mega128.2

User Guide

TekBots™

Oregon State University

Version 1.0
By Adriaan Smit
OSU ECE Graduate Student

Index

Introduction

The mega128.2 board was developed within the TekBots group at the Electrical and Computer Engineering department at Oregon State University. The mega128.2 board is intended as a tool for learning assembly, C programming, and microcontroller architecture. Besides a tool for learning, it also provides a stable platform for the development of other projects requiring a microcontroller.

1.1 Features

- **ATmega128 AVR 8-Bit RISC Microcontroller**
- **32Kbyte External RAM**
- **(ISP) In System Programming via PC Parallel port**
- **RS-232 Port**
- **IR Transmitter and Receiver**
- **8 Push Buttons**
- **8 LEDs for General Use**
- **All AVR I/O Ports Easily Accessible through Pin Header Connectors**
- **AVR External Memory Expansion Easily Accessible through Pin Header Connectors**
- **Power Supply for 7 – 15V AC/DC**

Getting Started

Each component in the microcontroller kit is functionally tested before it is packaged. The mega128.2 board is packaged with the test program still in program memory. This test program is explained in more detail in section 2.3.3

2.1 Kit Contents

Each AVR Kit is shipped with the following components packaged in static sensitive bags.

- mega128.2 Board
- 2x16 character LCD
- DB25 Programming Dongle (mega128-isp.1)
- 10 Pin Ribbon Cable

2.2 System Requirements

Although development can be done on older versions of hardware and software, the following is suggested as minimum requirements.

- 486 processor
- 16MB RAM

- 12MB free hard disc space (Software Tools)
- Windows 95 or higher
- Parallel Port (LPT)
- 7-15V AC/DC Power supply, 200mA

2.3 Quick Start

2.3.1 Power

The mega128.2 board uses either an 8.7V battery pack (Not included) or a 7-15V AC/DC power supply (Not included) as a power source. LED L10 indicates when power is applied to the board and the switch, S10, is in the on position. When both the batteries and the wall wart are connected simultaneously, diode D2 isolates the batteries form the wall wart power source. The 7-15V AC/DC Power Supply can be either with a negative or positive center connector. A full bridge rectifier ensures the right polarities and DC conversion. The battery pack however is direction sensitive with GND on pin one and V+ on pin two as shown in Figure below. For details on supplying power to daughter boards from the mega128.2 board, refer to section 3.12 of this document.

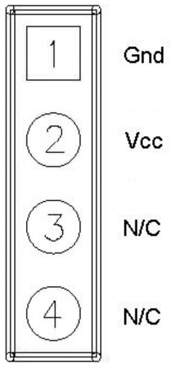


Figure 2.1: Battery Connector

2.3.2 Default Jumper Settings

The board is shipped with Jumpers 11, 10, 12, 5, 7, and 9 bridged as the default setting. This enables IR TXD, IR RXD, LCD, Ext Mem, UART0 TXD, and UART0 RXD. More detailed description on the jumper functions are given in section 3.8.

2.3.3 Power On Test

As mentioned, each mega128.2 board is delivered with the test program still in program memory. On power up, this program will execute. In order, the test program tests the I/O lines, LCD Display, Button inputs and LEDs, UART1 (IR transmit and IR receive), UART0 (RS232 transmit and receive), and the External SRAM. Following is a brief explanation of how to interpret the test output with the default jumper settings in place.

- **I/O Lines**

To test the I/O Ports, all the lines on every port are pulled high, stepping through from LSB to MSB. This is done twice. LEDs 1 through 8 are connected to port B. If nothing else is connected to any of the other ports, this part of the test will only be visible on L1 through L8. Note that by default the jumper to enable the IR transmitter on Port D.3 is in place so the IR transmit LED, L12, will come on whenever the line is pulled low. (IR transmit is active low.)

- **LCD Display**

After stepping through all the bits on the I/O ports twice, the LCD is tested by simply writing the text "***-LCD_ Test-***" on both the top and bottom rows of the LCD. After the LCD test, the remaining tests will also make use of the LCD display for instructional output.

- **Push Buttons**

For every button that is pushed, the corresponding LED (L1 through L8) is switched on. The LCD displays the text, "Test pushbuttons". The Push Buttons are also active low. Remember that Jumper 10 and 11 are still enabling the IR. This means that both the push buttons and the IR lines are connected to port D. Pushing button S4 will thus also cause the IR to transmit. The IR receiver is close enough to the transmitter that it actually receives at this point, causing PD2 to go low. This appears as a button being pushed and L3 goes on in addition to L4. To prevent this, disable IR receive by removing jumper, J10. To go to the next test, push buttons S1 and S8 need to be pushed simultaneously. The LCD will briefly display the text "Switch Test Done".

- **IR**

While testing IR RXD and TXD the LCD displays the text "IR Tx/Rx Test" and "Tx & L1 blink". During the IR test, L12 (the IR TXD indicator) and L1 should blink in time with each other.

- **External RAM**

The last test is the RAM test. The LCD will display the text "Ram Test_" while the LED's on port D (L1 through L8) are repeatedly being switched on from LSB to MSB. It will appear that the LSBs are always on while the MSBs are flashing quickly. If there are LEDs connected to port A and port G, the activity on those ports will also be visible.

After all tests have been completed, L3 is on and the LCD displays the text, "RAM Test Success". In newer versions, "TekBots Rock!" The code that tests UART0/RS232 is currently commented out. Also to test UART0, a loop back connector must be in place on the DB9 connector.

2.3.4 ISP Connection

At the ends of the 10 Pin ribbon cable, there are two female headers. Both headers are notched to prevent them from being connected backwards. Connect the one end of the ribbon cable to J22 on the mega128.2 board, and the other end to J1 on the mega128-isp.1 dongle. Note that if J1 on the dongle board does not have a notched shroud, the notch is clearly indicated on the silkscreen.
Lastly connect the mega128-isp.1 dongle to the parallel port of the computer. In most cases this would be LPT1.

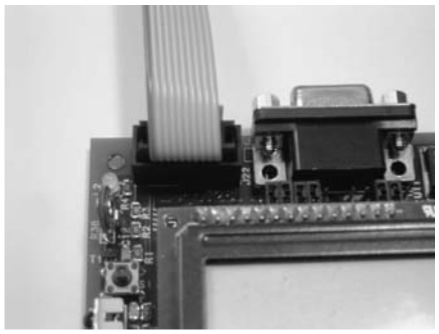


Figure 2.2: ISP Connector J22

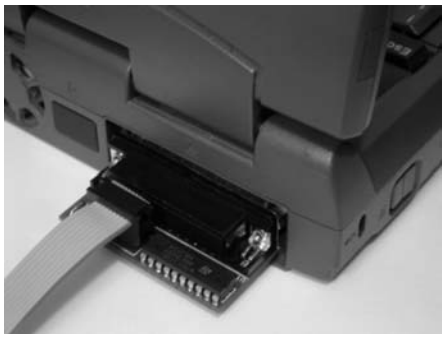


Figure 2.3: mega128-isp.1 Dongle

2.3.5 Programming ATmega128

AVR Studio is a software development package created by Atmel. AVR Studio 4.0 does not support parallel In System Programming. Packages that do support parallel ISP and that have been tested with the mega128.2 board are PonyProg2000 and CodeVision 1.23.8c
CodeVision can be used to write and build a program, and to program the AVR on the mega128.2 board. PonyProg2000 is only used for programming the AVR. Both programs are freely available for download at the following URL's:
PonyProg2000 - http://www.lancos.com/prog.html
CodeVision 1.23.8c - http://www.hpinfotech.ro/html/download.htm
A detailed explanation on Using PonyProg2000 can be found in: "AVR Starters Guide, by David Zier", section 3.2 and 3.3. For a detailed explanation on using CodeVision refer to Section 5 of this document.

Hardware Description

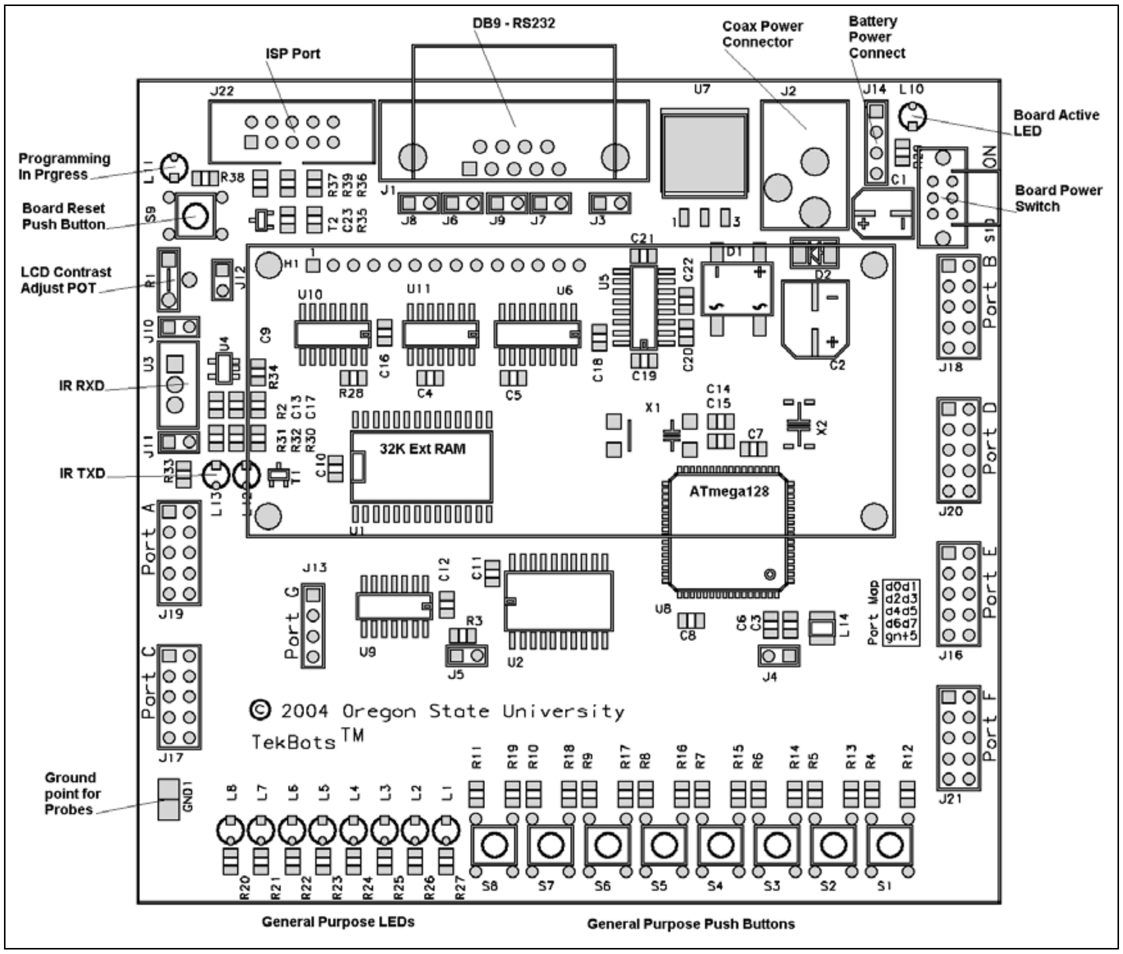


Figure 3.1: mega128.2 Board Layout

3.1 General Purpose LEDs

There are 8 LEDs on the mega128.2 board, L1 through L8, that are connected to port B and are driven directly by the AVR. These are high efficiency LEDs that only take 2mA to drive to full brightness. Thus always having them connected to port B does not load the port significantly.

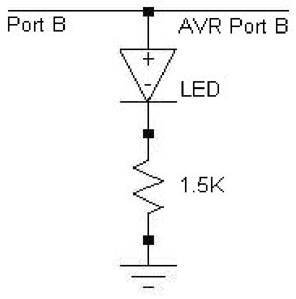


Figure 3.2: LED Active High

3.2 General Purpose Switches

There are 8 PushButton Switches on the mega128.2 board. These switches, S1 through S8, are connected to port D. An external 10K resistor is used to pull the port pin high. When the PushButton is pressed, the port pin is pulled low through a 470Ω resistor. This was done to prevent directly grounding a pin that may be configured to force a logic high.

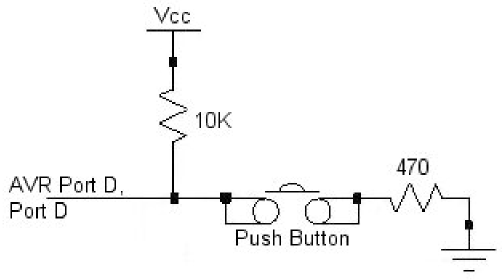


Figure 3.3: Read Button Active Low

3.3 Port Connectors

All the AVR ports are accessible through female headers. Female connectors were chosen so that it is both cheap and easy for the user to make custom connections to the mega128.2 board. The header pin out is identical for ports A through F. The legend on the lower right side of the mega128.2 board serves as a quick reference port map. Port G is not consistent with the other ports. It is mapped to a 4pin header and consists of the external memory control lines. The figures below show the mapping to the port headers. All ports are aligned on 0.1" centers, allowing for protoboards to be mounted directly on top of the board. The battery connector is the only connector that is not on the 0.1" grid.

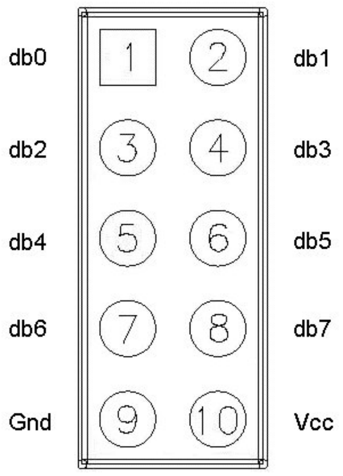


Figure 3.4: Port A through Port F

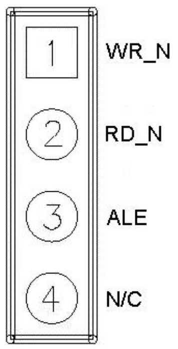


Figure 3.5: Port G

3.4 RS-232 Interface

The mega128.2 board has one DB9 connector, J1, for serial communication. There are 2 UART's on the ATmega128 chip that can be used for serial communication. To select between the two, jumpers J6, J8, J7 and J9 are used. J7 and J9 enable UART0 while J6 and J8 enable UART1. UART1 is connected through the RS232 level converter but is not connected to J1, the DB9. Instead it is connected to jumper connecter J3. Inserting a jumper on J3 allows for easy loop back testing on UART1. The figure below shows the diagram of the RS232 level conversion circuit.

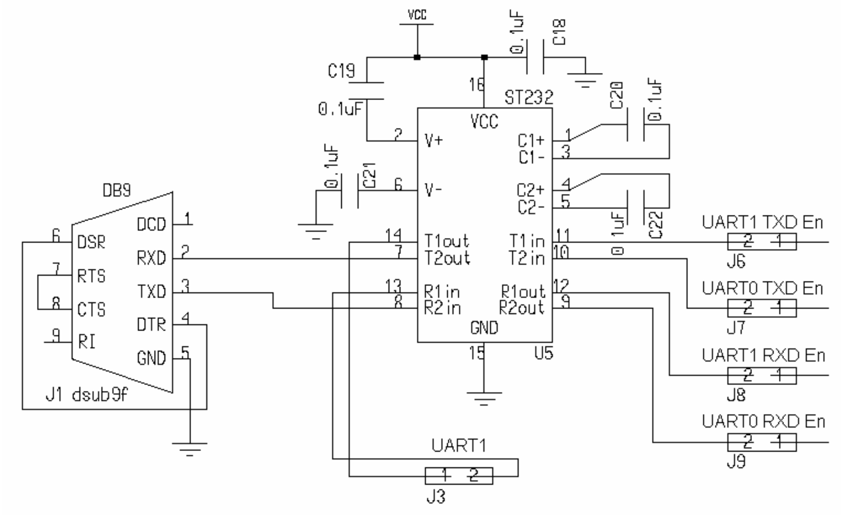


Figure 3.6: RS-232 Interface

3.5 IR Transmitter / Receiver

The IR Transmitter and Receiver is set up to utilize UART1. UART1 is connected to the IR through jumpers J10 and J11. The IR transmission is modulated at 38Khz by ANDing the signal with a local oscillator. Changing the value of resistor R34 changes the frequency of this oscillator. The IR LED is "on" in the mark state and "off" in the space state. LED L12 allows the user to see when the IR LED is being driven. . The figures below show the circuitry of the IR Transmitter and Receiver

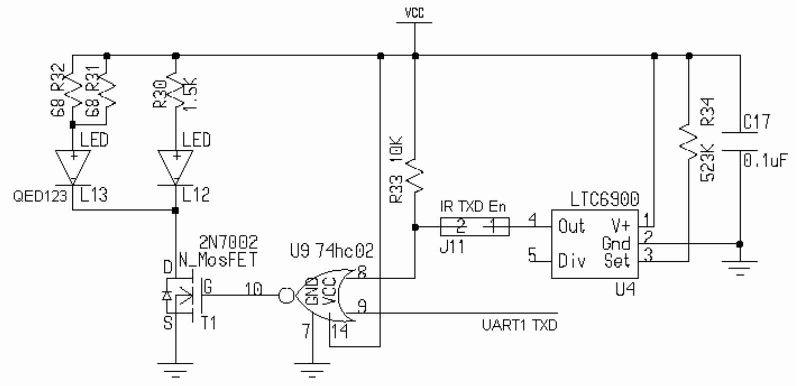


Figure 3.7: IR Transmitter

The PNA4612 has an internal 38KHz filter to prevent false triggering from other light sources. With the IR transmitter on ground the pair is able to communicate easily over 5 meters when properly aimed.

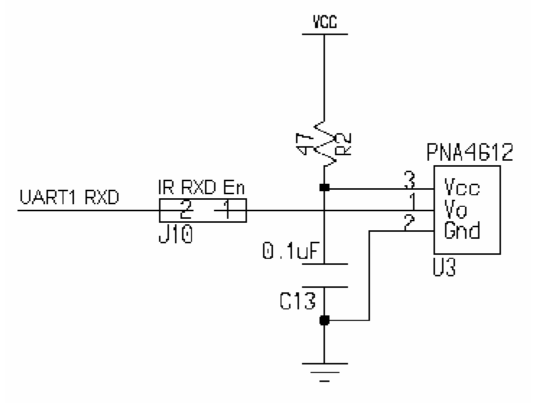


Figure 3.8: IR Receiver

3.6 Parallel ISP

The In System Programming is done via a parallel port. The interface is the same as the Kanda STK200/300 interface. An analog MUX is used to conserve the pins used on the ATmega128 chip. When programming the board, L11 is illuminated and the analog Multiplexer connects the AVR pins to the ISP header. When not programming, the analog MUX allows the AVR's I/O pins to revert to normal usage. The figure below shows the circuit diagram.

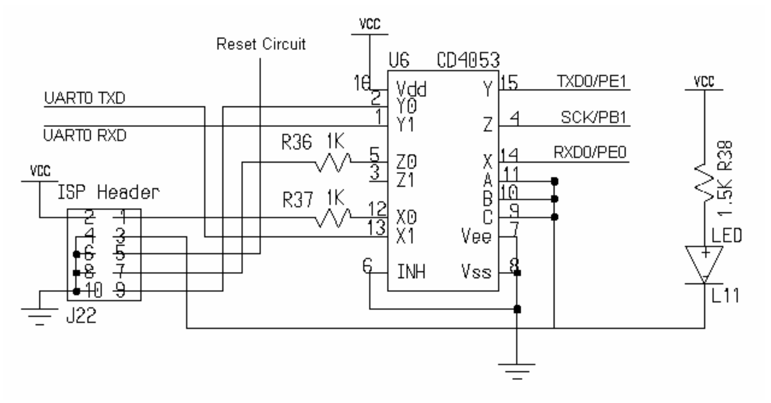


Figure 3.9: ISP Circuit

3.7 LCD

The LCD is mounted on top of the mega128.2 board and interfaces to it through a 14 pin header connection. The LCD uses the SPI port together with bit 3 of port F. The LCD works independent of the microcontroller's clock speed to conserve I/O pins. The figure below shows the circuit diagram.

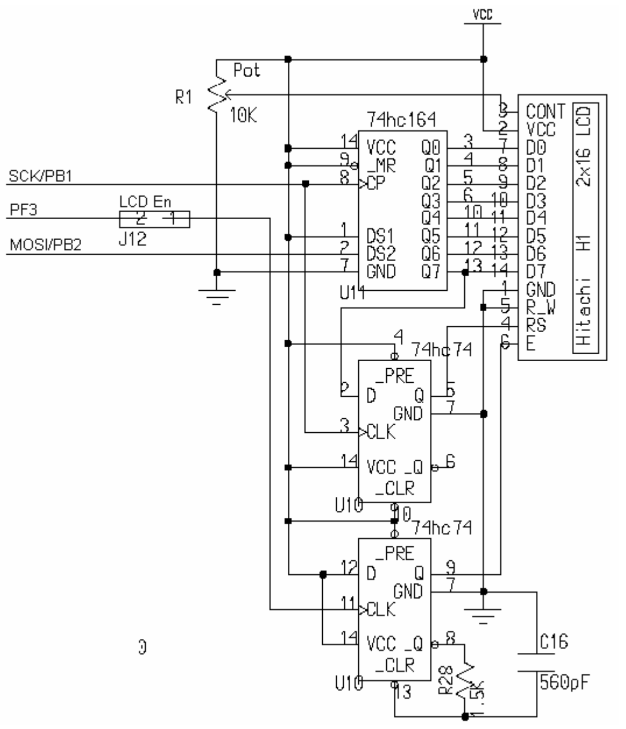


Figure 3.10: LCD Control Circuit

3.8 Jumper Settings

There are several jumpers that enable various functions on the mega128.2 board. The default jumper settings are listed in Section 2.3.2 of this document. One of the changes made between the mega128.1 and mega128.2 version is the placement location of the jumpers. On the mega128.2 board the jumpers are located in the same area as the involved circuitry for each particular jumper. The figure below shows the placement of the jumpers.

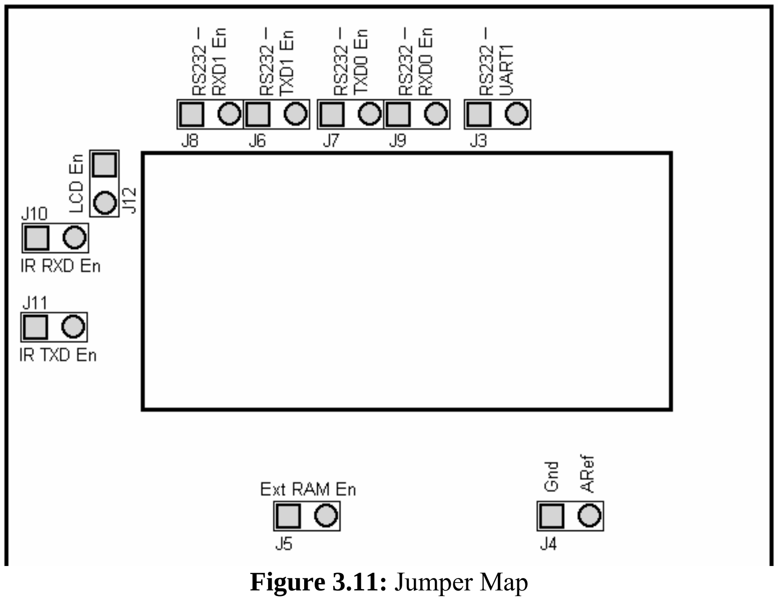


Figure 3.11: Jumper Map

Jumpers J8 and J10 are both enabling UART1 RXD. Also, jumpers J6 and J11 are both enabling UART1 TXD. J11 and J10 enable UART1 to transmit and receive over the IR port. J6 and J8 allow for the UART1 transmit and receive lines to be connected to J3. Thus UART1 does not connect to the DB9, J1.
J4 is an additional connector that was added so that an external voltage reference could be supplied for the analog to digital converter voltage reference.

3.9 Indicator LEDs

There are three LEDs on the mega128.2 board that are dedicated to indicate that specific functions are active. L10 is a greed LED and indicates that there is power connected to the board and that switch S10 is in the "on" position. Yellow LED L11 indicates that the In System Programming port is active. L12 is a red LED and indicates that the IR is transmitting.

Reference #	Color	Function
L10	Green	Board powered
L11	Yellow	ISP active
L12	Red	IR TXD active

Table 3.1: Indicator LEDs

3.10 Reset PushButton

Pushing PushButton S9 resets the ATmega128 microcontroller. This reset is OR'ed with the In System Programming chip reset. The ATmega128 reset is active low. The figure below shows the reset circuitry.

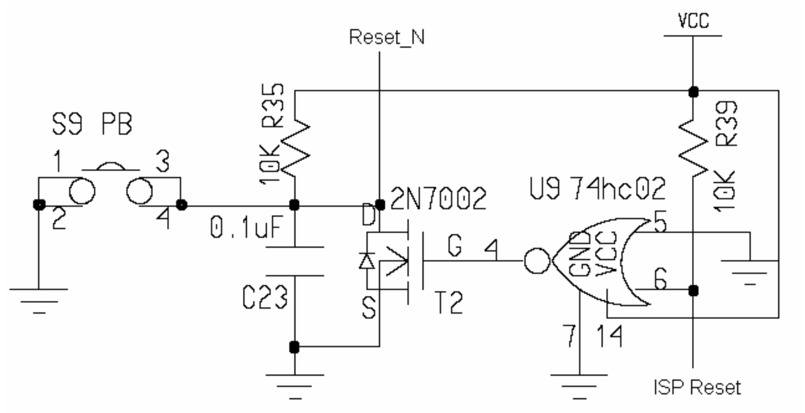


Figure 3.12: ATmega128 Reset Circuit

3.11 Power Supply

J2 is a coax connector that is used for connecting a power supply to the mega128.2 board. J14 is for connecting a battery pack to the mega128.2 board. The pinout for J14 is shown in Figure 2.1. The voltage regulator that is used on the board is a low dropout regulator.

The regulator is internally protected against reversed polarity and momentary input spikes. The regulator can source up to 1A. The figure below shows the power supply circuit.

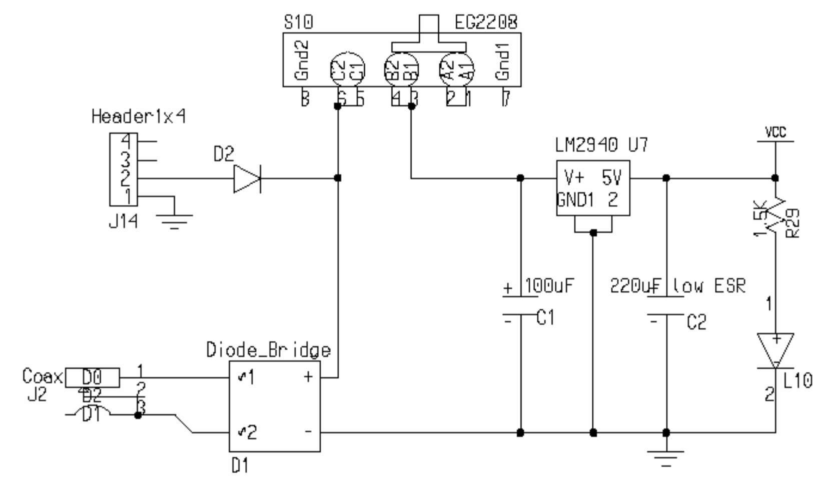


Figure 3.13: Power Supply

3.12 Expansion Possibilities

- **Memory Mapping**
  Ports A and C give the user access to the data and address lines. Port G gives access to WR_N, RD_N, and ALE lines. This provides the freedom to map additional external circuitry to memory. Memory block 0x0460 through 0x7fff is unallocated. External SRAM resides in the block 0x8000 – 0xFFFF.
- **Power Consumption**
  When powering external circuitry from the mega128.1 board the limiting factor is the heat dissipation of the voltage regulator. The TO-263 package with one square inch of copper area (allocated on layer 1) will safely dissipate about 2.75W with an ambient temperature of 25 C°. See Figure 4 of the LM2940 data sheet. To calculate if external circuitry can be powered from the mega128.1 board, do the following:
  $2.75 \leq (Vin - 5) * (Iavr + I\ daughter)$
  Where: Vin is the input voltage to the board
  Iavr is the current consumption of the AVR board
  Idaughter is the current consumption of the daughter board
  5 is the regulator output voltage
  To increase the amount of current that the external circuitry is allowed to draw, the IR portion of the mega128.1 board can be disabled. Refer to "Jumper Settings" in section 3.8 of this document.
- **SPI Port**
  The Serial Peripheral Interface allows for high-speed synchronous data transfer between the Atmeag128 and peripheral devises or other AVR devises.
  The table below shows the SPI mapped to port B.

SPI Function	**Port B Pin #**
SS_N	PB0
SCK	PB1
MOSI	PB2
MISO	PB3

Table 3.2: SPI Port Map

- **Analog Interface (A/D)**
  The ATmega128 is also equipped with an Analog to Digital Converter. The ADC is connected to an 8 channel Analog Multiplexer which allows 8 single-ended voltage inputs constructed form the pins of port F. Each channel has a resolution of 10 bits.
- **ProtoBoard Connections**
  All port connectors are aligned on 0.1" centers, allowing for protoboards to be mounted directly on top. The ports are all female connectors. This makes it both easy and cheap to build custom connectors to plug into the ports.

mega128.1 Specifications

4.1 Dimensions

Board and LCD mounting holes have an inner diameter of 0.116". All other holes have an inner diameter of 0.046" and are centered on a 0.1" grid. Dimensions are given in inches.

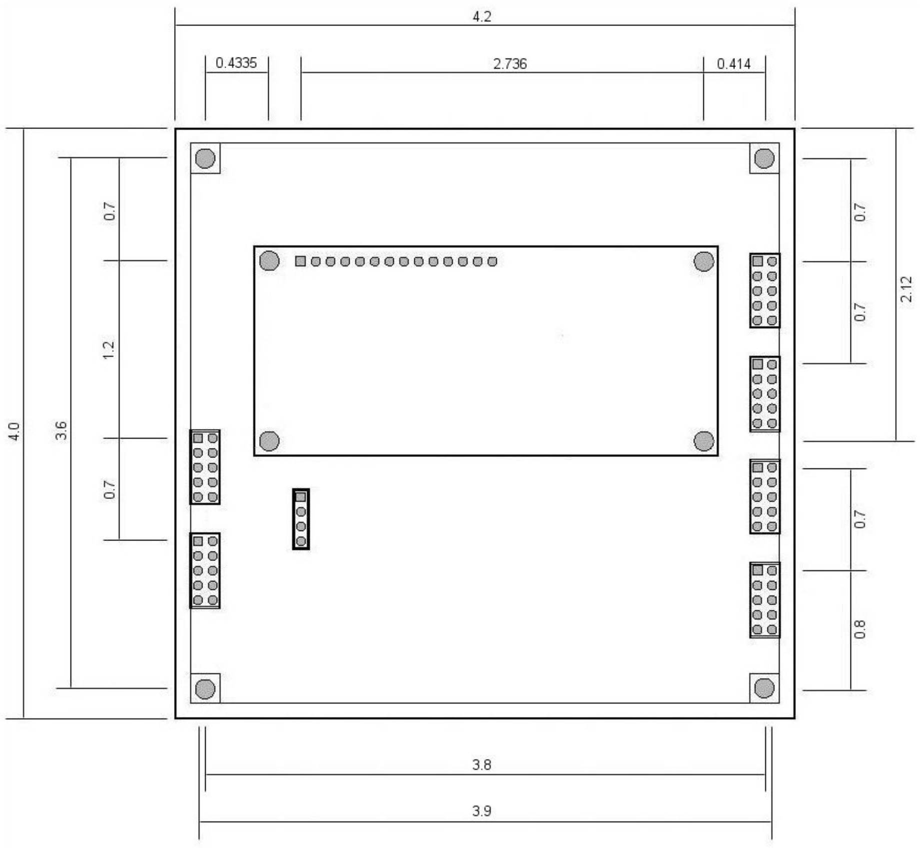


Figure 4.1: Board Dimensions & Placement

4.2 Voltage Input

The voltage regulator used on the mega128.2 board is a 5V, 1A Low Dropout Voltage Regulator. This allows for voltage input ranges from 7V up to 15V.

4.3 Current Consumption

40mA with IR disabled
140mA with IR LED at near 100% duty cycle

4.4 Memory Space

Internal - 4K Bytes
External - 32K Bytes

4.5 E^2 PROM

4K Bytes

4.6 Operation Temperature

0ºC to 70ºC

4.7 Board Material

The mega128.2 board is a four-layer board. Material used is FR4.

CodeVision 1.23.8c

This section gives a detailed explanation on using CodeVision 1.23.8c. This explanation is aimed at the mega128.2 board and might not apply to other AVR boards.

5.1 Creating a new Project

After opening the CodeVision Integrated Development Environment (IDE), select from the *File* menu option, *New*. In the Create New File popup window, select *Project* and click *OK*. You now have the option to use the CodeWizardAVR or to manually set up the project. For this discussion we will click on *No* in order to set the project up manually.
The next window will allow you to specify the project name and folder path. After saving the project, the Project Configuration window will pop up. This window has three tabs, shown in the figure below. Under the Files tab, there is the option to add and remove files to and from the project.

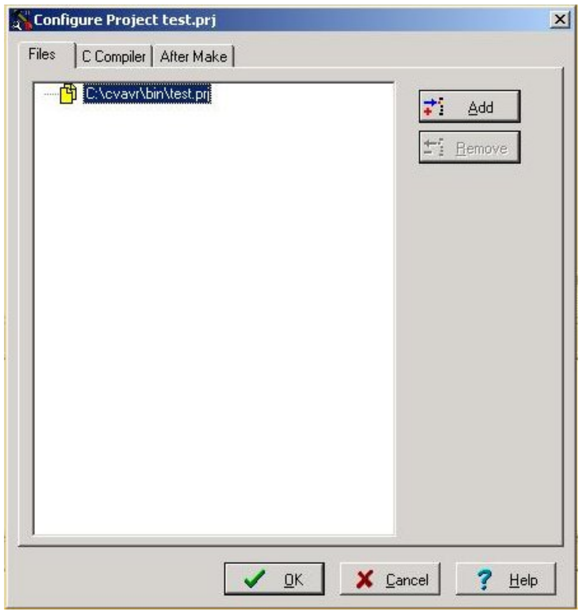
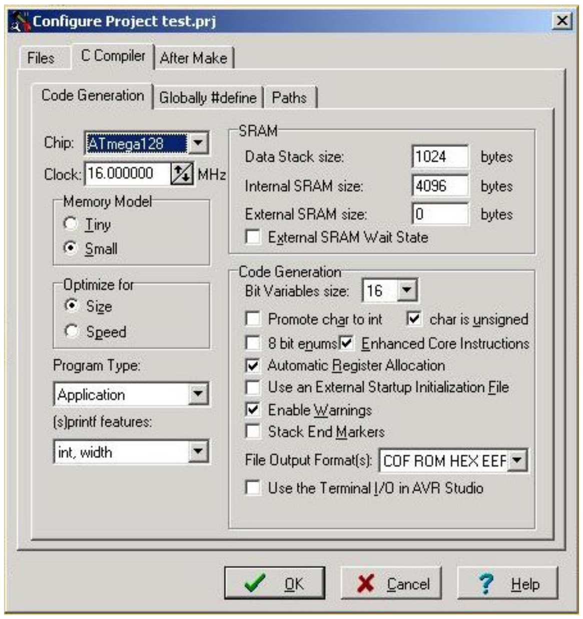


Figure 5.1: Adding Source Files to Project

The next tab is the C Compiler. This page is for AVR specific settings. The figure below shows the settings for the mega128.2 board.



Configure Project test.prj

Files | C Compiler | After Make

Code Generation | Globally #define | Paths

Chip: ATmega128

Clock: 16.000000 MHz

Memory Model: ( ) Tiny [✓ Small]

Optimize for: [✓ Size] ( ) Speed

Program Type: Application

(s)printf features: int, width

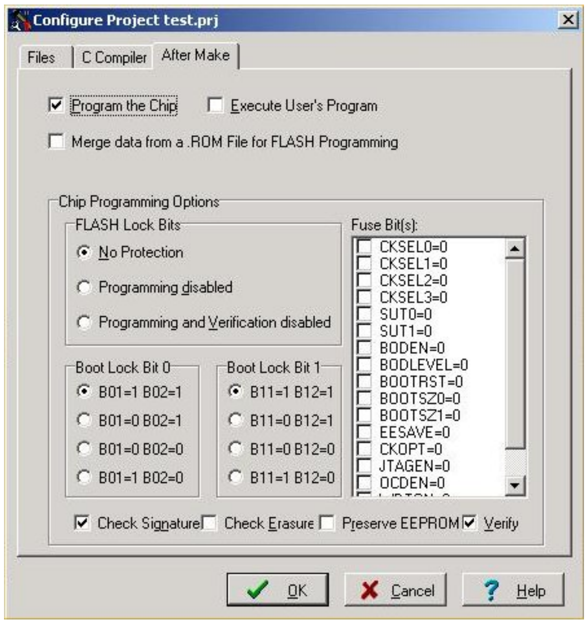
SRAM
- Data Stack size: 1024 bytes
- Internal SRAM size: 4096 bytes
- External SRAM size: 0 bytes
- [ ] External SRAM Wait State

Code Generation
- Bit Variables size: 16
- [ ] Promote char to int
- [✓] char is unsigned
- [ ] 8 bit enums
- [✓] Enhanced Core Instructions
- [✓] Automatic Register Allocation
- [ ] Use an External Startup Initialization File
- [✓] Enable Warnings
- [ ] Stack End Markers
- File Output Format(s): COF ROM HEX EEF
- [ ] Use the Terminal I/O in AVR Studio

OK | Cancel | Help

Figure 5.2: mega128.1Spesific Compiler Settings

Lastly, the third tab is for After Make instructions. To program the chip automatically after *Make*, check the "Program the Chip" option. Do not change the fuse bits unless you know what you are doing. They can put the AVR in an unusable state that cannot be recovered from. The figure below shows a proper setup of the third tab page.



Configure Project test.prj

Files | C Compiler | After Make

- [✓] Program the Chip
- [ ] Execute User's Program
- [ ] Merge data from a .ROM File for FLASH Programming

Chip Programming Options

FLASH Lock Bits: [✓ No Protection] ( ) Programming disabled ( ) Programming and Verification disabled

Boot Lock Bit 0: [✓ B01=1 B02=1] ( ) B01=0 B02=1 ( ) B01=0 B02=0 ( ) B01=1 B02=0

Boot Lock Bit 1: [✓ B11=1 B12=1] ( ) B11=0 B12=1 ( ) B11=0 B12=0 ( ) B11=1 B12=0

Fuse Bit(s):
- [ ] CKSEL0=0
- [ ] CKSEL1=0
- [ ] CKSEL2=0
- [ ] CKSEL3=0
- [ ] SUT0=0
- [ ] SUT1=0
- [ ] BODEN=0
- [ ] BODLEVEL=0
- [ ] BOOTRST=0
- [ ] BOOTSZ0=0
- [ ] BOOTSZ1=0
- [ ] EESAVE=0
- [ ] CKOPT=0
- [ ] JTAGEN=0
- [ ] OCDEN=0

- [✓] Check Signature
- [ ] Check Erasure
- [ ] Preserve EEPROM
- [✓] Verify

OK | Cancel | Help

Figure 5.3: Settings for After Make

After clicking *OK*, CodeVision should open up the source files that were added to the project. If no files were added a new source file should be created. To do this, click on *New* under the *File* menu option. In the popup window select *Source* and click *OK*. The new, empty source file should now be added under the project navigator. The figure below shows a project called "test" with a new file created, called "untitled.c".

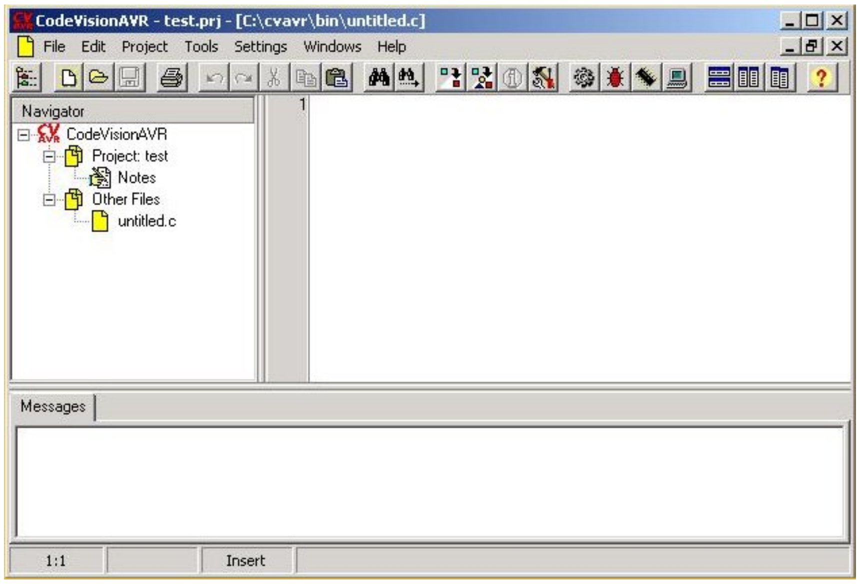

CodeVisionAVR - test.prj - [C:\cvavr\bin\untitled.c]
File Edit Project Tools Settings Windows Help
Navigator
CodeVisionAVR
Project: test
Notes
Other Files
untitled.c
Messages
1:1
Insert

Figure 5.4: Project "test" with new Source File

This new file needs to be saved. Under the *File* menu option click on *Save As.* Specify the name and folder in the new popup window and click *Save.* To add this file to the project, click on *Configure* under the *Project* menu option. This opens the window shown in Figure 5.1. After adding the source file to the project, the Navigator should show the source file under the project. The figure below shows a project called test with the new source file that is called test.c.

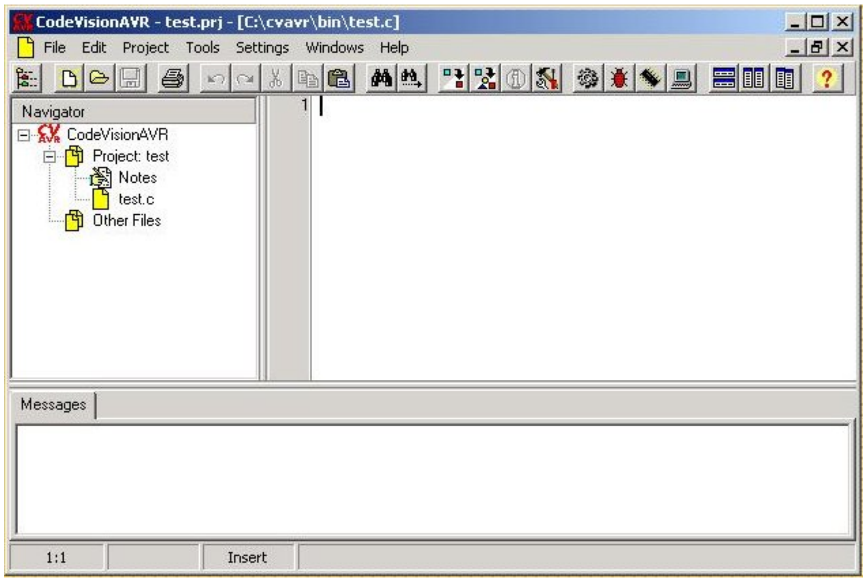
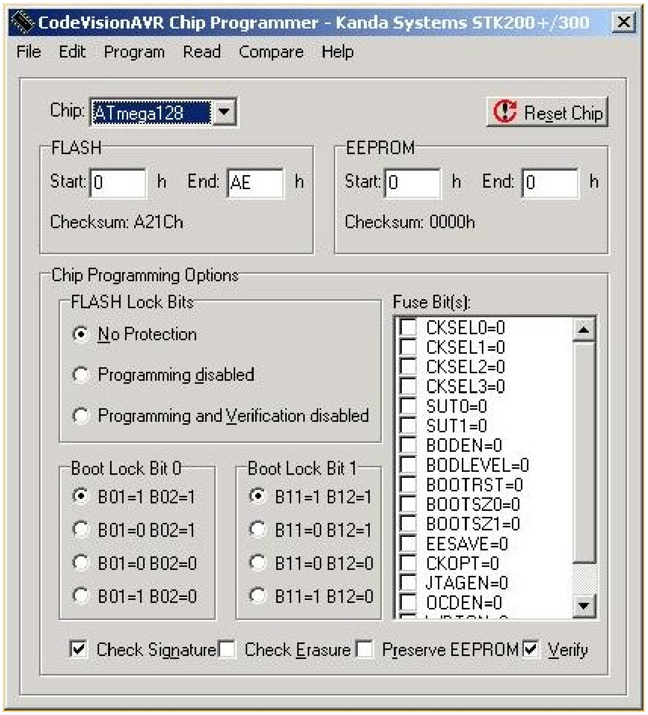

CodeVisionAVR - test.prj - [C:\cvavr\bin\test.c]
File Edit Project Tools Settings Windows Help
Navigator
CodeVisionAVR
Project: test
Notes
test.c
Other Files
Messages
1:1
Insert

Figure 5.5: Project "test" with Source File "test.c"

To Compile the source code press F9 or click on *Compile* under the *Project* menu option. An information window will pop up showing compile info and errors. If there are no errors the next step is run Make. Again under the *Project* menu, click on *Make* or press shift + F9. If there are no errors during the make process a window will pop up that allows you to program the chip. Make sure that the mega128.2 board is powered and then click on *Program*.

5.2 Loading *.hex files

When the software development has been done on a different IDE that does not have parallel programming capabilities, CodeVision can be used to program the AVR from a hex file. The Chip Programmer is used for this purpose. To open the Chip Programmer press shift + F4 or click on *Chip Programmer* under the *Tools* menu option. The figure below shows the Chip Programmer window.



Chip: ATmega128

FLASH
Start: 0 h End: AE h
Checksum: A21Ch

EEPROM
Start: 0 h End: 0 h
Checksum: 0000h

Chip Programming Options

FLASH Lock Bits
[✓ No Protection]
Programming disabled
Programming and Verification disabled

Boot Lock Bit 0
[✓ B01=1 B02=1]
B01=0 B02=1
B01=0 B02=0
B01=1 B02=0

Boot Lock Bit 1
[✓ B11=1 B12=1]
B11=0 B12=1
B11=0 B12=0
B11=1 B12=0

Fuse Bit(s):
☐ CKSEL0=0
☐ CKSEL1=0
☐ CKSEL2=0
☐ CKSEL3=0
☐ SUT0=0
☐ SUT1=0
☐ BODEN=0
☐ BODLEVEL=0
☐ BOOTRST=0
☐ BOOTSZ0=0
☐ BOOTSZ1=0
☐ EESAVE=0
☐ CKOPT=0
☐ JTAGEN=0
☐ OCDEN=0

☑ Check Signature ☐ Check Erasure ☐ Preserve EEPROM ☑ Verify

Figure 5.6: CodeVision Chip Programmer

To load the hex file, click on *Load FLASH* under the *File* menu option. Navigate to the file and click *Open.* To clear the AVR click on *Erase Chip* under the *Program* menu option. Lastly to program the AVR with the hex file, click on *FLASH* under the *Program* menu option. The ATmega128 should now be programmed.

6.0 C Source Code (test program)

```
#include <Mega128.h>
#include <mem.h>
//#include <stdlib.h>
#include <string.h>
#include <stdio.h>

//void initialization(void);
void delay_ms(unsigned int n);
void delay_us(unsigned int n);
void pokew(unsigned int addr, unsigned int data);
void cursor_home(void);
void home_line2(void);
//void fill_spaces(void);
void string2lcd(char *lcd_str);

unsigned char ext_data_byte[32767]  @0x8000;
unsigned char temp, read_byte;
unsigned int i, j, l;//, count;

//char addr_str[12] = {"Address: "};
//char data_wr[10] = {"WR:"};
//char data_rd[10] = {"RD:"};
//char space[2] = " ";
//char lcd_str[16];  //holds string to send to lcd
//char done_str[6] = {"Done!"};
//char hello_will[15]      = {"Hello William!"};
//char test2_ok[13]        = {"Test 2: Pass"};
char lcd_test[17]          = {"***-LCD_Test-***"};
//char switch_tst[12]        = {"Switch Test"};
char switch_test_end[17]   = {"Switch Test Done"};
char ir_test[14]           = {"IR Tx/Rx Test"};
char ir_results[17]        = {"IR TX & L4 blink"};
char ram_test[9]           = {"RAM Test"};
//char ram_error[15]         = {"RAM Test Error"};
char ram_success[17]       = {"RAM Test Success"};
char uart_test[10]         = {"UART Test"};
char uart_string[17]       = {"UART echo string"};
char switch_test_start[17] = {"Test pushbuttons"};
//char rock_on[14] = {"Tekbots Rock!"};
//char k, m;

void strobe_lcd(void){
//twiddles bit 3, PORTF creating the enable signal for the LCD
  PORTF = 0x08;
  PORTF = 0x00;
  }
```

```
void clear_display(void){
 SPDR = 0x00;    //command, not data
 delay_us(2);    //wait for data to leave SPI port
 SPDR = 0x01;    //clear display command
 delay_us(2);    //wait for data to leave SPI port
 strobe_lcd();   //strobe the LCD enable pin
 //delay_us(1640); //obligatory waiting for slow LCD
 delay_us(1900); //obligatory waiting for slow LCD
 }

void cursor_home(void){
 SPDR = 0x00;    //command, not data
 delay_us(2);
 SPDR = 0x02;   // cursor go home position
 delay_us(2);
 strobe_lcd();
 //delay_us(1640);
 delay_us(1900);
 }

 void home_line2(void){
 SPDR = 0x00;    //command, not data
 delay_us(2);
 SPDR = 0xC0;   // cursor go home on line 2
 delay_us(2);
 strobe_lcd();
 //delay_us(1640);  //shouldn't need this much time!
 delay_us(1900);
 }
/*
 void fill_spaces(void){
 int count;
 for (count=0; count<=15; count++){
     SPDR = 0x01; //set SR for data
     delay_us(2);
     SPDR = 0x20;
     delay_us(2);
     strobe_lcd();
     delay_us(40);
     }
 }
 */
void char2lcd(char a_char){
//sends a char to the LCD
//usage: char2lcd('H');  // send an H to the LCD
 SPDR = 0x01;   //set SR for data xfer with LSB=1
 delay_us(2);   //wait till data has left SPI data register
 SPDR = a_char; //send the char to the SPI port
 delay_us(2);   //wait till data has left SPI data register
 strobe_lcd();  //toggle the enable bit
 delay_us(160); //wait the prescribed time for the LCD to process
 }
```

```
void string2lcd(char *lcd_str){
//sends a string in FLASH to LCD
int count;
  for (count=0; count<=(strlen(lcd_str)-1); count++){
      SPDR = 0x01; //set SR for data
      delay_us(2);
      SPDR = lcd_str[count];
      delay_us(2);
      strobe_lcd();
      delay_us(160);
      }
}

void error_flash(void){
//flash PORTB LEDS on and off to indicate an error
 DDRB=0xFF;
 while(1){
   PORTB=0x00;
   delay_ms(1000);
   PORTB=0xFF;
   delay_ms(1000);
   }
 }

void init_ext_ram(void){
//External RAM will reside between 8000h - FFFFh.
//There will be 2 wait states for both read and write.
  PORTA=0x00;
  PORTC=0x00;
  PORTG=0x00;
  DDRA=0x00;
  DDRC=0xFF;
  DDRG=0xFF;
  MCUCR=0x80;
  XMCRA=0x42;
  XMCRB=0x80;
}

 void spi_init(void){
 /* Run this code before attempting to write to the LCD.*/
 DDRF=0x08;  //port F bit 3 is enable for LCD
 PORTB=0x00; //port B initalization for SPI
 DDRB=0x07;  //Turn on SS, MOSI, SCLK
 //Master mode, Clock=clk/2, Cycle half phase, Low polarity, MSB first
 SPCR=0x50;
 SPSR=0x01;
 }
```

```
void lcd_init(void){
 //initalize the LCD to receive data
  delay_ms(15);
  for(i=0; i<=2; i++){ //do funky initalize sequence 3 times
   SPDR = 0x00;
   delay_us(2);
   SPDR = 0x30;
   delay_us(2);
   strobe_lcd();
   //delay_us(4100);
   delay_us(6100);
   }

   SPDR = 0x00;
   delay_us(2);
   SPDR = 0x38;
   delay_us(2);
   strobe_lcd();
   delay_us(4100);

   SPDR = 0x00;
   delay_us(2);
   SPDR = 0x08;
   delay_us(2);
   strobe_lcd();
   delay_us(4100);

   SPDR = 0x00;
   delay_us(2);
   SPDR = 0x01;
   delay_us(2);
   strobe_lcd();
   delay_us(4100);

   SPDR = 0x00;
   delay_us(2);
   SPDR = 0x06;
   delay_us(2);
   strobe_lcd();
   delay_us(4100);

   SPDR = 0x00;
   delay_us(2);
   SPDR = 0x0E;
   delay_us(2);
   strobe_lcd();
   delay_us(4100);
}
```

```
void main (void)
  {

/**********port connectivity test **********/
/*
Toggle all bits 40 times on all ports A-G.
All jumpers should be removed to avoid driver conflicts
in any ports.
*/

/* needs i, j, temp as char variables */
//
  PORTA=0x00;
  DDRA=0xFF;
  PORTB=0x00;
  DDRB=0xFF;
  PORTC=0x00;
  DDRC=0xFF;
  PORTD=0x00;
  DDRD=0xFF;
  PORTE=0x00;
  DDRE=0xFF;
  PORTF=0x00;
  DDRF=0xFF;
  PORTG=0x00;
  DDRG=0xFF;
  for(l=0; l<=1; l++){ // do twice
    temp = 0x01;  //initalize temp
    for(j=0; j<=7; j++) {   //do for all bit positions
       PORTA = temp;
       PORTB = temp;
       PORTC = temp;
       PORTD = temp;
       PORTE = temp;
       PORTF = temp;
       PORTG = temp;
       delay_ms(500);
       PORTA = temp & 0x00;
       PORTB = temp & 0x00;
       PORTC = temp & 0x00;
       PORTD = temp & 0x00;
       PORTE = temp & 0x00;
       PORTF = temp & 0x00;
       PORTG = temp & 0x00;
       temp = temp << 1;  // left shift temp value
     } //for j
   }//for l
   //
/********** end of port connectivity test **********/

/*
After this point in the tests the LCD is intalized so
that it can be used to report success or failure of tests.
*/
//initalize the SPI port and then the LCD
spi_init();
```

```
lcd_init();
clear_display();

/********** LCD test **********/
/*
The string "***-LCD_Test-***" is sent to both LCD lines and held for 1sec.
Then LCD is then cleared.
*/
string2lcd(lcd_test);  //send string to LCD
home_line2();          //move cursor to line2
string2lcd(lcd_test);  //send string again
delay_ms(1000);        //erase after 1 sec
clear_display();

/********** pushbutton connectivity test **********/
/*
As each pushbutton is pushed, the corresponding PORTB
LED should light.  Multiple button pushes work also.
Jumper J12 must be removed so that portD bit 2 can
go high.  If J12 is left on, the IR detector probably
will see IR energy and pull portD bit 3 low. To terminate
this test, push S2 and S9 (bit 0 and 7) simutaneously.
*/

string2lcd(switch_test_start);  //indicate start of test

//turn off other unused ports for safety sake
SPCR=0x00;//kill spi so that portB is free
DDRA=0x00;
DDRC=0x0;
DDRE=0x00;
DDRF=0x08;
DDRG=0x00;

PORTD = 0x00;
DDRD = 0x00;  //PORTD all inputs
DDRB = 0xFF;  //PORTB all outputs
while (PIND != 0x7E){
  temp = PIND ^ 0xFF; //invert all bits of button inputs- active low
  PORTB = temp; // send the result out to LEDs
}

PORTB = 0x00;  //clear port B LEDs

//reinitalize the SPI port
spi_init();
lcd_init();
clear_display();
string2lcd(switch_test_end);  //send string again
delay_ms(1000);
/********** end of pushbutton connectivity test **********/
```

```
/********** UART1 Test **********/
/*
Echo characters back from DB9 connectors using PC as terminal.
Communication Parameters: 8 Data, 1 Stop, No Parity
Rcvr On, Xmtr On, Asynch Mode, Baud rate: 9600
Jumpers J15 and J19 must be in place for the MAX232 chip
to get data.  However nothing is usually connected to its
serial outputs for this UART port.  A better test will be
to send data to itself using the IR channel.

//initalize UART1
UCSR1A=0x00;
UCSR1B=0x18;
UCSR1C=0x06;
UBRR1H=0x00;
UBRR1L=0x67;
 while (1) {
 k=getchar(); //receive the character
 putchar(k); // and echo it back
 };
*/
/********** end of UART1 Test **********/

/********** 32Khz crystal test ***********/
/*
This test should blink port B pin
This test does not work correctly because it is running
off the 16MHz system clock and not the 32Khz clock.
//
PORTB=0x00;
DDRB=0x80;  //enable OC2 output pin
ASSR=0x08;  //this value should enable running off 32KHz crystal
TCCR2=0x19; //clear on compare mode, toggle OC2 on match
TCNT2=0x00; //initial count is 0x00
OCR2=0xFF;  //compare value is 0xFF
*/
/********** end of 32Khz crystal test ***********/

/********** UART0 Test **********/
/*
Echo characters from DB9 using loopback connector.
Communication Parameters: 8 Data, 1 Stop, No Parity
Rcvr On, Xmtr On, Asynch Mode, Baud rate: 9600
Jumpers J14 and J16 must be in place.
//  */
//
clear_display();
cursor_home();
string2lcd(uart_test);  //send string to LCD
home_line2();          //move cursor to line2



//initalize UART0
UCSR0A=0x00;
UCSR0B=0x18;
```

```
UCSR0C=0x06;
UBRR0H=0x00;
UBRR0L=0x67;

 for (i=0; i<=(strlen(uart_string))-1; i++) {
   putchar(uart_string[i]);  //send char by char to uart
   char2lcd(getchar());      //xfer from uart to LCD
   }
   delay_ms(2000); //hold message for 2 sec
//
/********** end of UART0 Test **********/


/********** IR LED and IR receiver test **********/
/*
This test turns on the IR led and then senses through
the IR reciever if it is on.  If so, it turns on the
PORTB LED bit 0.  Jumpers J12 and  and J13 must be
installed.
*/

clear_display();
string2lcd(ir_test);  //send string to LCD to announce test
home_line2();
string2lcd(ir_results); // indicate correct response

DDRD=0x08;
DDRB=0x01;
PORTB=0x00;

for (i=0; i<=3; i++){  //do test 4 times
  PORTD = 0x00;     //turn on IR Transmitter LED - active low assertion
  delay_ms(1);      //delay for receiver to sense IR

  if ((PIND & 0x04) == 0x04)  //Sense the IR receiver output - active low
    PORTB = 0x00;   //if output is high,(no IR detected) turn LED L4 off
  else
    PORTB = 0x01;   //if output is low (IR detected) turn on LED L4

  delay_ms(500);   // wait half a second to see the LED
  PORTD = 0x08;    //IR Transmitter off now
  delay_ms(1);     //delay for reciever to sense IR

  if ((PIND & 0x04) == 0x04)  //Sense the IR receiver output - active low
    PORTB = 0x00;  //if output is high,(no IR detected) turn LED L4 off
  else
    PORTB = 0x01;  //if output is low (IR detected) turn on LED L4

    delay_ms(500);  //wait half second to see LED off
} // for i
/********** end of IR LED and IR receiver test **********/
```

```
/********** RAM Test **********/
/*
This test writes a 0x00,0xA5,0x5A,0x00 pattern to each external
RAM cell and does a readback to determine if the write and read
was successful. The top two bits of PORTB LEDS will be strobing
very quickly while the test runs.  If the test fails, PORTB LEDS
will flash 1 sec on, 1 sec off.  If the test concludes successfully,
at least the top four PORTB LEDS will go out.
*/
  spi_init();
  lcd_init();
  clear_display();
  string2lcd(ram_test);  //send string to LCD to announce test
  delay_ms(1000);

  SPCR=0x00;       //kill spi so that portB is free
  init_ext_ram();  //initalize the external RAM for use
  DDRB=0xFF;       //setup PORTB to signal success or failure
  for (i=0; i<=32767; i++){   //check all RAM cells

   ext_data_byte[i] = 0x00;  //initalize location to 0x00
   read_byte = (unsigned char) peekb(&ext_data_byte[i]);// read back inital value
   if (read_byte != ext_data_byte[i]) error_flash(); //make sure values are identical

   ext_data_byte[i] = 0xA5;  //write the location with 0xA5
   read_byte = (unsigned char) peekb(&ext_data_byte[i]);// read back
   if (read_byte != ext_data_byte[i]) error_flash();

   ext_data_byte[i] = 0x5A;  //initalize location to 0x5A
   read_byte = (unsigned char) peekb(&ext_data_byte[i]);// read back
   if (read_byte != ext_data_byte[i]) error_flash();

   ext_data_byte[i] = 0x00;  //initalize location to 0x00
   read_byte = (unsigned char) peekb(&ext_data_byte[i]);// read back
   if (read_byte != ext_data_byte[i]) error_flash();

   PORTB++;  //twiddle the PORTB LEDS
   delay_us(100);  //slow things down enough to see the LEDS twiddle
   }

   PORTB=0x00; // if successful, cut all PORTB LEDs off
   spi_init(); //reinitalize spi and lcd again
   lcd_init();
   clear_display();
   string2lcd(ram_success);  //send string to LCD to announce test

   while(1){}  //wait here
   //clear_display();
   //delay_ms(500);
   //string2lcd(rock_on);  //send string to LCD to announce test
   //delay_ms(500);
   //}
 //
/********** end of RAM Test **********/

}//main
```

7.0 Parts List

```
# meg128 parts list
# parts in "()" are digikey part numbers 800-344-4539
# parts in "[]" are mouser part numbers 800-346-6873
# part documentation found in files inclosed in "{}"

#################################################################################################
MANUFACTURER   PART NUMBER           PACKAGE  REF DES            DESCRIPTION
*3M            929974-01-04                   J13,J14            4  contact, female, 1 row
               [517-974-01-04]
*3M            929975-01-05                   J16,J17,J18,J19,   10 contact, female, 2 row
               (929975-01-05-ND)              J20,J21            !! speical order !!
*3M            929834-01-36                   J3,J4,J5,J10,J11,  0.1” 1x2, male
               [517-834-01-36]                J12
*3M            929834-01-36                   J6,J7,J8,J9        0.1” 1x4 male
               [517-834-01-36]
*3M            929974-01-14                   H1                 14  contact, female, 1 row
               [517-974-01-14]
*3M            30310-6002HB                   J22                10 contact box header
               [517-30310-6002]
*NorComp       182-009-112-531                J1                 DB9 male
               (182-09M-ND)
*CUI Stack     PJ-002A                        J2                 2mm, coaxial power connector
               (CP-002A-ND)
*FOX           [559-FST327]          FST327   X2                 32khz crystal
*FOX           (300-8061-1-ND)       cm309s   X1                 16Mhz crystal
*Fairchild     HLMP1700              T100     L1,L2,L3,L4,L5,    Red LED
               (HLMP1700-ND)                  L6,L7,L8,L12
*Fairchild     HLMP1719              T100     L11                Yellow LED
               (HLMP1719-ND)
*Fairchild     HLMP1790              T100     L10                Green LED
               (HLMP1790-ND)
*Panasonic     PNA4612M00YB          special  U3                 IR detector
               (PNA4612M00YB-ND)
*Fairchild     QED123                T-1 3/4  L13                IR transmitter
               (QED123-ND)
*Mountain      101-0661              6mm      S1,S2,S3,S4,S5,    6mm tactile switch
 Switch        [101-0661]                     S6,S7,S8,S9
*E-Switch,Inc. EG2208                EG2208   S10                main on/off switch.
               (EG1941-ND)
*Vishay/Dale   IMC1210SY100K         1210     L14                10uH smt choke
               [70-IMC1210-10]
*Panasonic     EEV-FK1A221P          smt      C2                 220uF, low esr, 10V, elect.
               (PCE3390CT-ND)
*Panasonic     ECE-V1EA101XP         smt      C1                 100uF, 25V, elect.
               (PCE3354CT-ND)
*National Semi LM2940CS-5.0          TO-263   U7                 5v regulator
               (LM2940CS-5.0-ND)
*Cypress       CY62256L-70SNC        soic28   U1                 32Kx8 Static RAM
               (428-1079-ND)
*Atmel         ATmega128-16AC        tqfp     U8                 microprocessor
               (ATmega128-16AC-ND)
*Zetex,        2N7002                sot-23   T1,T2              enhancement mode mosfet
 Fairchild     (2N7002NCT-ND)
*TI, Fairchild 74HC02                soic     U9                 quad NOR
               (MM74HC02M-ND)
*TI,Fairchild  CD4053B               soic     U6                 analog mux
               (CD4053BCM-ND)
*TI            74AHC573              soic     U2                 octal latch
               (296-4612-5-ND)
*TI, Fairchild 74HC74                soic     U10                dual D-FF
               (MM74HC74AM-ND)
*TI            74HC164               soic     U11                8-bit shift register
               (296-8247-5-ND)
*ST Micro      ST232CD               soic     U5                 RS-232 transciever
               (ST232CD)
```

MANUFACTURER	PART NUMBER	PACKAGE	REF DES	DESCRIPTION
*XICON	LU-085-10K [32AG401]	vertical	R1	10K trim pot
*Intl Rectifier	10BQ040 (10BQ040-ND)	SMB	D2	Schottky diode, 1A, SMB pkg.
*Intl Rectifier	DF01STRR16 (DF01SDI-ND)	special	D1	1A bridge rectifier
*Linear Technology	LTC6900CS5 (LTC6900CS5-ND)	tsot-23	U4	prec osc
*Yageo America	0805CG180J9B200 (311-1102-1-ND)	sm805	C14,C15	Ceramic 18pF XTAL cap
*Yageo	08052R561K9B20D (311-1125-1-ND)	sm805	C16	560pF, ceramic chip cap
Panasonic	5% thick film resistors			
	47 (P47ACT-ND)	sm805	R2	
"	10K (P10KACT-ND)	sm805	R33,R35,R39,R4,R5,R6, R7,R8,R9,R10,R11,R3	
"	1.5K (P1.5KACT-ND)	sm805	R30,R38,R29,R28,R20,R21, R22,R23,R24,R25,R26,R27	
"	470 (P470ACT-ND)	sm805	R12,R13,R14,R15,R16,R17, R18,R19	
"	68 (P68ACT-ND)	sm805	R31,R32	
"	1K (P1.0KACT-ND)	sm805	R36,R37	
"	523K (P523KCCT-ND)	sm805	R34	
Panasonic	10% ceramic caps, 16V			
	0.1uF (PCC1812CT-ND)	sm805	C7,C8,C9,C10,C11,C12, C4,C5,C18,C19,C20, C21,C22,C13,C17, C23,C3,C6	

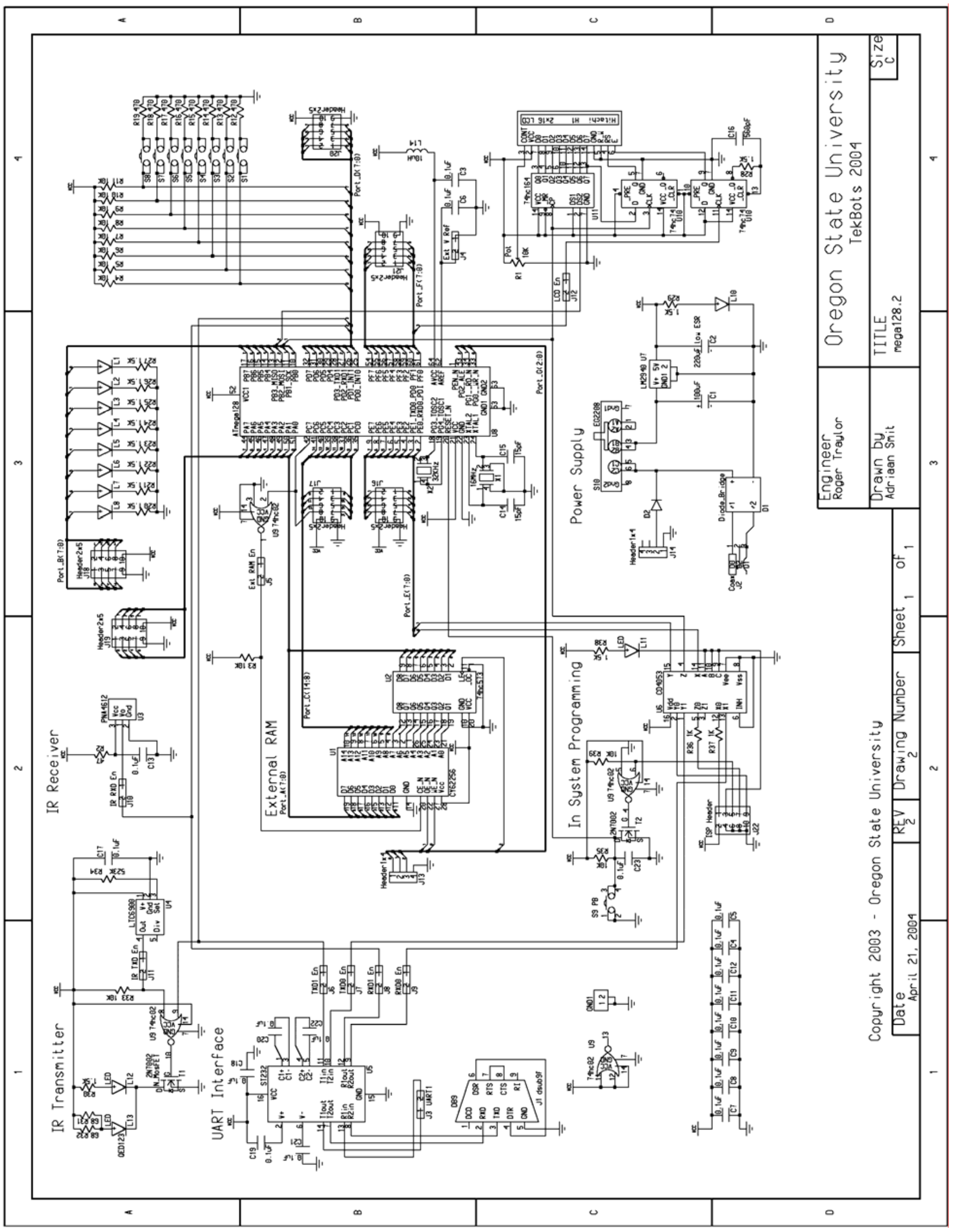
Parts external to the board assembly:

```
# meg128 off board parts list
# parts in "()" are Digikey part numbers 800-344-4539
# parts in "[]" are Mouser part numbers 800-346-6873
# parts in "<>" are CommCon part numbers 626-301-4200
# parts in "{}" are Timeline part numbers 310-784-5488
```

MANUFACTURER	**PART NUMBER**	**QUANTITY**	**DESCRIPTION**
Comm Con	<1105-14T>	1	1x14 male pin header, for mounting on LCD
AMP	(A26228-ND)	10	Economy 0.1 centerline low profile post shunts
MSC	85601243	4	M2.5 x 0.45 x 20mm phillips pan head machine screw
MSC	68024967	4	M2.5 flat washer, id 2.7mm, od 6mm, 0.5mm tall
MSC	68024041	4	M2.5 x 0.45 hex nut, hex size 5mm, height 1.6mm
3M	[517-SJ-5018BK]	4	0.23 tall x .50 square black bumpers
ABE	M0511-25-A-0	4	12mm tall, clear hole spacer, 4.5mm dia, for M2.5 screw (about 0.339 each in 1200 qty)
Samsung	{1622}	1	16x2 Samsung LCD (11/24/03 quote is for $6.00/ 75, 5.00/100 or more)
*Optrex America Inc.	DMC16249UB	1	16x2 LCD, no backlight

```
Sources:
  MSC:              our account#  is 01762507
                    http://www.mscdirect.com/
```

8.0 Schematic Diagram


Oregon State University
TekBots 2004
TITLE mega128.2
Engineer Roger Traylor
Drawn by Adriaan Smit
Sheet 1 of 1
Drawing Number 2
REV 2
Date April 21, 2004
Size C
Copyright 2003 - Oregon State University
Power Supply
In System Programming
External RAM
IR Receiver
IR Transmitter
UART Interface