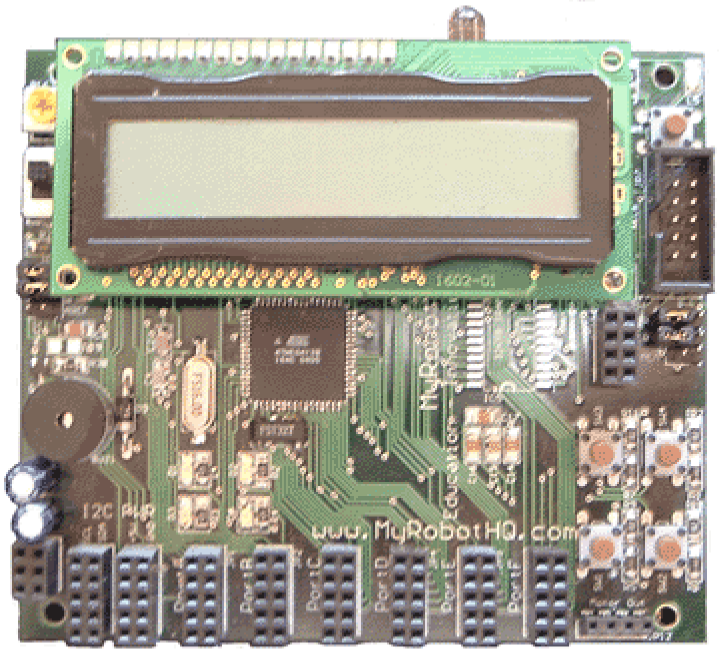
MyRobotHQ™

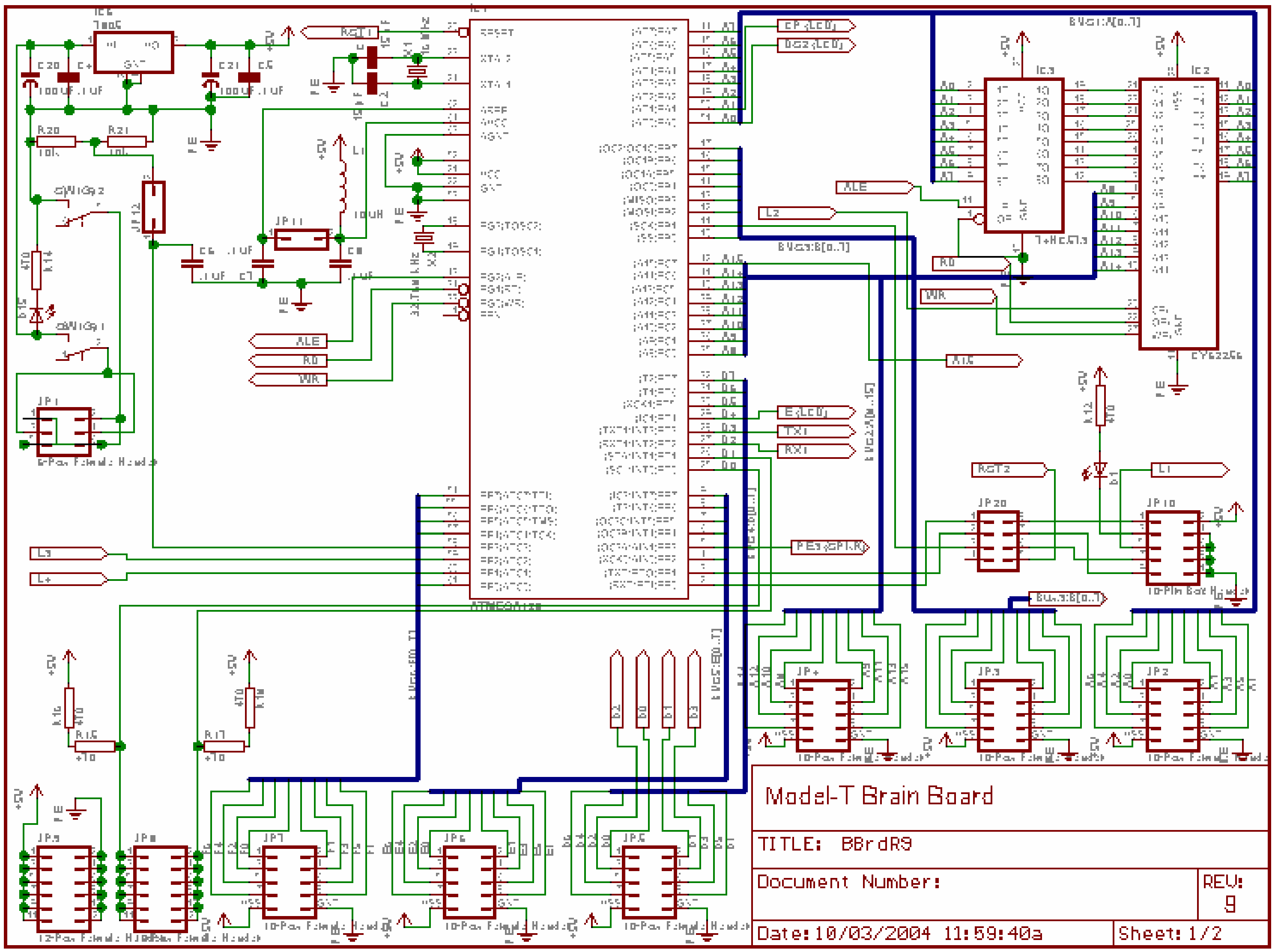
Education ♦ Innovation ♦ Entrepreneurialship



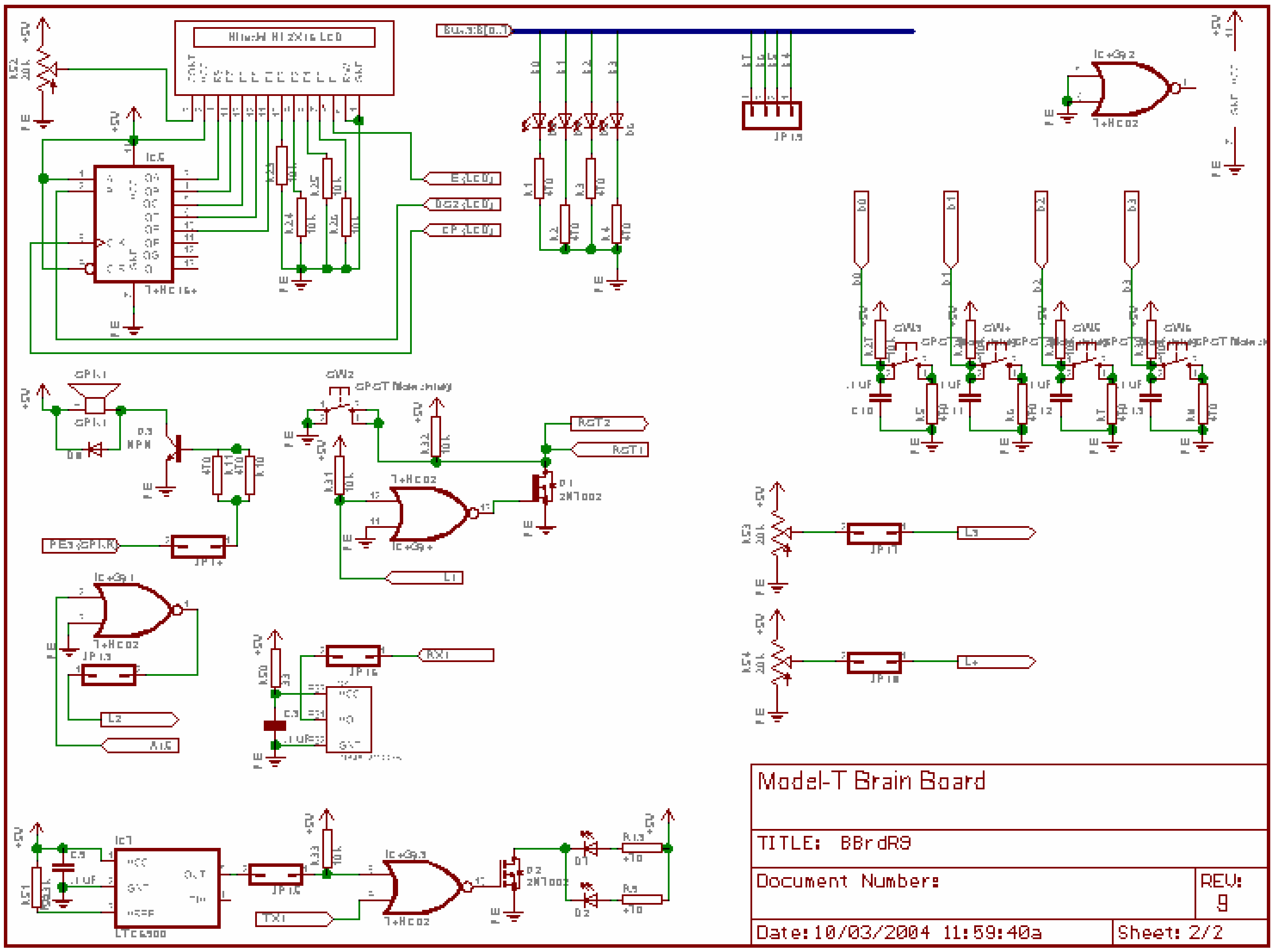
Mini-Computer User Manual

Document Version – 1.0

Table of Contents



Document Version – 1.0


Model-T Brain Board
TITLE: BBrdR9
Document Number:
REV: 9
Date: 10/03/2004 11:59:40a
Sheet: 2/2
Hitachi 2X16 LCD
74HC164
74HC02
2N7002
LTC6900
NPN
SPK1
E(LCD)
DS2(LCD)
CP(LCD)
RST2
RST1
RX1
TX1
L1
L2
L3
L4
AIG
PE3(SPKR)
JP13
JP14
JP15
JP16
JP17
JP18
SW2
SW3
SW4
SW5
SW6
SPST Momentary
+5V
PE
D1
D2
D3
D7
D8
R13
R9
470

Introduction

The MyRobot Mini-Computer is just that – a “mini” computer. It has all the features and functionality of a computer, but on a smaller level. The small size and simpler functionality make it the ideal ‘brain’ for your robot. And thanks to an ever expanding and easy to use open-source library of function calls, the Mini-Computer board makes controlling your robot very uncomplicated.

Installing GNU Tools and other IDE Software

Hardware

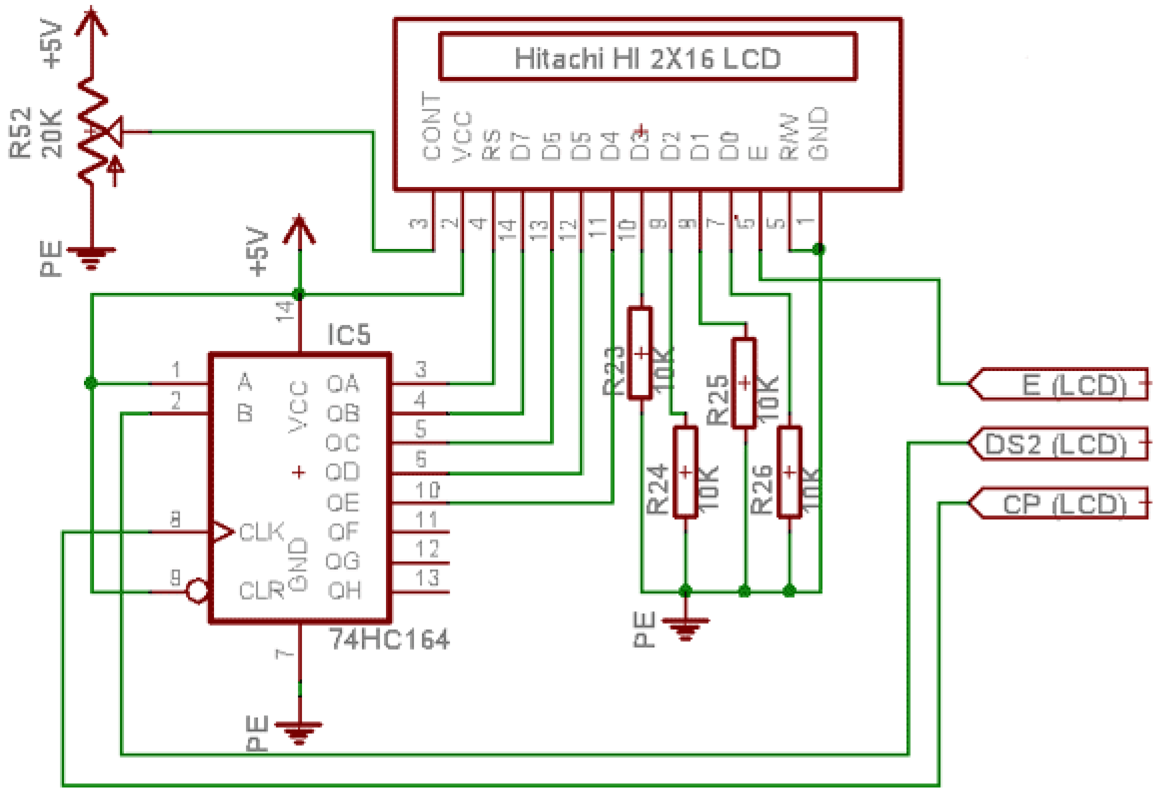
Each sub-section below describes a different hardware module built into yoru mini-computer board. They are each broken down into 5 parts: Theory of operation, schematic, circuit description, example code, and conclusion.

Due to space considerations, the example code section contains the location of the example code on the CD and website instead of printing them here in the manual You may find it usefull to read and study each section in the order it is presented as the hardware concepts and example code will become increasingly complex and build off of previously explained modules and example code.

3.1 - 2 Line X 16 Character LCD Display

With the MyRobot Mini-Computer board, your robot now has several ways of giving you feedback. However, probably none of them will be as useful or easy to use as the Hitachi 2X16 LCD display. This section describes the physical communication layer that allows IC1 (the microcontroller) to output data to the LCD. The myrobot.h library contains several easy to use functions to allow you to display any type of data on the LCD. The example code below shows you how to use them, and each function is explained in further detail in the Software section.

Schematic:



Circuit Description:

R52 is a 20 kilo-ohm variable resistor (also known as a potentiometer) which is used to control the contrast of the LCD. As you turn R52, its resistance changes, and simultaneously the darkness of the LCD display is lightened or darkened.

The Hitachi 16X2 LCD is attached to IC1 (ATMega128 microcontroller) through IC5 which is a 74HC164 Serial-to-Parallel Shift Register. Without IC5, six I/O lines would be required to control the LCD. Instead however, we can control the LCD using only three lines: E, DS2, & CP. The trade off is that this configuration makes access a little slower, and our control software a little more complicated. Luckily for you though, all the access software has been compiled into function calls, and IC1 and IC5 run so fast, they will still end up waiting on the LCD anyways.

Typically, most LCDs can be controlled using either 8-lines or 4-lines. The four resistors (R23 – R26) configure the LCD for 4-line control. To transfer data to the LCD, 5 bits are shifted into IC5 using the DS2 (Data) and CP (Clock) lines. These lines are connected to PIN-51 (PA0) and PIN-50 (PA1) – see Table 3.1.1.

Once all 5 bits have been shifted in and are sitting on the Qn output pins, the Enable line connected to PIN-29 of IC1 (PD4) is pulsed to clock the data into the LCD. It's important to note here that the propagation delay on IC5 (the time it takes data presented to an input pin to make it to an output pin) is much smaller than the propagation delay for the LCD to read in the data and output it on the display. This is what allows the three-wire configuration to be just as fast, effectively, as the standard six-wire configuration.

IC1		IC5		LCD		Description
Pin	Name	Pin	Name	Pin	Name	
51	PA0	2	B	-	-	Bus Name: DS2(LCD)
50	PA1	8	CLK	-	-	Bus Name: CP(LCD)
29	PD4	-	-	5	E	Bus Name: E(LCD)
-	-	3	QA	4	RS	Command or Data Select
-	-	4	QB	14	D7	Data (MSB)
-	-	5	QC	13	D6	Data
-	-	6	QD	12	D5	Data
-	-	10	QE	11	D4	Data (LSB)

Table 3.1.1 - Pin Connection Between IC1, IC5, & LCD

	Propagation Delay
IC5	20 nS
LCD	40 uS – 1.64 mS

Table 3.1.2 - Delay Time for IC5 and LCD

Example Code:

X:\MRMC\user_manual\code_examples\lcd_xmpl.c
www.myrobot.com\minicomputer\code\examples\lcd_xmpl.c

Conclusion:

If you ever turn on your board and nothing is being displayed on the LCD, try turning R52 to change the contrast. You will notice with use that if your batteries are low, your display will become lighter and lighter. After a while, tweaking with R52 won't help anymore and you'll need to recharge your batteries. If this does not help, make sure your have an lcd_init() statement at the beginning of your code (see the example code).

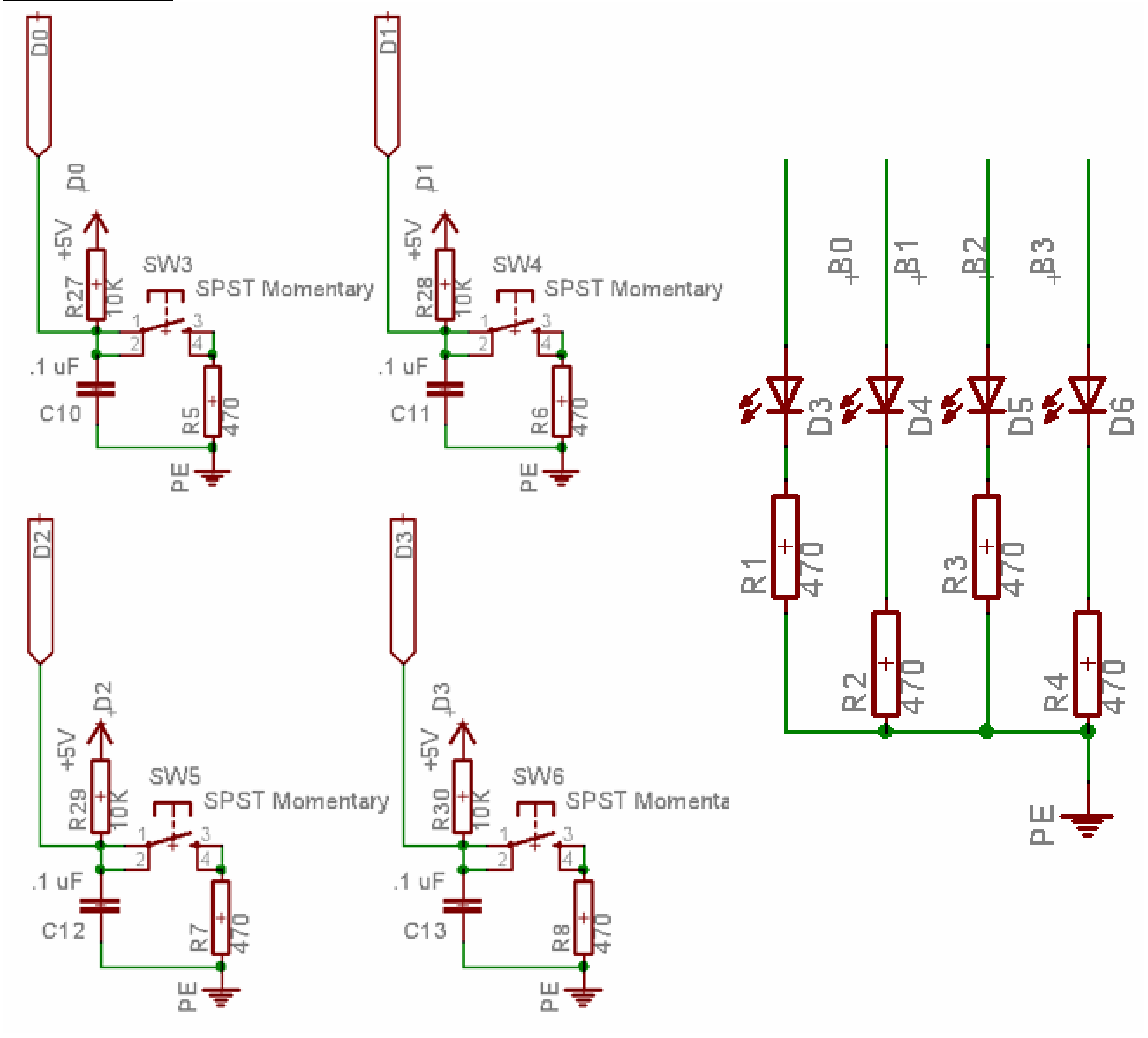
More Information:

SN74HC164 Data Sheet – www.myrobothq.com/datasheets/sn74hc164.pdf
LCD Data Sheet - www.myrobothq.com/datasheets/HDM16216H-5.pdf
Hantronix Web Site - www.hantronix.com
How-To Control an LCD – home.iae.nl/users/pouweha/lcd/lcd.shtml

User Programmable LEDs

The four user programmable LEDs allow another way for your robot to give you feedback, and the four user programmable switches allow you to give feedback to your robot. Each switch is connected to a hardware interrupt. This configurations allows you to use them for a wide variety of ways: such as stepping through code for debugging purposes, implementing a menu system, and many other things.

Schematic:



Circuit Description:

The LEDs are connected to the first four pins of PortB (PB0-PB3). The four switches are connected to the first four pins of PortD (PD0-PD3) which can be configured as general input or output, or as external interrupts (INT0-INT3). It is also important to note that PD2 (SW5) is connected to the infrared receiver when its jumper is in place and PD3 (SW6) is connected to the infrared transmitter when its jumper is in place. Some of the PortB pins connected to the

LEDs are used when programming the board, so you'll notice that they light up during that time.

The hardware for each switch is identical. The line that runs to a PortD pin is connected to a 10 K-ohm pull-up resistor that maintains +5 volts (logic 1) on the line when PortD is configured as an input and the switch is not being pushed. A capacitor is also placed between the line and ground which protects it from electro-magnetic radiation that can create a false trigger (i.e. making the microcontroller think the switch was pushed when it really wasn't).

When the switch is pushed, it connects the line to the 470 ohm resistor which is in turn connected to ground. Since the 470 ohm resistor is much smaller than the 10 K-ohm resistor, it drives 0 volts (logic 0) on the line. This transition from logic 1 to logic 0 can trigger the external interrupt INTn when IC1 is configured in software for it. The capacitor at this point also helps to protect against mechanical bouncing which can actually cause the interrupt to trigger several times with just one button push.

The LEDs are simply connected between a pin on PortB and a 470 ohm resistor connected to ground. Whenever 5 volts (logic 1) is driven on the line (either by IC1 or some other external circuitry like the programmer) the LED will light up. Note that in order to drive a high with IC1, the appropriate DDRB bit must be set to activate the pin as an output and the appropriate PORTB bit must be set to drive a high within software – see the example code.

Example Code:

X:\MRMC\user_manual\code_examples\led_but_xmpl.c
www.myrobot.com\minicomputer\code\examples\led_but_xmpl.c

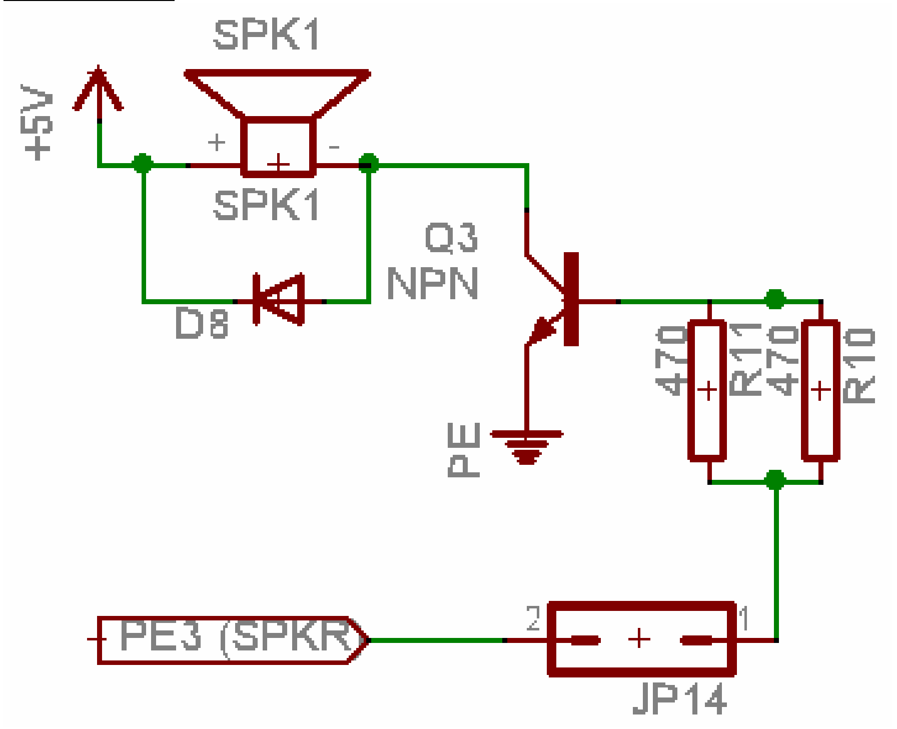
Conclusion:

If one of the LEDs does not light up when you think it should. Try using a piece of wire to connect the appropriate pin to the 5 volt output on the PORTB header. This will light the LED unless it's burnt out. If a switch isn't working like it's supposed to, use an ohm meter or continuity tester to test if the resistance changes significantly across the switch when the button is pushed. These two methods will help you during debugging to test the mechanical part of the system. If the mechanical system works, then the problem must lie in software.

Piezo Buzzer

Yet another way for your robot to give you feedback is via the Piezo Buzzer. While the rated frequency is 2048Hz ± 200Hz, this versatile little speaker is capable of emitting sound within the entire human hearing range. Now your robot can twerp and beep like R2D2, dance to it's own music, or sound a warning to keep people from stepping on it.

Schematic:



Circuit Description:

The speaker can be physically disconnected from IC1 by removing jumper JP14. When the jumper is in place, the piezo speaker is driven by toggling PE3 (Pin 5) at the desired frequency. The signal output by IC1 is too weak to directly drive the speaker effectively, so it is amplified by transistor Q3.

Note	Frequency (Hz)
E	1318.51
F	1396.91
G	1567.98
A	1760.00
B	1975.53
C	2093.00
D	2349.32
E	2637.02

Table 3 - Common Musical Tones with Frequencies

Example Code:

X:\MRMC\user_manual\code_examples\buz_xmpl.c
www.myrobot.com\minicomputer\code\examples\buz_xmpl.c

Conclusion:

The Piezo Buzzer gives your robot a pretty unique way to communicate with you. While flashing lights are easily missed and LCD contrasts decrease with battery voltage, beeps are easily heard and rarely missed. If the speaker does not sound when it should, always check the jumper first. A dirty jumping block is the most common hardware-based reason for a buzzer not sounding.

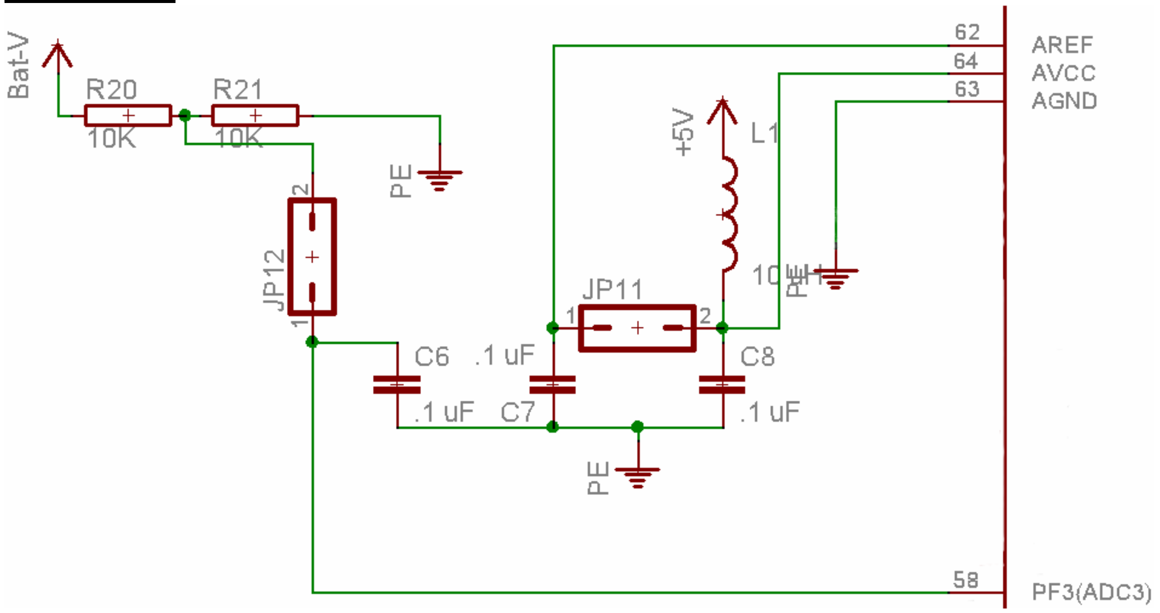
More Information:

Piezo Buzzer Data Sheet – www.myrobothq.com/datasheets/CEM1203.pdf

ADC & Low Battery Detection Circuitry

The ATMega128 (IC1) has eight built-in analog to digital conversion channels with up to 10-bit (1024 steps) accuracy. One of the these channels (ADC3) can be connected via JP12 to the low battery detection circuitry which allows your robot to monitor it's own power level and know when to return home to recharge.

Schematic:



Circuit Description:

R20 and R21 connect directly between the battery and ground when the Mini-Computer board is turned on. Even though the battery voltage is above 5 volts, the resistors create a voltage ladder that cuts the voltage in half. If the battery voltage is above 10 volts, then an ADC reading will just read 5 volts.

Jumper JP12 connects PF3 (Pin 58) to the two resistors. Capacitor C6 protects the line against electromagnetic radiation creating false readings. This simple battery voltage detection circuitry can be disconnected by simply removing the jumper.

When doing an ADC reading, the upper and lower reference voltages must be set. These are determined by AGND, AVCC, and AREF. AGND is tied to ground, which provides the lower reference. AVCC is tied to the regulated 5 volt power supply through inductor L1, and AREF can likewise be tied to 5 volts by jumper JP11. However, if you want a lower or higher reference voltage, the pin on JP11 that connects to AREF can be connected to any reference voltage you'd like. There is also an internal 2.56 reference voltage which can be configured and used via software. See the ATMega128 datasheet for more details.

Capacitors C6, C7, and C8, and the inductor L1 protect the ADC from electromagnetic transients which can easily create errors in your readings.

Example Code:

X:\MRMC\user_manual\code_examples\adc_xmpl.c
www.myrobot.com\minicomputer\code\examples\adc_xmpl.c

Conclusion:

An analog to digital converter provides a powerful way of interfacing your robot to the real, analog world. As you gain experience, you'll discover just how essential and powerfull they really are. Many sensors, like the Sharp GP2D120, provide only analog output. Many other sensors, like the SRF04 and SRF08 Ultrasonic Ranging Modules, come in both analog and digital form, but the analog sensor will cost a whole lot less. Thanks to the easy to use function libraries included with your board, you can quickly and easily instruct your robot on how to interface with these sensors.

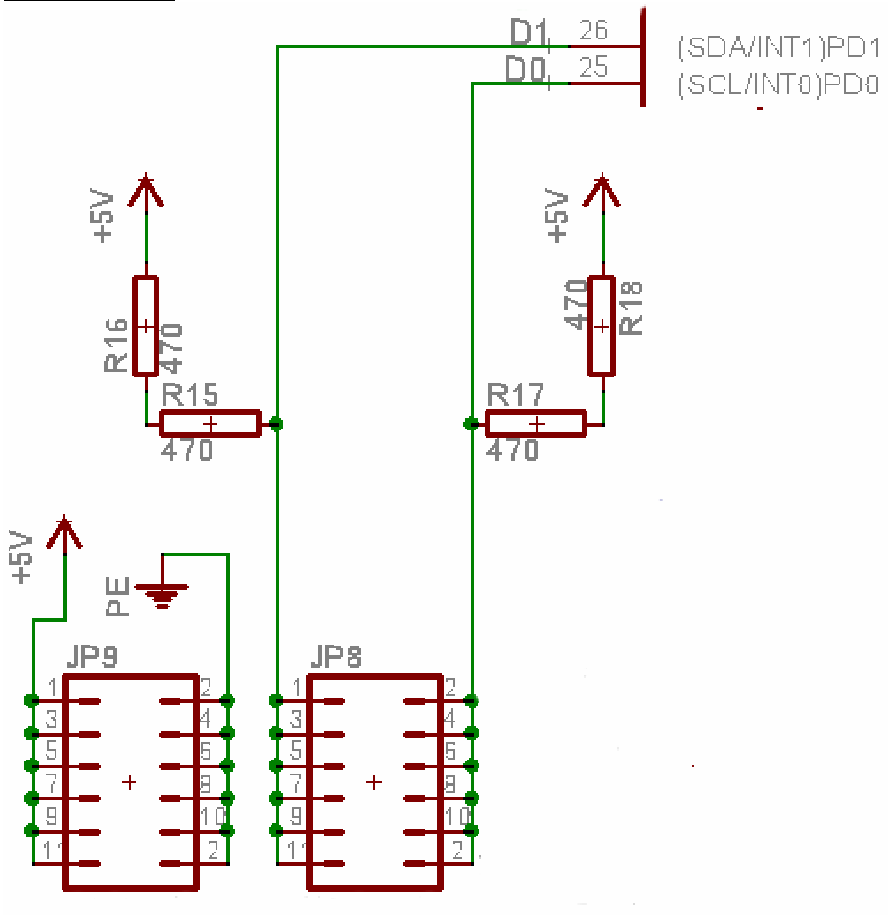
More Information:

ATMega128 Datasheet – www.myrobothq.com/datasheets/atmega128.pdf
Sharp GP2D120 Datasheet – www.myrobothq.com/datasheets/gp2d120.pdf
SRF04 Ultrasonic Sensor – www.myrobothq.com/datasheets/SRF04.pdf

I2C and Regulated Power Supply Port

I2C protocol was developed by Phillips and stands for Inter-Integrated Chip protocol. Due to branding rights, Atmel refers to their implementation as TWI or Two-Wire Interface. Regardless of what you call it, the I2C port allows fast communication between devices using only two wires. On your Mini-Computer board, 6 I2C ports and 6 regulated 5-volt outputs have been provided for connecting I2C capable devices. Many devices such as EEPROMs, sensors, and other micro-controller based devices communicate via the I2C protocol only.

Schematic:



Circuit Description:

The I2C port uses only two lines of the microcontroller: PD0 (Pin 25) for the SCL line, and PD1 (Pin 26) for the SDA line. When using these lines for I2C communication, they can not be used for other general I/O operations. These two lines are tied high by the pull-up resistors R15 – R18. These resistors are essential parts of I2C operation.

The header for the regulated output provides 5-volts for powering external circuitry. It is important to note here, that the 5-volt regulator on the Mini-

Computer board is only designed to output up to 500 mA (.5 Amps). Much more than this and it will start to get very hot. The regulator has built-in thermal protection, so if you try to draw too much power out of it, then it will shut itself down. You'll know when this is happening because your Mini-Computer board will shut itself off for several minutes until the regulator cools down. Repeated heating and cooling of the regulator can damage it.

Example Code:

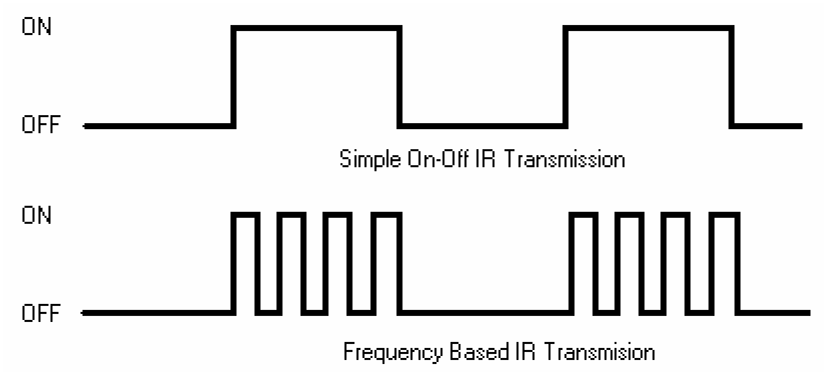
Conclusion:

Due to limitations of size and power the Mini-Computer does not include many of the common computer ports such as serial or parallel ports. However, I2C allows it to communicate with outside periferials very quickly, easily, and with as few wires as possible. Via the I2C port, your robot now has the power to delegate commands to external microcontroller based circuits, store data in I2C capable memory, or improve its senses via I2C capable sensors. Perhaps you could build a robot with parallel processing capabilities?

More Information:

Infra-Red Transceiver

When most people think of an Infrared Transmitter, they think of turning an LED on and off. On means 1 and off means 0. While this is the simplest method of implementing an infrared transmitter, it is prone to many errors such as interference from other transmitters, sunlight, and heat. The most widely used method combines the simple method with a high frequency signal. Instead of just turning the IR LED on and off, they turn it on very quickly at a certain frequency during the 'on' period. The result looks similar to the diagram below.

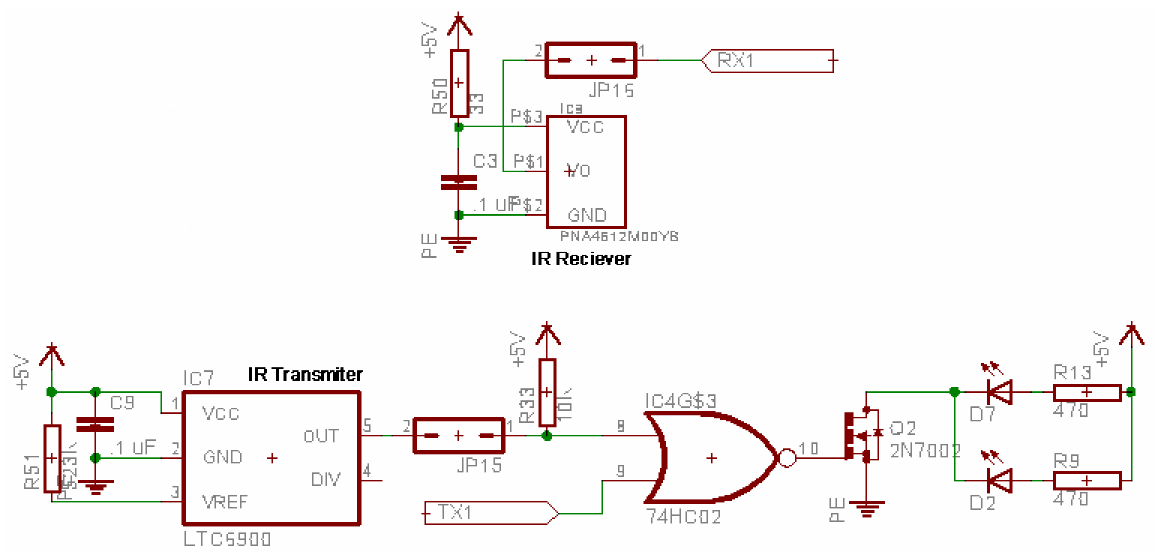


The advantage of using this method is that IR receivers can be designed to detect a small 'window' of frequencies, and thereby be much more efficient at only letting the information we want through while ignoring interfering information such as from sunlight or other transmitters operating at different frequencies.

The transmitter and receiver parts of the IR tranciever on your Mini-Computer board are actually two separate and independent circuits. The IR receiver is primarily composed of an IC which detects and amplifies all IR signals at a cetain frequency, and cuts off all IR signals at other frequencies. By cutting off other frequencies, the IC protects against interference such as sunlight, and allows multiple IR transmitters operating at different frequencies to be operating concurrently.

The IR transmitter works by mixing a high frequency signal and a low frequency signal. The low frequency signal is what we normally think of as our transmitter signal turning the LED on and off. The high frequency signal is added on top of it in order to produce a transmission signal like the one illustrated above.

<u>Schematic:</u>



Circuit Description:

The IR receiver is almost entirely composed of IC3. This integrated circuit detects and amplifies any IR signals around a frequency of 38.0 KHz.

Example Code:

X:\MRMC\user_manual\code_examples\ir_xmpl.c

Conclusion:

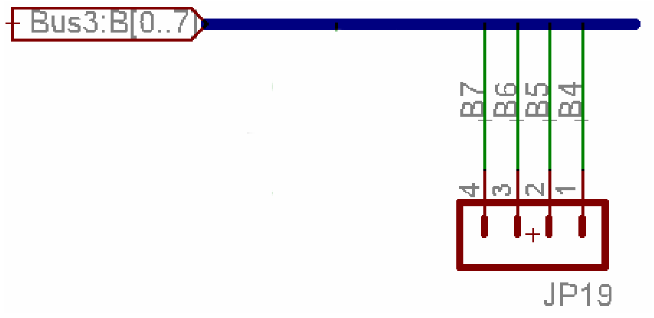
The IR transceiver on your Mini-Computer board allows you to communicate with virtually any and all IR capable devices. This includes computers, PDAs, printers, other robots, and remote controls. The example code included with your board shows you how to remotely control your robot via any standard remote control using NEC protocol. Perhaps you could program your robot to upload data periodically to your desktop or PDA? Or maybe you could write a laser tag program and compete your robot against your friends? Use your imagination, improve your skills, and have fun!

More Information:

PWM Output Port

The ATMega128 microcontroller (IC1) contains six Pulse Width Modulation (PWM) channels. Four of those channels are routed to the PWM Output Port on the Mini-Computer. This port allows an easy interface for motor controller boards that allow a PWM input signal, such as the MyRobot MDRV board, or for other circuitry that can make use of a PWM output signal such as Digital to Analog Conversion (DAC) or speech synthesis.

Schematic:



Circuit Description:

The upper four bits of PortB (PB4-PB7) on IC1 (ATMega128 Microcontroller) are connected to the PWM Output Port. Each PortB pin has an alternate use associated with the Timer/Counters and PWM Output control of IC1. For motor control speed control, you will most likely be using PB5 (OC1A) and PB6 (OC1B) as the PWM signal output. This output is driven by the Timer/Counter1 periferial of IC1. See the ATMega128 datasheet and the example code for more information in using the creation of the proper PWM signal.

Name	Pin	Alternate Use
PB4	14	OC0 (Output Compare and PWM Output for Timer/Counter0)
PB5	15	OC1A (Output Compare and PWM Output A for Timer/Counter1)
PB6	16	OC1B (Output Compare and PWM Output B for Timer/Counter1)
PB7	17	OC2/OC1C(1) (Output Compare and PWM Output for Timer/Counter2 or Output Compare and PWM Output C for Timer/Counter1)

Table 4 - Alternate Uses of PB4-7

Example Code:

X:\MRMC\user_manual\code_examples\mspd_xmpl.c
www.myrobot.com\minicomputer\code\examples\mspd_xmpl.c

Conclusion:

There are many advantages to using DC motors as opposed to Servos or steppers - especially in competitions. The easiest and most efficient way to control the speed and power delivered to a DC motor is with a PWM signal. The generation of this signal is not always very easy, but with the provided example code, you now have an easy to use and quickly modifiable PWM signal generator.

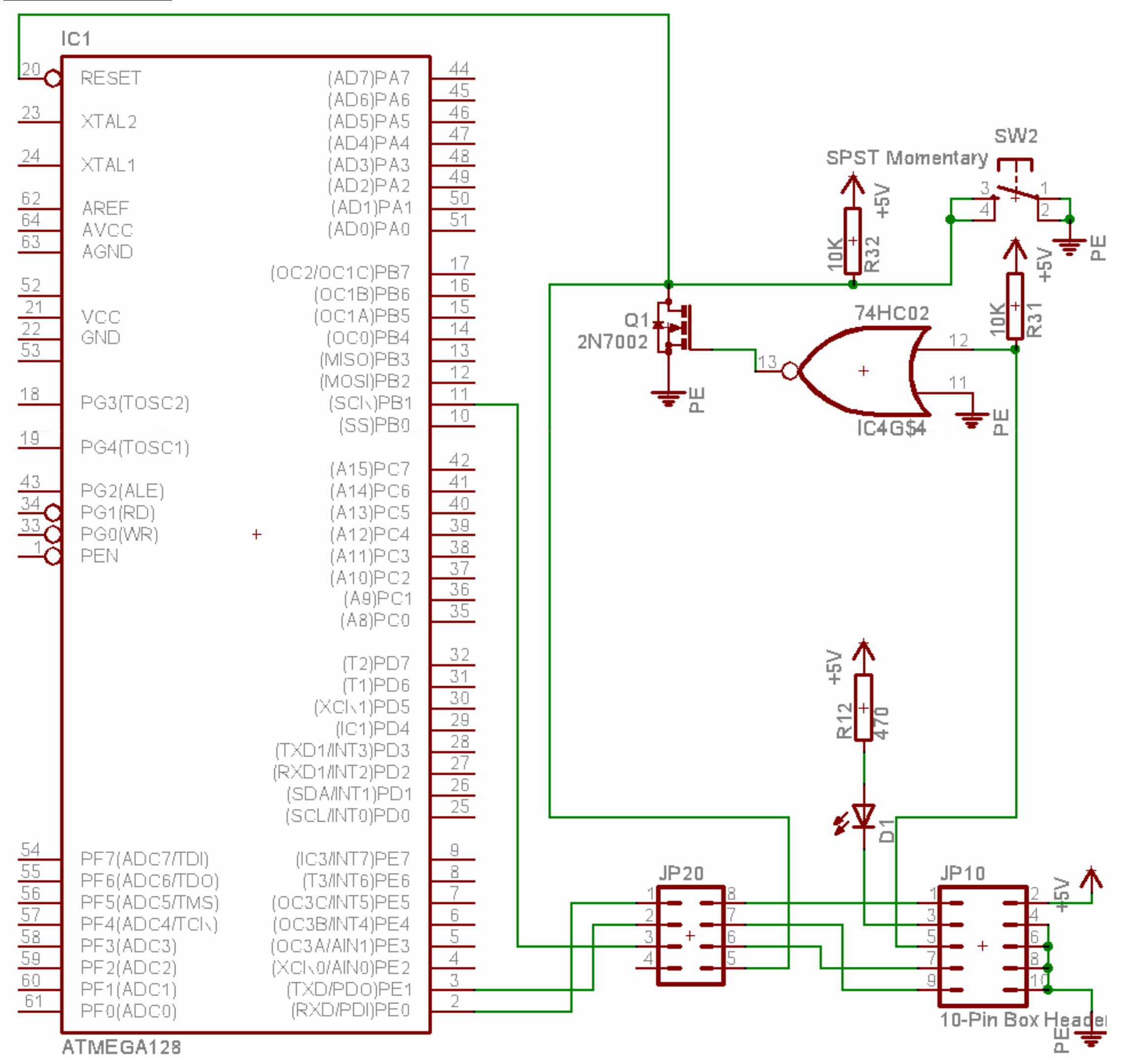
Unfortunately, there are not many inexpensive motor drivers designed for a PWM DC motor speed control. However, the MyRobot MDRV board is designed for unique power requirements for small and medium sized competition robots, and is designed to interface directly to your Mini-Computer board.

More Information:

ISP Programming

The MyRobot Mini-Computer board exploits the capability of the ATMega128 microcontrollers In-System Programming (ISP) protocol to reprogram itself without any special high-voltages or external circuitry. Simply use the dongle included with your board to attach your PC to the Mini-Computer via the 10-pin header and reprogramming is a snap.

Schematic:



Circuit Description:

During reprogramming, pin 5 of JP10, which is normally held high by pull-up resistor R31, is driven low to hold IC1 in reset mode. This switches IC1 into ISP mode. Additionally, LED D1 lights up to indicate when reprogramming is in progress. The lines, PE0 (pin 2), PE1 (pin 3), and PB1 (pin 11) are used for transmitting data between the PC and the Mini-Computer during programming.

Header JP20 is normally shorted together during normal operation and programming. It is provided to allow programming of other MyRobot products such as the MyRobot Odometer^{TM}. For more information, please see our website.

Example Code:

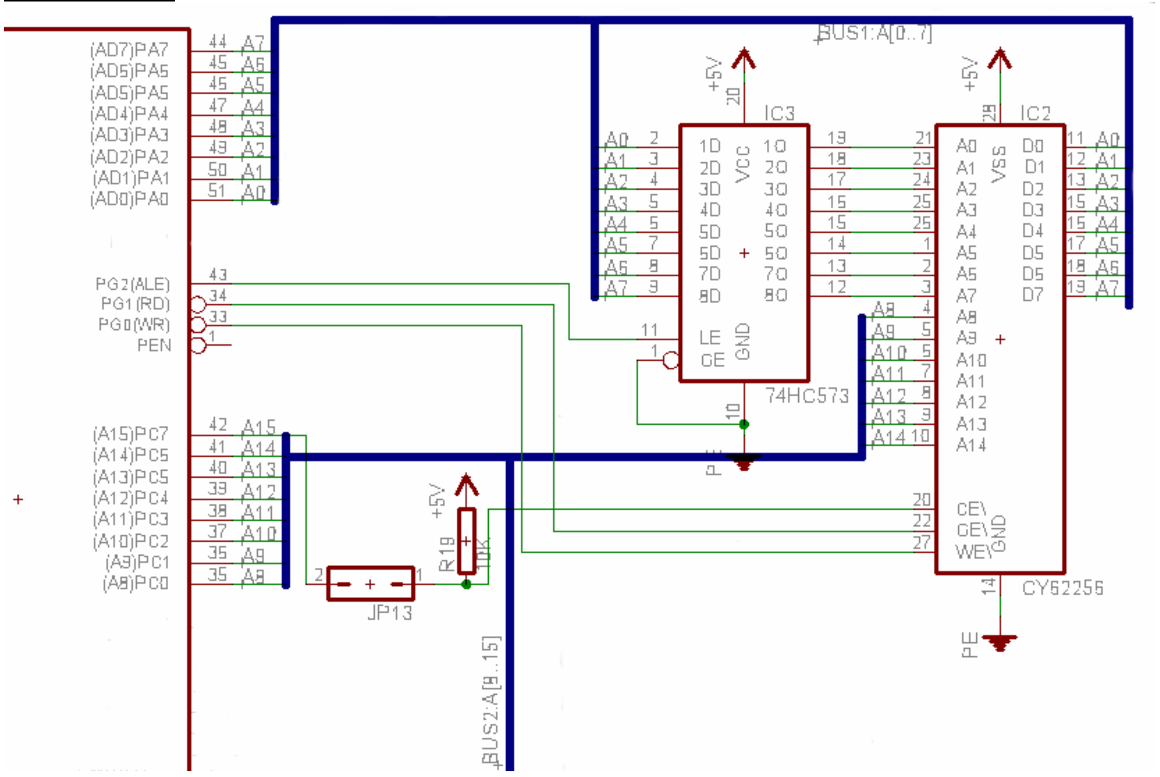
Conclusion:

More Information:

External SRAM

The AVR microcontroller on your Mini-Computer has 4 Kilo-Bytes (KB) of Static Random Access Memory (SRAM) built into it. However, an additional 32 KB of external SRAM is also provided which gives your Mini-Computer 36 KB of total data memory. While 4 KB is more than enough for most robotics projects, it is sometimes not enough for more advanced projects such as incorporating a Real-Time Operation System (RTOS) with several application running at once.

Schematic:



Circuit Description:

The ATMega128 microcontroller (IC1) on your mini-computer interfaces to the external SRAM (IC2) both directly and via the 74HC573 Octal D-Type Flip-Flop chip (IC3). In order to address 32 KB you need 2^15 I/O lines. The first eight of these lines (A0-A7) are associated with PortA, and the last seven (A8-A14) are associated with PortC. PC7 (A15) is also used as an enable/disable line for the external SRAM, and PG0 (WR), PG1 (RD), and PG2 (ALE) are used as control lines. Fortunatly for you, the complicated protocol for interfacing the SRAM is done internally by the chip, and all you need to do are call a few simple function calls (see the example code). However, the actual interface process is described below for those who wish to know how it works.

Since PortA is used in addressing, reading, and writing data, IC3 is used to drive the address while the data is read or written on PortA. This works by driving the lower 8-bits of the address on PortA (A0-A7), outputting the same lower 8-bits of the address on IC3 by toggling the ALE (PG2) line, and either outputting the data on PortA in the case of a write operation, or setting PortA as an input and reading the data in through PinA in the case of a read operation.

PC7 (PortC pin 7) is only used in addressing if you are interfacing with a 64 KB chip, and so is also labeled A15. On the Mini-Computer Board, it is not used in the addressing. However, PC7 is used as an enable line for the external SRAM chip. In order to interface IC2, jumper JP13 must be in place, and you must drive a low on PC7. When a high is driven on PC7 or the jumper is not in place, the external SRAM is disabled, and it's outputs will go into high-impedence mode.

Once the proper address is by driven by the outputs of IC3 and PC0-PC6 (A8-A14), and PortA is driving the data in the case of a write operation (PortA = Data, DDRA = $FF), or configured as an input (PinA = Data, DDRA = $00) for a read operation, the proper Read/Write patter should be present on the RD (PG1) and WR (PG2) lines, and PC7 (A15) should be toggled over two clock cycles. See the table below:

Line	Read Operation	Write Operation
IC3 Output	Lower 8-bits of address	
PC0-PC6 (A8-A14)	Upper 7-bits of address	
DDRA	$00	$FF
PortA	$00	Data
PinA	Data	$XX*
**PG0 (ALE)	0	0
PG1 (RD)	0	X*
PG2 (WR)	1	0
PC7	0	0

*X denotes that we don't care whether the state is low or high. It doesn't matter in this case.
**We assume the ALE line has already been toggled by taking from low to high back to low (L -> H -> L) in order to output the lower 8-bits of the address on IC3. This is done in both read and write operations.

<u>Example Code:</u>

X:\MRMC\user_manual\code_examples\sram_xmpl.c
www.myrobot.com\minicomputer\code\examples\sram_xmpl.c

<u>Conclusion:</u>

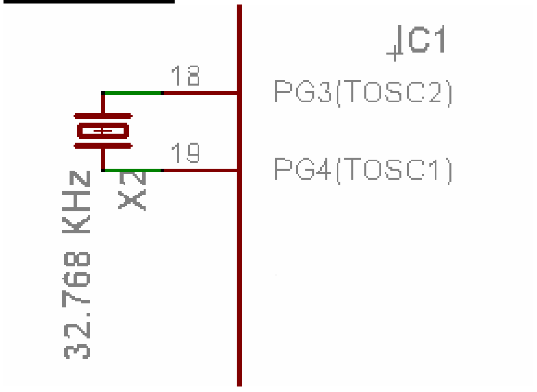
While the above Circuit Description may seem very complicated, the actual use of the external SRAM is very simple. Also, with the extra memory, the computing capabilities of your robot are raised much closer to that of a conventional computer. Eventually, every serious robot hobbyist and robot competitor will want to employ some form of Real-Time Operating System (RTOS). The biggest road block to implementing an RTOS in embedded systems is memory. MyRobot included the extra 32 KB of memory with this in mind, in hopes that you will use your Mini-Computer to its fullest capacity.

More Information:

32.678 Khz Real Time Clock

The 32.678 Khz crystal attached to IC1 is the same type of crystal used in watches.

Schematic:



Circuit Description:

Example Code:

Conclusion:

More Information:

General Features

Reset Switch

Polarized Power Connector

I/O Ports

On/Off Switch

Software – modelt.h Library Functions

Delay Functions

ms_spin()

Header Format:

void ms_spin(uint16_t _ms);

Include file:

modelt.h

Description:

The ms_spin() function is used for precise delay in the execution of code for a certain number of milliseconds. The argument '_ms' is a 16-bit unsigned integer which allows you to delay anywhere from 1 to 65536 milliseconds. Note that there are 1,000 milliseconds in 1 second.

Example:

```
uint8_t i=0;                  //Declare and initialize 8-bit unsigned integer i
while (1) {                   //Loop forever
        sprintf(t, "%d ", i); //Write the value of i to string t
        line1(t);             //Display string t on LCD
        i++;                  //Increment i
        ms_spin(5000);        //Wait 5 seconds before going back through the loop
}
```

us_spin()

Header Format:

void us_spin(uint16_t _us);

Include file:

modelt.h

Description:

The us_spin() function is used for precise delay in the execution of code for a certain number of microseconds. The argument '_us' is a 16-bit unsigned integer which allows you to delay anywhere from 1 to 65536 microseconds. Note that there are 1,000,000 microseconds in 1 second.

Example:

```
uint8_t i=0;                  //Declare and initialize 8-bit unsigned integer i
while (1) {                   //Loop forever
        sprintf(t, "%d ", i); //Write the value of i to string t
        line1(t);             //Display string t on LCD
        i++;                  //Increment i
```

```
                //Wait 65.5 milliseconds before looping again
                ms_spin(65);            //Wait 65   milliseconds
                us_spin(500);           //Wait 500 microseconds
        }
```

LCD Control Functions

lcd_4()

Header Format:

void lcd_4(uint8_t a);

Include file:

modelt.h

Description:

This function is used to send 4-bit control and character commands to the LCD. Every byte of data sent to the LCD needs to be split up in to two 4-bit nibbles before being sent. (one nibble is half a byte)

An 8-bit unsigned integer 'a' is given as an argument, however lcd_4() only shifts out the five least significant bits of this argument to IC5 (the 74HC164 shift register). The LSB of argument 'a' is the RS bit. This bit should be a 1 for sending character data to the LCD or a 0 for sending control commands. The next four bits should be a nibble of the control command or character data. Each nibble must be sent in the right order: high nibble first, low nibble second.

Example:

```
uint8_t lcd8_data = 0x41;           //'A' in ASCII
uint8_t rs = 1;                     //Set RS to indicate this is a character to display
uint8_t upper_bits,lower_bits;      //dummy variables to split 8 Bits of data
                                    //into 2 4-Bit nibbles

upper_bits = (0x0F & lcd8_data>>4)|rs<<4;    //Assign the upper nibble of lcd8_data
                                             //and the RS bit to upper_bits.
lower_bits = (0x0F & lcd8_data)|rs<<4;       //Assign the lower nibble of lcd8_data
                                             //and the RS bit to lower_bits.
lcd_4(upper_bits);
lcd_4(lower_bits);
```

lcd_8()

Header Format:

void lcd_8(uint8_t a, uint8_t rs);

Include file:

modelt.h

Description:

The lcd_8() function essentially just runs the lcd_4() function twice. It takes in two 8-bit unsigned integer arguments 'a' and 'rs'. 'a' should contain the 8-bit character or command to be sent to the LCD, while RS should equal 1 if ‘a’ is a control command or 0 if it is character data that should be written out on the display.

Example:
```
lcd_8(0x80,0);          //move cursor to 1st line
ms_spin(5);             //wait for LCD
```

line1()

Header Format:

void line1(char *t);

Include file:

modelt.h

Description:

line1() takes the argument ‘t’, which is a pointer to a string, and writes that string to the first line of the LCD. The string pointed to by ‘t’ should not exceed the number of characters that can be displayed on the line. For the LCD that is included with Mini-Computer board, this is 16 bytes (or characters).

Example:
```
uint8_t temp = 100;
char lcdstr[16];

sprintf(lcdstr,"temp = %d   ", temp);      //Write a formatted string to the lcdstr array
line1(lcdstr);                             //Display lcdstr on the LCD
```

line2()

Header Format:

void line2(char *t);

Include file:

modelt.h

Description:

line2() takes the argument ‘t’, which is a pointer to a string, and writes that string to the second line of the LCD. The string pointed to by ‘t’ should not exceed the number of characters that can be displayed on the line. For the LCD that is included with Mini-Computer board, this is 16 bytes (or 16 characters).

Example:
```
uint8_t temp = 100;
char lcdstr[16];

sprintf(lcdstr,"temp = %d   ", temp);      //Write a formatted string to the lcdstr array
line2(lcdstr);                             //Display lcdstr on the LCD
```

lcd_init()

Header Format:

void lcd_init(void);

Include file:

modelt.h

Description:

The lcd_init() function needs to be called only once during program initialization. It sends the necessary control codes to the LCD to enable it for use.

Example:

```
//Main
int main(void)
{

//Initialization
        reset();                        //Reset and all ports and peripherals
        lcd_init();                     //Initialize the LCD

//Main Execution Code
        while (1)                       //Loop forever.
        {};
}
```

pstrp()

Header Format:

int pstrp(char *t,uint8_t posnum);

Include file:

modelt.h

Description:

pstrp() is short for Put STRing at Point. This function takes two arguments, a pointer to a string ‘t’ and an 8-bit unsigned integer ‘posnum’ indicating where the string should start. For the LCD included with your Mini-Computer board, the integer ‘posnum’ should be between 1 and 32. 17 will start the string at the first character of the second line.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

Table 5 - Position of starting character corresponding to 'posnum' value

Example:

```
//This function uses the pstrp() function to center the string 'Centered'
//on line 1 of the LCD
strcpy(str, "Centered     "); //Copy the string 'Centered' to array 't'.
pstrp(str, 4);                //Insert the string stored in 't' four
                              //characters in from line 1
```

clr_lcd()

Header Format:

void clr_lcd(void);

Include file:

modelt.h

Description:

clr_lcd() clears both lines of the LCD display. It does not require any arguments and does not return any value.

Example:

```
strcpy(str, "This is on Line1");     //Write something to string ‘str’
line1(str);                          //Display string ‘str’ on LCD
ms_spin(5000);                       //Wait 5000 milliseconds (5 seconds)
clr_lcd();                           //Clear the LCD display
```

LED Control

led()

Header Format:

int8_t led(uint8_t ledn, uint16_t ms_on);

Include file:

modelt.h

Description:

led() is used for turning the user programmable LEDs on and off. The argument ‘ledn’ is an unsigned 8-bit integer representing the LED that should be turned on. Its value should be between 1 and 4. The argument ‘ms_on’ is likewise an unsigned 16-bit integer representing the number of milliseconds to turn the LED on for. The value of this argument should be between 0 and 65536. Note that there are 1000 milliseconds in 1 second. Also note, that this function makes use of the ms_spin() delay function internally. This means that the delay is done via loops, not interrupts, and thus no other software can be run while waiting for the led() function to expire.

Example:

```
strcpy(t, SW3MSG);            //Send a message to the LCD
line1(t);

led(1,2000);                  //Turn on LED1 for 2 seconds (2000 mS).

clr_lcd();                    //Clear the lcd.
```

Analog to Digital Conversion

getadc()

Header Format:

uint16_t getadc(uint8_t adcport);

Include file:

modelt.h

Description:

getadc() takes the argument ‘adcport’ which is an unsigned 8-bit integer whose value should range from 0 to 7 and corresponds to Analog to Digital port ADC0 to ADC7. The function then initializes the indicated ADC port, waits for the conversion to complete, and then returns with an unsigned 16-bit value representing the conversion results. The low reference voltage for this conversion are the 5-volt regulated power supply and ground. Note that the returned ADC value only has an accuracy of 10-bits, so the returned value will be 16-bits, but the value will never exceed 0x3F.

Example:
```
adcval = getadc(3);                         //Store ADC result from ADC3 in adcval
voltage = (adcval * ((2*4.98)/1024.0));     //Compute battery voltage from result
sprintf(lcdstr,"Bat: %10.2fv   ", voltage); //Write computed voltage to string lcdstr
line1(lcdstr);                              //Display string on line 1 of LCD
```

Two Wire Interface (TWI) Control

init_twi()

Header Format:

void init_twi(void);

Include file:

twi.h

Description:

This function initializes the Two Wire Interface (TWI), also known as Inter-Integrated Chip (I2C), peripheral on the Mini-Computer processor. This command only needs to be run once in the initialization section of the main() function. It requires no arguments and returns no value.

Example:
```
//Main
int main(void)
{
        //Initialization
        reset();                        //Reset and all ports and peripherals
        lcd_init();                     //Initialize the LCD
        init_twi();                     //Initialize the TWI

        while (1) {                     //Loop forever
        };
}
```

TWI_MT()

Header Format:

uint8_t TWI_MT(uint8_t addr, uint8_t length, uint8_t *var);

Include file:

twi.h

Description:

This function allows the Mini-Computer board to transmit data via the I2C bus to a slave device. Every I2C slave has a 7-bit address, allowing up to 127 devices to be addressed. However, the 8-bit unsigned integer argument ‘addr’ should contain the address shifted one bit to the left (i.e. an address of 0x71 would become 0xE2). This is because the last bit is reserved for the R/W bit during transmission. The next argument, ‘length’, is also an 8-bit unsigned integer which should indicate the number of bytes to be transmitted to the slave. The last argument, ‘var’, is a pointer to an 8-bit unsigned integer (or array of integers) which should contain the actual bytes of data to be transmitted. This function returns 0 if transmission has been successful, and an integer corresponding to the error code stored in the TWSR register of IC1 if the transmission is unsuccessful.

Example:

```
#define         SRF_08                  0xE0
#define         COMMAND_REG             0x00
#define         RANGE_INCH              0x50

int main(void)
{
        //Local Variables
        uint8_t temp[2];                //values to start a ranging session

        //Initialization
        init_twi();                     //Initialize the TWI

        temp[0] = COMMAND_REG;
        temp[1] = RANGE_INCH;
        if(TWI_MT(SRF_08, 2, temp))
                        ERROR(1);
}
```

TWI_MR()

Header Format:

uint8_t TWI_MR(uint8_t addr, uint8_t length, uint8_t *var);

Include file:

twi.h

Description:

This function allows the Mini-Computer board to receive data via the I2C bus from a slave device. Every I2C slave has a 7-bit address, allowing up to 127 devices to be addressed. However, the 8-bit unsigned integer argument, ‘addr’, should contain the address shifted one bit to the left (i.e. an address of 0x71 would become 0xE2). This is because the last bit is reserved for the R/W bit during reception. The next argument, ‘length’, is also an 8-bit unsigned integer. The length argument should indicating the number of bytes to be received from the slave. The last argument, ‘var’, is a pointer to an 8-bit unsigned integer (or array of integers) where the received bytes will be stored. This function returns 0 if reception has been successful, and an integer corresponding to the error code stored in the TWSR register of IC1 if the reception is unsuccessful.

Example:

```
#define        SRF_08                  0xE0
#define        RESULT_REG              0x02

int main(void)
{
       //Local Variables
       uint8_t temp[2];                //values to start a ranging session
       uint8_t srf_results[32];        //array that stores the range results

       //Initialization
       init_twi();                     //Initialize the TWI

       //Retrieve the SRF-08 ranging results
       temp[0] = RESULT_REG;
       if(TWI_MT(SRF_08, 1, temp))     //Tell the SRF-08 which register we want to read
              ERROR(2);
       if(TWI_MR(SRF_08, 4, srf_results))    //Read the first two results
              ERROR(3);
}
```

Other

reset()

Header Format:

void reset(void);

Include file:

modelt.h

Description:

The reset() function can be called to quickly and easily restore the processor to the way it was on initialization. It turns off all peripheral devices, disables all interrupts, and makes all I/O ports an input. This function is useful to call during initialization in order to ensure a known state on the ports and peripheral registers before accessing them. It can also be useful to call whenever a simulated software reset is desired without actually resetting the device.

Example:

```
int main(void)
{
//Local Variables

//Initialization
       reset();                        //Reset and all ports and peripherals
       lcd_init();                     //Initialize the LCD
       init_twi();                     //Initialize the TWI

//Enable global interrupts (Keep this instruction at the end of initialization).
       asm volatile ("sei");

//Main Execution Code

       while (1) {};
}
```

Appendix

A1 – Definitions

IR – Infrared
MSB – Most Significant Bit; The highest value bit (furthest to the left) in a byte
LSB – Least Significant Bit; The lowest value bit (furthest to the right) in a byte
Tranciever – Used to indicate the presence of both a receiver and transmitter in a circuit for bi-directional capability.