ECE 441 Tech Demo

Design Artifacts

Authors: Trevor Horine

November 20, 2021

Contents

1 Schematic

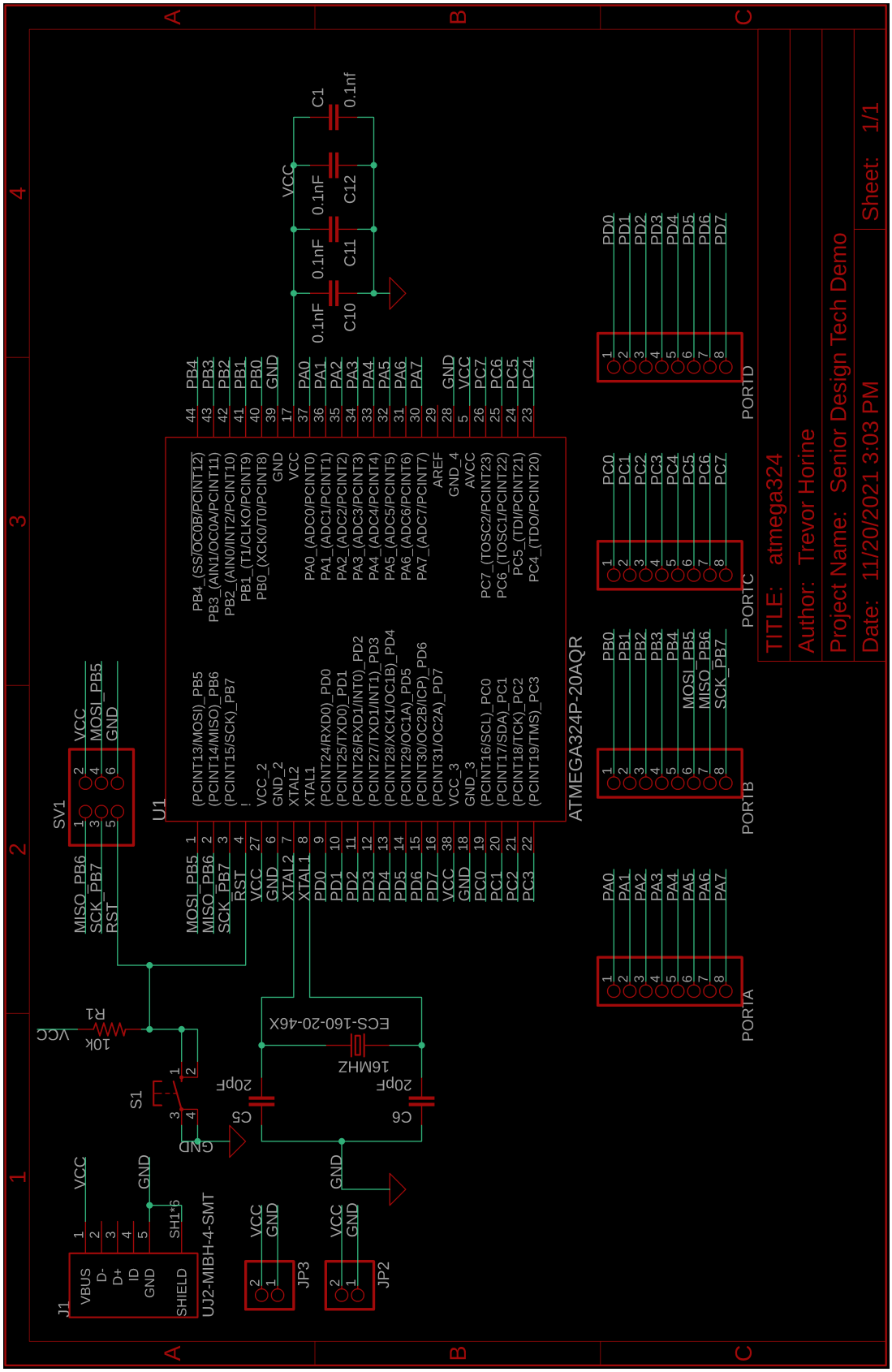

Figure 1: Detailed schematic microcontroller circuit.

2 PCB Layout

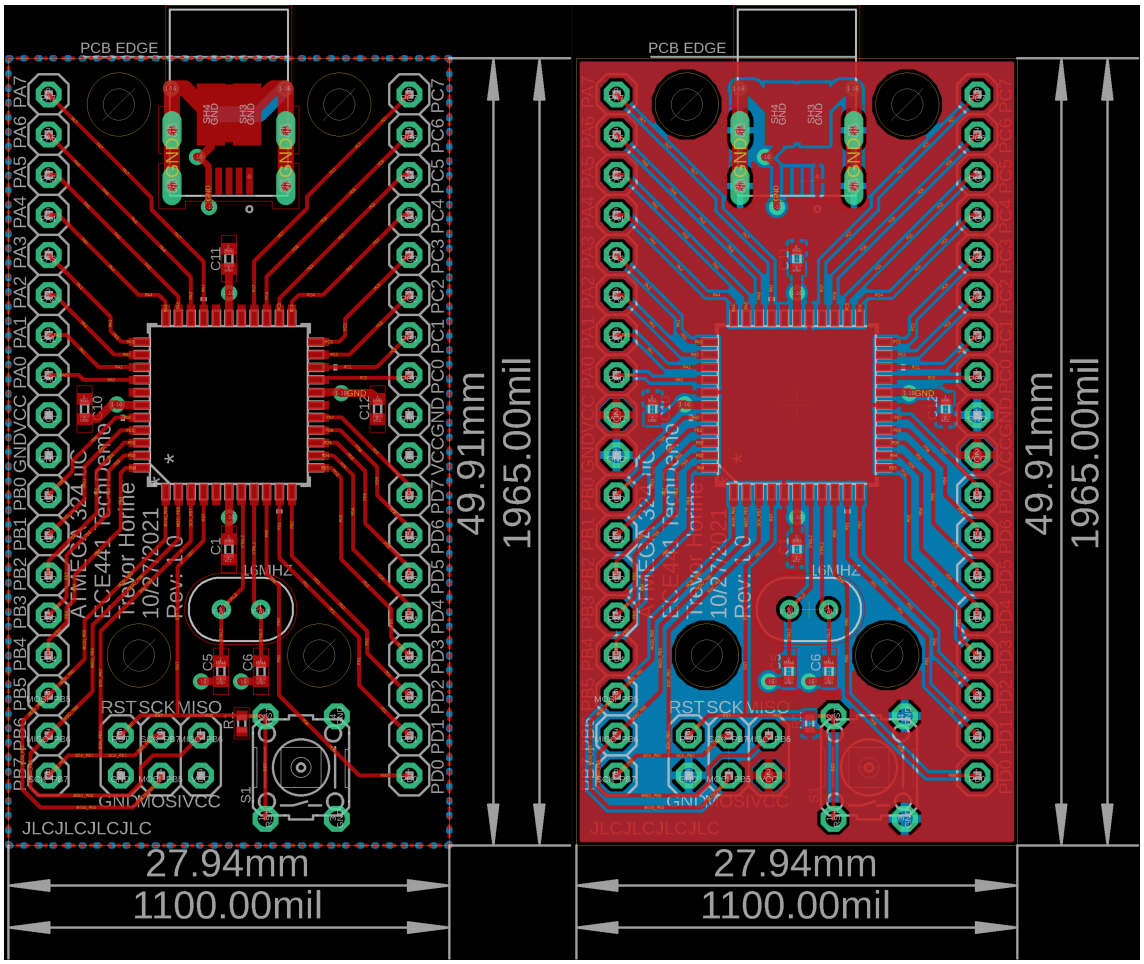

Figure 2: layout of PCB, left without GND and Vcc planes, right with.

3 Built Circuit

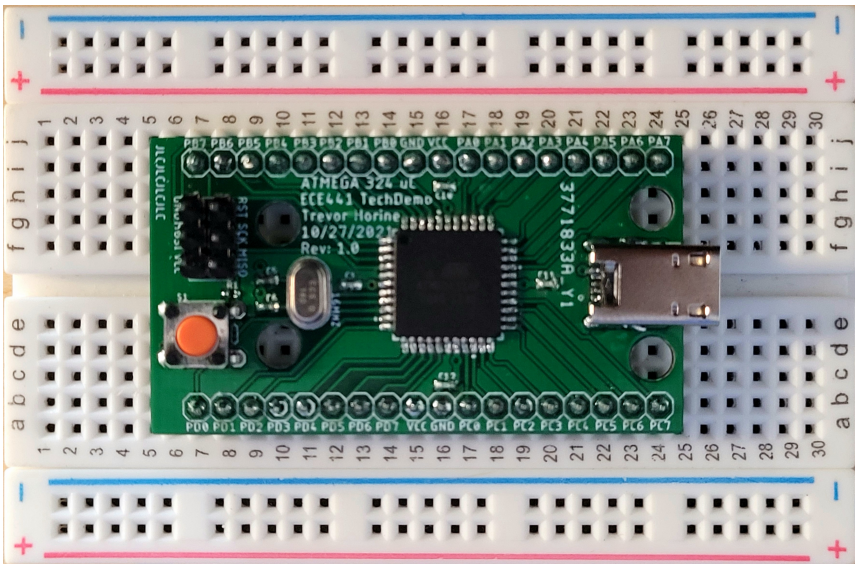

Figure 3: Picture of assembled PCB.

4 Testing Schematic

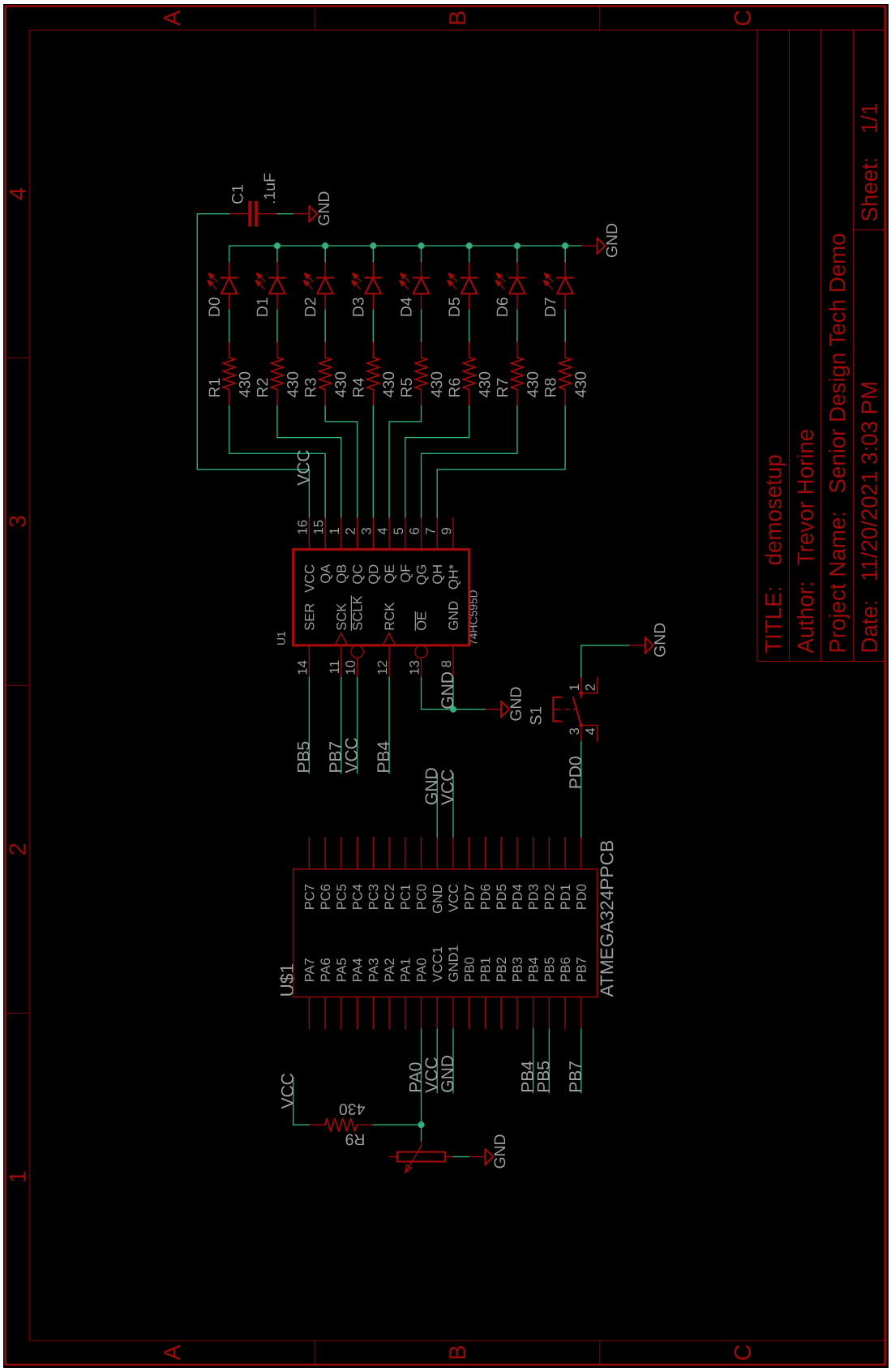

Figure 4: Detailed schematic testing setup.

5 Testing Circuit

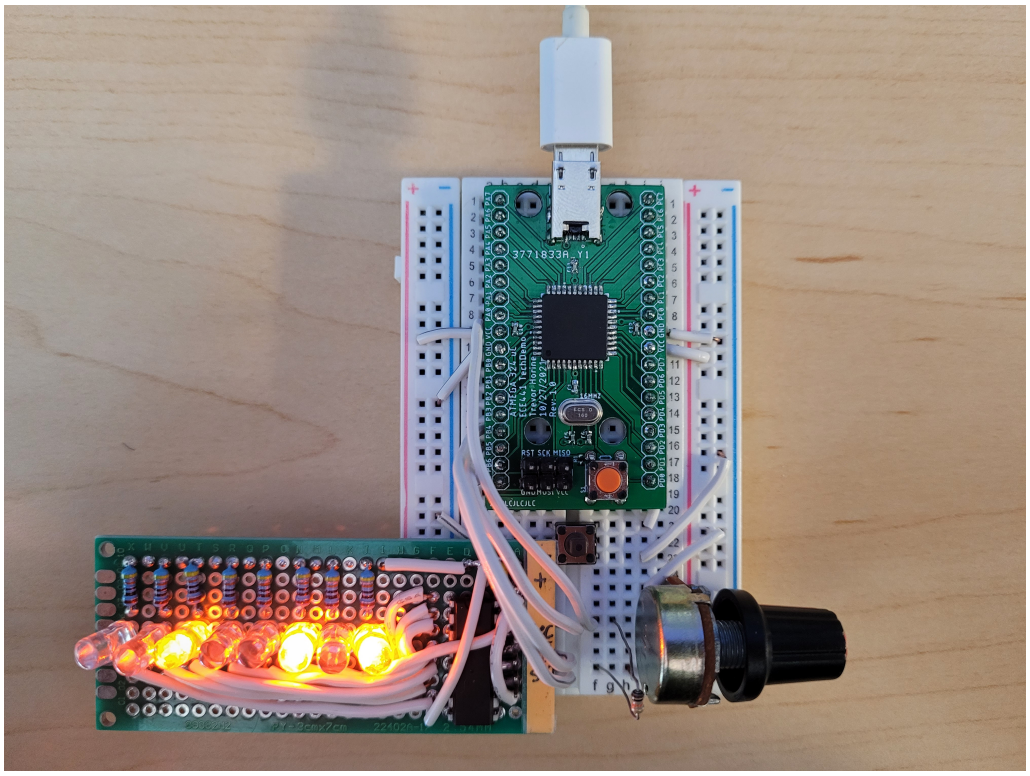

Figure 5: Testing setup to show analog and digital inputs as well as digital outputs.

6 Bill of Materials

Qty	Value	Device	Parts	Description
1	Button	10-XX	S1	OMRON SWITCH
2	pin headers	PINHD-1X21X02OCT	JP2	PIN HEADER
4	pin headers	PINHD-1X81X08OCT	PORTA, PORTB, PORTC, PORTD	PIN HEADER
1	pin headers	PINHD-2X3	SV1	PIN HEADER
4	0.1nF	C-EUC0402	C10, C11, C12, C1	CAPACITOR
1	10k	R-US_R0402	R1	RESISTOR
2	20pF	C-EUC0402	C5, C6	CAPACITOR
1	ATMEGA324P-20AQR	ATMEGA324P-20AQR	U1	uC
1	ECS-160-20-46X	ECS-160-20-46X	16MHZ	QUARTZ CRYSTAL T/H
1	UJ2-MIBH-4-SMT	UJ2-MIBH-4-SMT	J1	Micro B

7 Code

7.1 Test Code

```
// techdemo.c
// Trevor Horine 10/11/2021
// take in analog and digital input output digital output.
// analog input is middle of voltage divider sampled with ADC
// digital input is button conected to PORTD
// digital output is LED array attached to PORTB controlled with SPI

//Connections

//Voltage divider (analog in)
//      middle of voltage divider       PORTA0

//button (digital in)
//      one side                        PORTD0
//      other side                      ground

//Shift register LED display
//      reglck                          PORTB4 (SS_n)
//      srclk                           PORTB7 (sclk)
//      sdin                            PORTB5 (mosi)
//      oe_n                            ground
//      gnd                             ground
//      vcc                             vcc
//      sd_out                          no connect

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <util/delay.h>

volatile uint16_t ext_count = 0;        //used for timeing with TCNT0
uint8_t mode = 0;                       //used to swich between counting and ADC
uint16_t adc_result;                    //holds adc result
volatile uint8_t SPIout = 0;

//*************************************************************
//                      ISR for timer counter zero
//*************************************************************

ISR(TIMER0_COMPA_vect){                 //the isr function
  ext_count++;                          //increment the count
}//TIMER0_COMPA_vect

//*************************************************************
//                            init_tcnt0
//*************************************************************
//initalize timer/counter zero to CTC mode
void init_tcnt0(){
  TIMSK0 |= (1<<OCIE0A);                //enabel interupt for timmer0 comp
  TCCR0A |=  (1<<WGM01);                //CTC mode
  TCCR0B |= (1<<CS01);                  //prescale 1/8
```

```
52   OCR0A  |=  0xFF;                        //compare at 255
53 }//init_tcnt0
54
55 //*****************************************************************
56 //                            spi_init
57 //*****************************************************************
58 //initalize SPI
59 void spi_init(void){
60   DDRB   = 0xF0;                         //output mode for SS, MOSI,
       SCLK
61   SPCR0   = (1<<SPE0) | (1<<MSTR0);      //master mode, clk low on
       idle, leading edge sample
62   SPSR0   = (1<<SPI2X0);                 //choose double speed
       operation
63 }//spi_init
64
65 //*****************************************************************
66 //                            update_spi
67 //*****************************************************************
68 //update SPI
69 void update_spi(void){
70   SPDR0 = SPIout;                        //send to display
71   while (bit_is_clear(SPSR0,SPIF0)){} //wait till data sent out (
       while loop)
72   PORTB |= 0x10;                         //HC595 output reg - rising
       edge...
73   PORTB &= ~(0x10);                      //and falling edge
74 }//update_spi
75
76 //*****************************************************************
77 //                            read_adc
78 //*****************************************************************
79 //get adc value
80 void read_adc(void){
81   ADCSRA |= (1<<ADSC);                   //poke the ADSC bit and start
       conversion
82   while(bit_is_clear(ADCSRA, ADIF));  //spin while interrupt flag
       not set
83   ADCSRA |= (1<<ADIF);                   //its done, clear flag by
       writing a one
84   adc_result = ADC;
85   SPIout = (adc_result/4);               //div by 4 to get 8 bit and
       send to spi
86 }//read_adc
87
88 //*****************************************************************
89 //                            debounce_switch
90 //*****************************************************************
91 //read button
92 //Adapted from Ganssel's "Guide to Debouncing"
93 int8_t debounce_switch() {
94   static uint16_t state = 0;             //holds present state
95   state = (state << 1) | (! bit_is_clear(PIND, 0)) | 0xE000;
96   if (state == 0xF000) return 1;
97   return 0;
98 }//debounce_switch
99
100 //*****************************************************************
```

```
101 //                                  main
102 //*************************************************************
103 int main() {
104   DDRB = 0xFF;                          //set all port B pins to
      output
105   DDRD = 0x03;                          //set portd bit 2 as input
106   DDRA = 0xFF;                          //set all port A pins to
      output
107   PORTD = 0x03;                         //set port D all pullups
108   init_tcnt0();                         //initalize timer counter
      zero
109   spi_init();                           //initalize spi
110   PORTB |= 0x40;
111   //ADC setup
112   DDRA  &= ~(_BV(DDA0));                //make port A bit 0 the ADC
      input
113   PORTA &= ~(_BV(PA0));                 //port A bit 0 pullups must
      be off
114   ADMUX = 0x40;                         //single-ended input, PORTF
      bit 7, right adjusted, 10 bits
115                                         //reference is AVCC
116   ADCSRA =  0x97;                       //ADC enabled, don't start
      yet, single shot mode
117
118   sei();                                //enable global interrupts
119
120   while(1){
121     if(debounce_switch()) { //check button push
122       if (mode == 1) { //switch to count mode
123         mode = 0;
124         SPIout = 0;
125         ext_count = 0;
126       }//if
127       else if (mode == 0) {//switch to adc mode
128         mode = 1;
129       }//else if
130     }//if
131     if(mode == 0){//if count mode
132       if(ext_count == 520){             //incrment at 1sec
      intervals
133         SPIout++;
134         ext_count = 0;                  //clear ext_count
135       }//if
136     }//if
137     if(mode == 1){//if adc mode
138       read_adc();                       //read adc
139     }//if
140     update_spi();                       //update spi after either
      count or adc
141   }//while
142 }// main
```

7.2 Make File

```
1 PRG            = techdemo
2 OBJ            = $(PRG).o
3 MCU_TARGET     = atmega324p
4 F_CPU          = 16000000UL
```

```
DEFS            =
LIBS            =
CC              = avr-gcc
OPTIMIZE        = -O2    # options are 1, 2, 3, s
#OPTIMIZE       = -Os    # options are 1, 2, 3, s
# Override is only needed by avr-lib build system.

override CFLAGS         = -g -Wall $(OPTIMIZE) -mmcu=$(MCU_TARGET) $
    (DEFS) -DF_CPU=$(F_CPU)
override LDFLAGS        = -Wl,-Map,$(PRG).map

OBJCOPY         = avr-objcopy
OBJDUMP         = avr-objdump

all: $(PRG).elf lst text eeprom

$(PRG).elf: $(OBJ)
  $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^ $(LIBS)

clean:
  rm -rf *.o $(PRG).elf *.eps *.png *.pdf *.bak
  rm -rf *.lst *.map $(EXTRA_CLEAN_FILES) *~

program: $(PRG).hex
  avrdude -p m324p -c usbasp  -e -U flash:w:$(PRG).hex

lst:  $(PRG).lst

%.lst: %.elf
  $(OBJDUMP) -h -S $< > $@

# Rules for building the .text rom images

text: hex bin srec

hex:  $(PRG).hex
bin:  $(PRG).bin
srec: $(PRG).srec

%.hex: %.elf
  $(OBJCOPY) -j .text -j .data -O ihex $< $@

%.srec: %.elf
  $(OBJCOPY) -j .text -j .data -O srec $< $@

%.bin: %.elf
  $(OBJCOPY) -j .text -j .data -O binary $< $@

# Rules for building the .eeprom rom images

eeprom: ehex ebin esrec

ehex:  $(PRG)_eeprom.hex
ebin:  $(PRG)_eeprom.bin
esrec: $(PRG)_eeprom.srec

%_eeprom.hex: %.elf
  $(OBJCOPY) -j .eeprom --change-section-lma .eeprom=0 -O ihex $<
```

```
    $@
62
63 %_eeprom.srec: %.elf
64   $(OBJCOPY) -j .eeprom --change-section-lma .eeprom=0 -O srec $<
      $@
65
66 %_eeprom.bin: %.elf
67   $(OBJCOPY) -j .eeprom --change-section-lma .eeprom=0 -O binary $<
       $@
```