is mis−labeled in the artwork layer.
- The hole for the laser beam is too small.
- The alignment holes are too small for any reasonable hardware.
- The alignment holes are too close, move further out.
- R10 is too close to Q3.
- Contemplate putting VR1 down flat
- C2 and C1 are too close to VR1.
- C2 and C1 are too close to VR1.
- C4 would be better if the leads were .2".
- Move GP2D120 up to better center it.
- Label the wire colors for the GP2D120 connection to N4.
- Make lead space for C6 be .2".
- Think about using polypropelyne [sp?] tubing for sensor shade. Reposition holes to fit tubing snuggly.

This is the Revision I version of the Servo4 module. The status of this project is finished.

Servo4 Module (Revision I)

Table of Contents

This document is also available as a PDF document.

1. Introduction

The Servo4 module allows for the control of up to 4 hobby grade servos. It can be configured in the following ways:

Pure Servo Mode
: In pure servo mode, it is expected that up to 4 unmodified servos are attached to the board. The four servos can be independently controlled. There is current feedback on all four servos.

Differential Steering Mode
: In differential steering mode, the module can control up to 4 servos. The first two servos are expected to be servos that have modified been for continuous rotation. The second two servos are regular unmodified servos. The first two servos have current feedback and the second two servos do not. There are two trim pots that are used to set the no rotation condition for the first two servos.

MSSC Compatibility Mode
: MSSC stands for Mini−Serial Servro Controller. The MSSC board is quit popular. In MSSC mode, up to 4 servos can be controlled per board. In MSSC compatibility mode, the servos are directly controlled from a host computer via an RS−232 serial port. Up to 16 boards can be chained together to provide control for up to 64 servos. The two trim pots are used to set the "address" of the board.

As you can see, this board is quite flexible. Please see the section on Hardware Configuration to see how the jumpers for each configuration.

2. Hardware Configuration

Up to four RC servos are connected to connectors N2 (servo 0) through N5 (servo3). Each connector has the following pin definitions:

Pin	Location	Description
1	Left	Servo control signal (varies between 0 and 5 volts)
2	Center	5 Volts
3	Right	Ground (0 Volts)

On many servos, the black wire is the ground wire. You will have to check you servo documentation to be absolutely sure though.

The connection to the controlling module occurs via N1 in the upper left corner. Alternatively, in MSSC mode, the connection is via N12.

Power for the servos comes from N6, the blue two terminal connector in the upper right corner. Connect a power source of 6−12 volts to connector N6, where the upper terminal is the positive terminal ('+') and the the lower terminal is negative ('−'). The on board regulator, will regulate the voltage down to 5 volts for the servos.

The hardware configuration for each mode is summarized in the table below:

Mode	Jumpers				Trim Pots	
	N7	N8	N10	N11	R4	R5
Pure Servo Mode	Right (2−3)	Right (2−3)	Right (2−3)	Off	Unused	Unused
Differential Steering Mode	Left (1−2)	Left (1−2)	Right (1−2)	Off	Servo 0 Stop	Servo 1 Stop
Differential Steering Calibration Mode	Left (1−2)	Left (1−2)	Left (1−2)	Off	Servo 0 Stop	Servo 1 Stop
MSSC Compatibility Mode	Left (1−2)	Left (1−2)	Right (1−2)	On	Address A2−A3	Address A4−A5

In differential steering calibration mode, N11 is jumpered to the left and it causes yellow LED D1 to light. It causes both servos 0 and 1 to be enabled. The value of trim pot R5 to be sent to servo 0 and trim pot R6 to be sent to servo 1. The purpose of calibration mode is to allow you to adjust the two modified servos that are connected to servo 0 and servo 1 and adjust them until they stop rotating. This frees the programmer from having to experiment to find the `position' number for each servo that corresponds to each servo being motionless. The values of the stop value are read out using the Read Current Draw command for servo 2 and 3.

In MSSC (Mini Serial Servo Controller) mode, all power comes in from the power source connected to N6 (the blue 2−terminal block.) No cable is connected to N1. In order to talk to the Servo4 module, a cable is constructed from a two pin male header and a female 9−pin DB9 connector. The connections are summerized in the table below:

From	To	Description
DB9 Pin 3	2−pin Header Pin 1	Host Transmitted Data
DB9 Pin 5	2−pin Header Pin 2	Ground (0 Volts)
DB9 Pin 7	DB9 Pin 8	RTS to CTS
DB9 Pin 4	DB9 Pin 1 and DB9 Pin 6	DTR to DSR and DCD

In MSSC (Mini Serial Servo Controller) mode, the servo board controls 4 servos at a time. An address is 6−bits long and is represented as a decimal number between 0 and 63 inclusive as shown below:

aa bb ss

where

aa
: is the two high order address bits and is set by trim pot R5.

bb
: is the two middle order address bits and is set by trim pot R6.

ss
: is the servo specifier and selects between servo 0 through servo 3.

The trim pots are set according to the following table:

Value	Position
00	7:00 (full counter clockwise)
01	10:00
10	2:00
11	5:00 (full clockwise)

Thus, to set the MSSC address to servo bank 0−3, both trim pots are turned full counter clockwise. Similarly, to set the MSSC address to servo bank 60−63, both trim pots are turned full clockwise.

3. Programming

The Servo4 module can independently control up to 4 servos. Each servo has 1) an enable bit and 2) a current position. The position is represented as an 8−bit number. Some experimentation may be needed to determine how the 8−bit numbers correspond to actual servo positions. All servos are initialized to have the enable flags *off*.

In MSSC (Mini Serial Servo Controller) mode, commands are of the form:

nn,ppp[cr]

where

nn
: is a decimal number between 0 and 63, inclusive, that specifies a servo to position.

ppp
: is a decimal number between 0 and 255, inclusive, that specifies the position that the servo is to go to.

[cr]
: is a carriage return character (e.g. decimal ASCII 13) that causes the command to take effect.

An example of a few MSSC commands are shown below, where there is an implicit carriage return after every line:

```
0,128
1,0
255,63,255
```

Note that the last command consists of three comma separated numbers; only the last two numbers are used. Lastly, the receipt of the carriage return automatically enables the servos.

The Servo4 commands are summarized in the table below:

Command	Send/ Receive	Byte Value								Discussion
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
MSSC Command Execute (Carriage Return)	Send	0	0	0	0	1	1	0	1	Set position using previously sent MSSC servo number and position. Enable all four servos.
MSSC Next Number (',')	Send	0	0	1	0	1	1	0	0	Start next MSSC number.
MSSC Decimal Digit ('0'−'9')	Send	0	0	1	1	*d*	*d*	*d*	*d*	Next decimal digit of MSSC number, where *dddd*=0000 corresponds to '0' and *dddd*=1001 is a '9'.
MSSC Ignore	Send	0	0	*x*	*x*	*x*	*x*	*x*	*x*	If 00*xx xxxx* does not match one of the previous MSSC commands, it is simply ignored.
Set High	Send	0	1	*h*	*h*	*h*	*h*	*s*	*s*	Set high order 4 bits of servo *ss* to *hhhh* and set the remaining 4 low order bits to zero.
Set Low	Send	1	0	*l*	*l*	*l*	*l*	*s*	*s*	Set the low order 4 bits of servo *ss* position to *llll*.
Set Enable and Position	Send	1	1	0	0	0	*e*	*s*	*s*	Select servo *ss* and set its position to *pppppppp* and enable flag to *e*.
	Send	*p*	*p*	*p*	*p*	*p*	*p*	*p*	*p*	
Set Enable Flag Only	Send	1	1	0	0	1	*e*	*s*	*s*	Select servo *ss* and set its enable flag to *e*.
Read Position	Send	1	1	0	1	0	0	*s*	*s*	Return the current position *pppppppp* for servo *ss*.
	Receive	*p*	*p*	*p*	*p*	*p*	*p*	*p*	*p*	
Read Enable	Send	1	1	0	1	0	1	*s*	*s*	Return the enable bit *e* for servo *ss*.
	Receive	0	0	0	0	0	0	0	*e*	
Read Enables	Send	1	1	0	1	1	0	0	0	Return the enable flags *eeee* for all four servos.
	Receive	0	0	0	0	*e*	*e*	*e*	*e*	
Set Enables	Send	1	1	0	1	1	0	0	1	Set enable flags for all four servos to *eeee*.
	Send	0	0	0	0	*e*	*e*	*e*	*e*	
Read Current Draw	Send	1	1	0	1	1	1	*s*	*s*	Return the *aaaaaaaa* current draw for servo *ss*.
	Receive	*a*	*a*	*a*	*a*	*a*	*a*	*a*	*a*	
Shared Commands	Send	1	1	1	1	1	*c*	*c*	*c*	Execute shared command *ccc*.

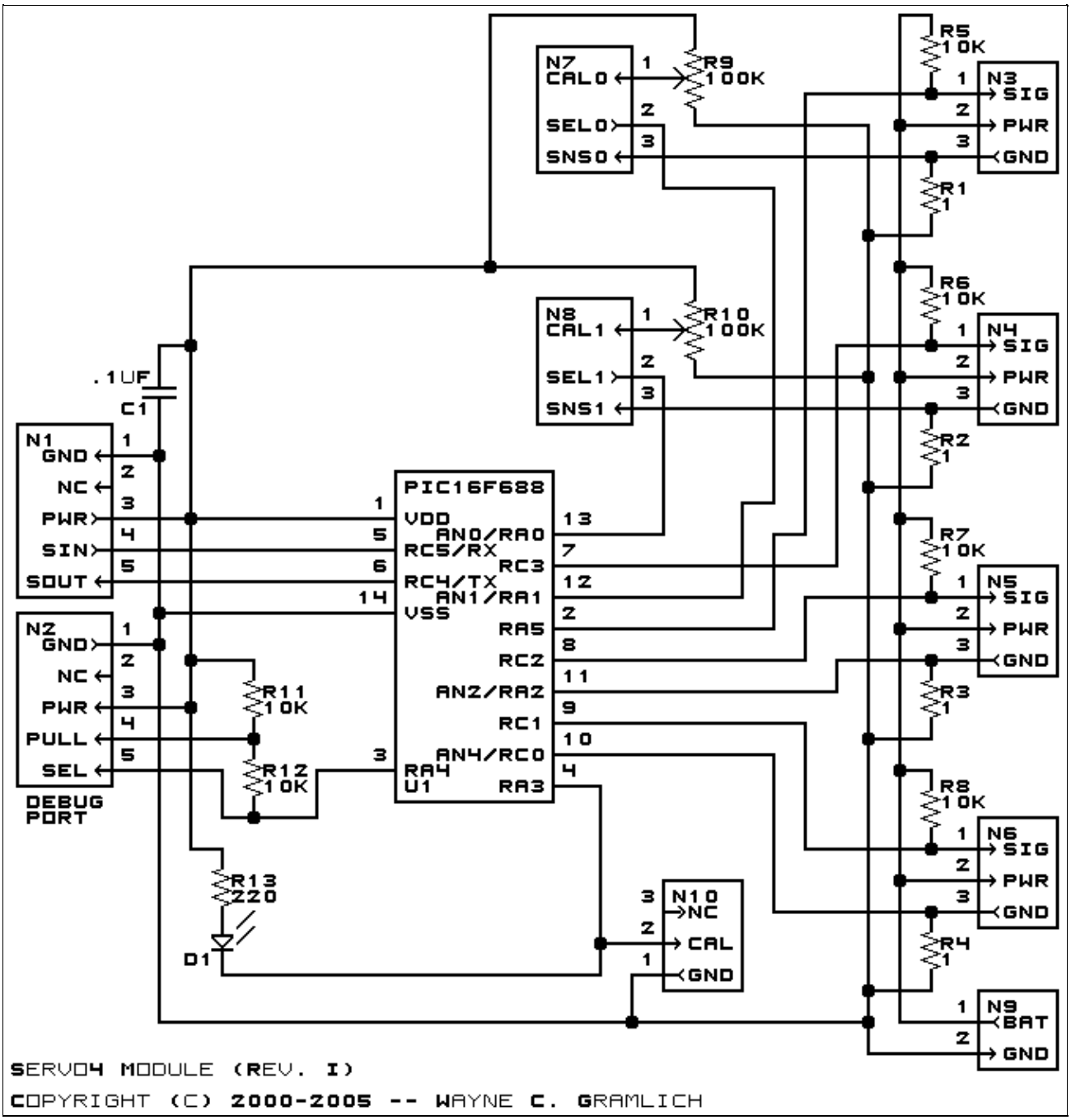
The Servo4 module does *not* know the minimum and maximum extent for each servo. This has to be determined by experimentation.

4. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

4.1 Circuit Schematic

The schematic for the Servo4 module is shown below:



The parts list kept in a separate file −− servo4.ptl.

4.2 Printed Circuit Board

The printed circuit board files are listed below:

servo4_back.png
: The solder side layer is shown below:

servo4_front.png
: The component side layer is shown below:

servo4_artwork.png

The artwork layer is shown below

servo4.gbl
: The RS−274X "Gerber" back (solder side) layer.

servo4.gtl
: The RS−274X "Gerber" top (component side) layer.

servo4.gal
: The RS−274X "Gerber" artwork layer.

servo4.drl
: The "Excellon" NC drill file.

servo4.tol
: The "Excellon" tool rack file.

5. Software

The Servo4 software is available as one of:

servo4.ucl
: The µCL source file.

servo4.asm
: The resulting human readable PIC assembly file.

servo4.lst
: The resulting human readable PIC listing file.

servo4.hex
: The resulting Intel® Hex file.

6. Issues

The following issues have come up:

- In my zeal to remove the L2940 voltage regulator from the design, I totally screwed up. The 10K pull−up resistors exceed the V_{IH} for control pins.

This is the Revision C version of the Serial1 Module. The status of this project is work in progress.

Serial1 Module (Revision A)

Table of Contents

This document is also available as a PDF document.

1. Introduction

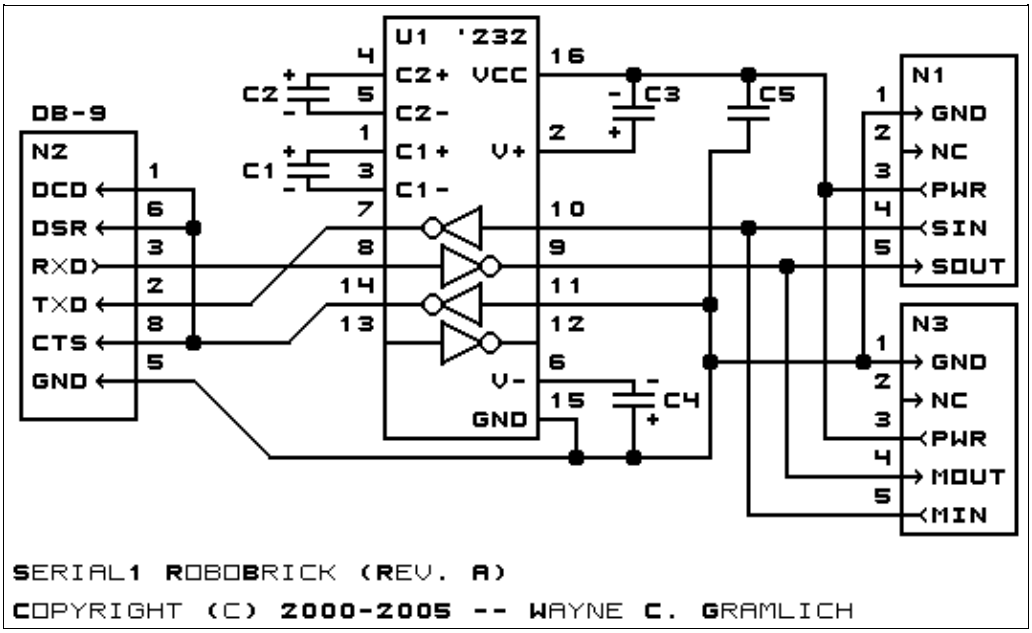
The Serial1 Module is a Module that connects a master Module to a computer via a stanadard 4−wire telephone cord extension.

2. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

2.1 Circuit Schematic

The schematic for the Serial1 Module is shown below:



The parts list kept in a separate file −− serial1.ptl.

2.2 Printed Circuit Board

The printed circuit board files are listed below:

serial1_back.png
: The solder side layer is shown below:

serial1_front.png
: The component side layer is shown below:

serial1_artwork.png
: The optional artwork layer is shown below:

serial1.gbl
: The RS−274X "Gerber" back (solder side) layer.

serial1.gtl
: The RS−274X "Gerber" top (component side) layer.

serial1.gal
: The RS−274X "Gerber" artwork layer.

serial1.drl
: The "Excellon" NC drill file.

serial1.tol
: The "Excellon" NC drill rack file.

3. Issues

The following fabrication issues need to be addressed:

- Use electrolytic capacitors instead of tantalum capacitors.
- Perhaps move the connector in some.

This is the Revision C version of the Sonar8 module. The status of this project is work in progress.

Sonar8 Module (Revision C)

Table of Contents

This document is also available as a PDF document.

1. Introduction

The Sonar8 module is used to send and receive ultra sonic sonar pulses using the Robot Electronics SRF04 sonar ranging module. Up to 8 SRF04's can be controlled by this module.

2. Programming

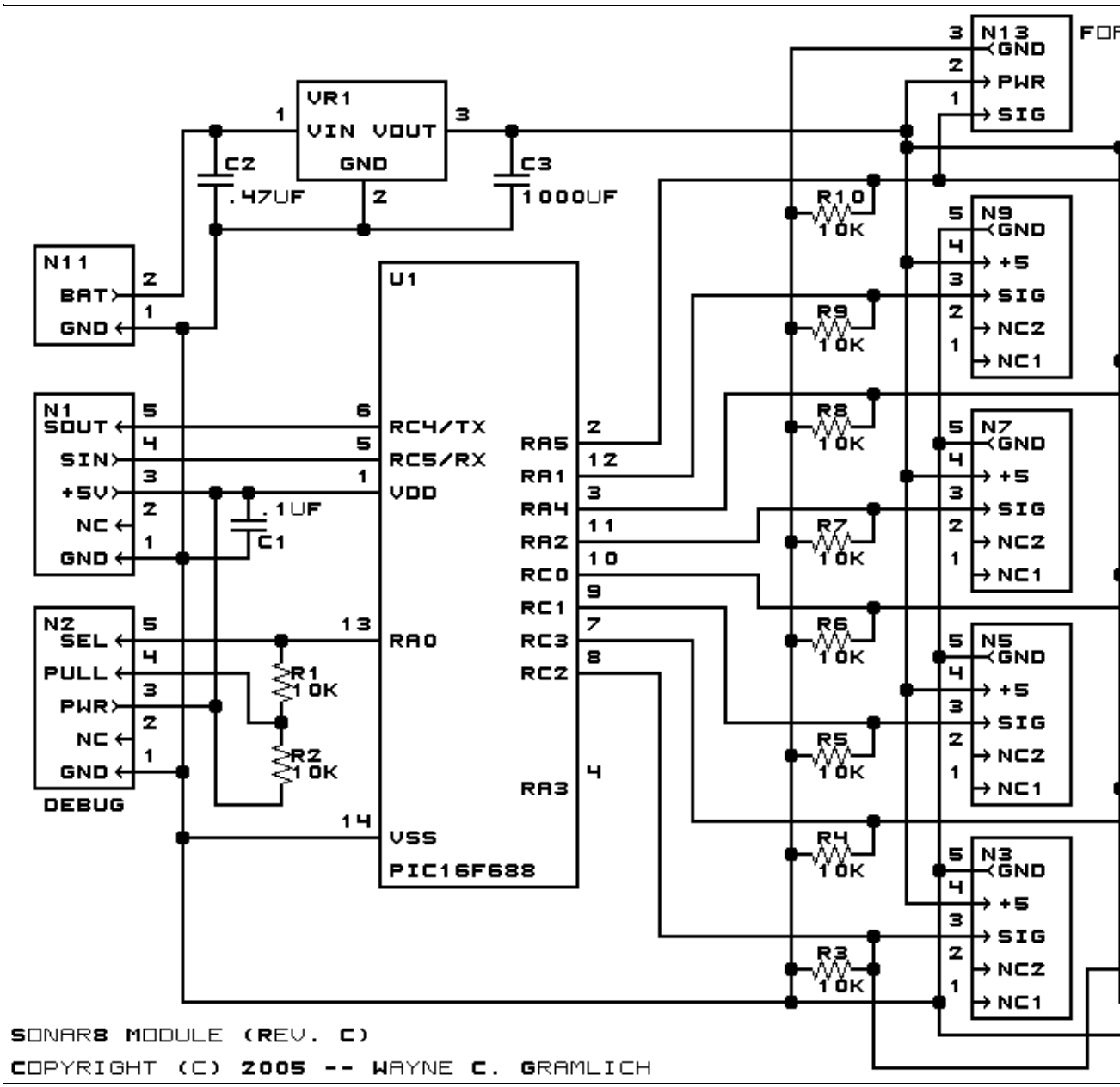
There is no programming specification for the Sonar1 RoboBrick yet.

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the Sonar1 RoboBrick is shown below:



The parts list kept in a separate file −− sonar8.ptl.

3.2 Printed Circuit Board

The printed circuit board files are listed below:

sonar8_back.png
: The solder side layer.

sonar8_front.png
: The component side layer.

sonar8_artwork.png
: The artwork layer.

sonar8.gbl
: The RS−274X "Gerber" back (solder side) layer.

sonar8.gtl
: The RS−274X "Gerber" top (component side) layer.

sonar8.gal
: The RS−274X "Gerber" artwork layer.

sonar8.drl
: The "Excellon" NC drill file.

sonar8.tol
: The "Excellon" tool rack file.

4. Software

The Sonar8 software is available as one of:

sonar8.ucl
: The µCL source file.

sonar8.asm
: The resulting human readable PIC assembly file.

sonar8.lst
: The resulting human readable PIC listing file.

sonar8.hex
: The resulting Intel® Hex file that can be fed into a PIC12C5xx programmer.

5. Issues

The following issues came up during fabrication:

- Connector N11 does *not* interfere with the Ping sensor plugged into N12. Thus, N11 can be moved to the right edge of the board.
- The capacitor C3 is a problem in its current location. First, it interferes with the PIC16F688 in socket U1. Second, it interferes with the Ping when it is plugged into N12. Third, there is currently only room for an 8mm round capacitor.
- Seriously consider making R3−R10 vertical, to allow U1 to be placed into a veritical mode. This should clear up space for the power supply and allow a larger C3 capacitor.

This is the Revision F verion of the Switch8 Module. The status of this project is finished.

Switch8 Module (Revision F)

Table of Contents

This document is also available as a PDF document.

1. Introduction

The Switch8 Module allows you to read up to 8 digital inputs. An interrupt can be generated on the states of selected inptus.

2. Programming

The basic operation is to send a query to the Switch8 Module to read the 4 bits of data. The programmer can download a complement mask to cause any of the bits to be complemented prior to reading.

The Switch8 Module supports Module Interrupt Protocol. The interrupt pending bit is set whenever the the formula:

L&(~I) | H&I | R&(~P)&I | F&P&(~I)]

is non−zero, where:

- I is the current input bits XOR'ed with the complement mask (C)
- P is the previous value of I
- L is the low mask
- H is the high mask
- R is the raising mask
- F is the falling mask

and

- ~ is bit−wise complement
- | is bit−wise OR
- & is bit−wise AND

Once the interrupt pending bit is set, it must be explicitly cleared by the user.

The Switch8 Module supports both the standard shared commands and the shared interrupt commands in addition to the following commands:

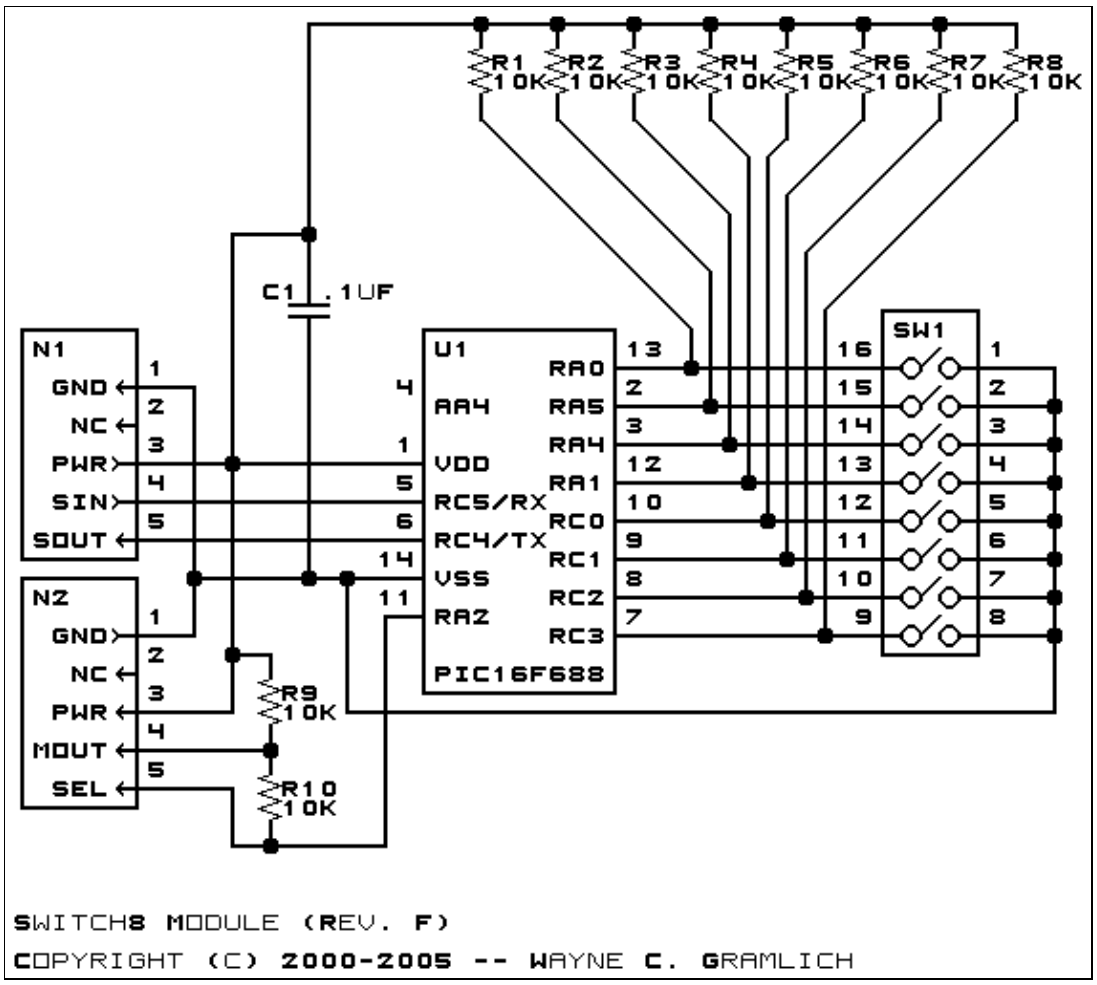
Command	Send/ Receive	Byte Value								Discussion
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
Read Inputs	Send	0	0	0	0	0	0	0	0	Return input values *abcdefgh* (after XOR'ing with complement mask)
	Receive	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	
Read Complement Mask	Send	0	0	0	0	0	0	0	1	Return complement mask *cccccccc*
	Receive	*c*	*c*	*c*	*c*	*c*	*c*	*c*	*c*	
Read Low Mask	Send	0	0	0	0	0	0	1	0	Return low mask *llllllll*
	Receive	*l*	*l*	*l*	*l*	*l*	*l*	*l*	*l*	
Read High Mask	Send	0	0	0	0	0	0	1	1	Return high mask *hhhhhhhh*
	Receive	*h*	*h*	*h*	*h*	*h*	*h*	*h*	*h*	
Read Raising Mask	Send	0	0	0	0	0	1	0	0	Return raising mask *rrrrrrrr*
	Receive	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	
Read Falling Mask	Send	0	0	0	0	0	1	0	1	Return falling mask *ffffffff*
	Receive	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	
Read Raw	Send	0	0	0	0	1	0	0	0	Return raw data *abcd* (without XOR'ing with complement mask)
	Receive	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	
Set Complement Mask	Send	0	0	0	0	1	0	0	1	Set complement mask to *cccccccc*
	Send	*c*	*c*	*c*	*c*	*c*	*c*	*c*	*c*	
Set Low Mask	Send	0	0	0	0	1	0	1	0	Set low mask to *llllllll*
	Send	*l*	*l*	*l*	*l*	*l*	*l*	*l*	*l*	
Set High Mask	Send	0	0	0	0	1	0	1	1	Set high mask to *hhhhhhhh*
	Send	*h*	*h*	*h*	*h*	*h*	*h*	*h*	*h*	
Set Raising Mask	Send	0	0	0	0	1	1	0	0	Set raising mask to *rrrrrrrr*
	Send	*r*	*r*	*r*	*r*	*r*	*r*	*r*	*r*	
Set Falling Mask	Send	0	0	0	0	1	1	0	1	Set falling mask to *ffffffff*
	Send	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	
Read Interrupt Bits	Send	1	1	1	0	1	1	1	1	Return the interrupt pending bit *p* and the interrupt enable bit *e*.
	Receive	0	0	0	0	0	0	*e*	*p*	
Set Interrupt Commands	Send	1	1	1	1	0	*c*	*c*	*c*	Set Interrupt Command *ccc*.
Shared Commands	Send	1	1	1	1	1	*c*	*c*	*c*	Execute Shared Command *ccc*.

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the Switch8 Module is shown below:



The parts list kept in a separate file −− switch8.ptl.

3.2 Printed Circuit Board

The printed circuit board files are listed below:

switch8_back.png
: The solder side layer.

switch8_front.png
: The component side layer.

switch8_artwork.png
: The artwork layer.

switch8.gbl
: The RS−274X "Gerber" back (solder side) layer.

switch8.gtl
: The RS−274X "Gerber" top (component side) layer.

switch8.gal
: The RS−274X "Gerber" artwork layer.

switch8.drl
: The "Excellon" NC drill file.

switch8.tol
: The "Excellon" tool rack file.

4. Software

The Switch8 software is available as one of:

switch8.ucl
: The µCL source file.

switch8.asm
: The resulting human readable PIC assembly file.

switch8.lst
: The resulting human readable PIC listing file.

switch8.hex
: The resulting Intel® Hex file.

5. Issues

Any fabrication issues are listed here.

This is the Revision D version of the TwinGearSensorLeft RoboBrick. The status of this project is work in progress.

TwinGearSensorLeft Robobrick (Revision D)

Table of Contents

This document is also available in PDF format.

1. Introduction

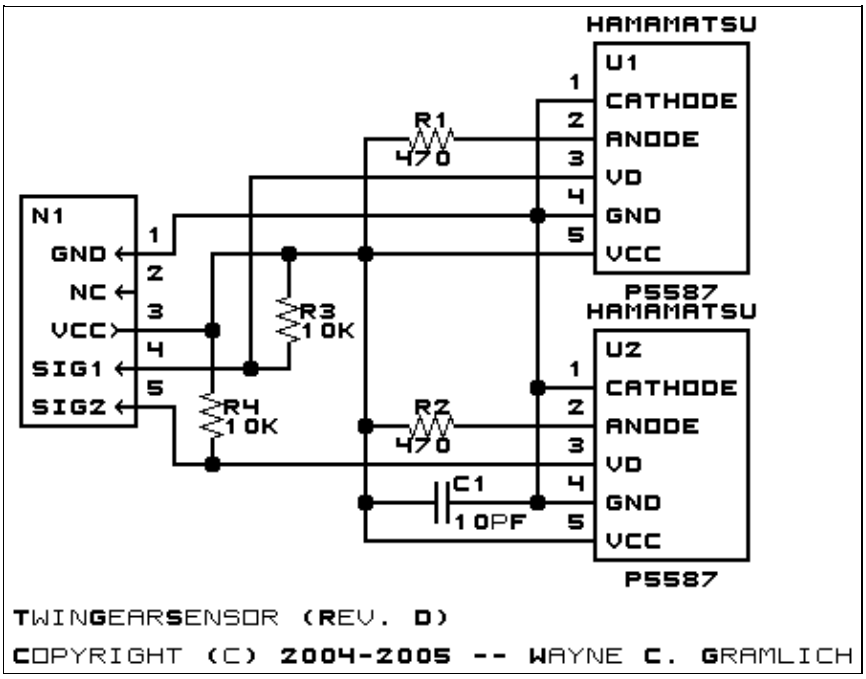
The TwinGearSensorLeft board is designed to pick up a quadrature signal from a shaft using two Hamamatsu P5507 chips.

2. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

2.1 Circuit Schematic

The schematic for the TwinGearSensorLeft RoboBrick is shown below:



The parts list kept in a separate file −− twingearsensorleft.ptl.

2.2 Printed Circuit Board

The printed circuit board files are listed below:

twingearsensorleft_back.png
: The solder side layer.

twingearsensorleft_front.png
: The component side layer.

twingearsensorleft_artwork.png
: The artwork layer.

twingearsensorleft.gbl
: The RS−272X "Gerber" back (solder side) layer.

twingearsensorleft.gtl
: The RS−272X "Gerber" top (component side) layer.

twingearsensorleft.gal
: The RS−272X "Gerber" artwork layer.

twingearsensorleft.drl
: The "Excellon" NC drill file.

twingearsensorleft.tol
: The "Excellon" tool rack file.

5. Issues

Any fabrication issues that come up are listed here.

This is the Revision D version of the TwinGearSensorRight RoboBrick. The status of this project is work in progress.

TwinGearSensorRight Robobrick (Revision D)

Table of Contents

This document is also available in PDF format.

1. Introduction

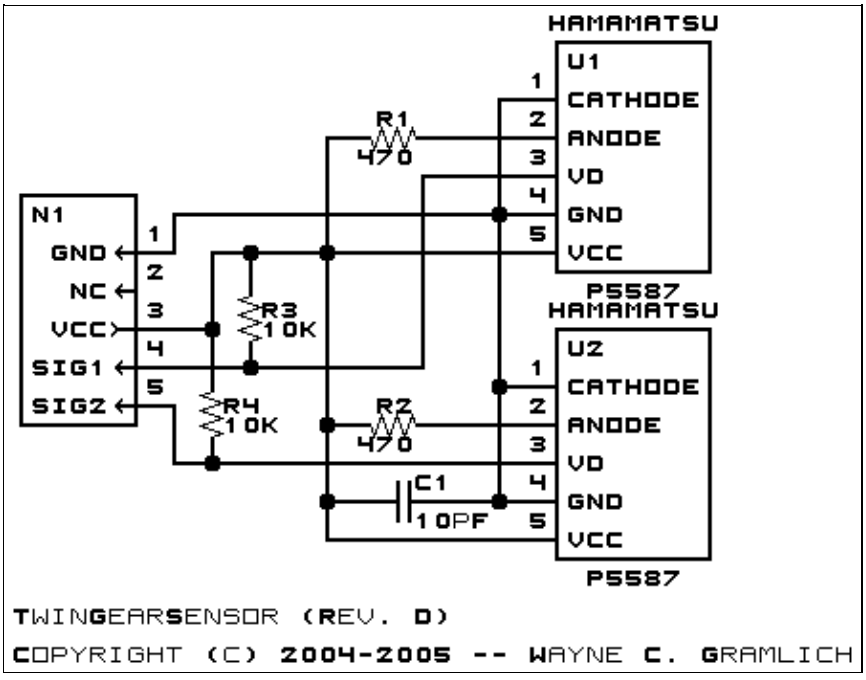
The TwinGearSensorRight board is designed to pick up a quadrature signal from a shaft using two Hamamatsu P5507 chips.

2. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

2.1 Circuit Schematic

The schematic for the TwinGearSensorRight RoboBrick is shown below:



The parts list kept in a separate file −− twingearsensorright.ptl.

2.2 Printed Circuit Board

The printed circuit board files are listed below:

twingearsensorright_back.png
: The solder side layer.

twingearsensorright_front.png
: The component side layer.

twingearsensorright_artwork.png
: The artwork layer.

twingearsensorright.gbl
: The RS−272X "Gerber" back (solder side) layer.

twingearsensorright.gtl
: The RS−272X "Gerber" top (component side) layer.

twingearsensorright.gal
: The RS−272X "Gerber" artwork layer.

twingearsensorright.drl
: The "Excellon" NC drill file.

twingearsensorright.tol
: The "Excellon" tool rack file.

5. Issues

Any fabrication issues that come up are listed here.
