```c
/*
 * FileName: main.c
 * Version: 1
 *
 * Created: 10/26/2022 1:34:18 PM
 * Author: Ethan Zeronik
 *
 * Operations: Blink and LED every 500ms
 *
 * Hardware:
 *   Atmega2560          micro controller
 *   PORTB.7             LED13 active high
 */

/* NOTE: Includes */
#include <avr/io.h>

#include "Delay.h"
#include "Debugger.h"

/* NOTE: Custom Macros */
// TODO: None

/* NOTE: Global Variables */
// TODO: None

/* NOTE: Function prototypes */
// inits IO ports
void IO_init(void);

/* NOTE: Application implementation */
// the main loop of the function, provided to us
int main(void)
{
    DLY_initInterrupt();

    IO_init();

    initDebug();

    sei();

    while(1)
    {
        if(DLY_getTick() > 500)
        {
            PORTB = ~PORTB & 0x80;

            DLY_setTick(0);
        }
    }
}

/* NOTE: Function implementations */
void IO_init(void)
{
    // set port B.7 as an output
```

```
58      DDRB  = 0x80;
59      PORTB = 0x00;
60  }
61
```

```c
/*
 * FileName: main.c
 * Version: 1
 *
 * Created: 10/26/2022 2:23:36 PM
 * Author: Ethan Zeronik
 *
 * Operations: ADC using interrupts
 *
 * Hardware:
 *   Atmega2560          micro controller
 *   PORTA               LED bar
 *   PORTF.0             Potentiometer in
 */

/* NOTE: Includes */
#include <avr/io.h>

#include "AnalogToDigital.h"

/* NOTE: Custom Macros */
// TODO: None

/* NOTE: Global Variables */
// TODO: None

/* NOTE: Function prototypes */
// inits IO ports
void IO_init(void);

/* NOTE: Application implementation */
// the main loop of the function, provided to us
int main(void)
{
    ADC_initInterrupt();

    IO_init();

    sei();

    while(1)
    {
        // get the value and bit shift it right 2
        PORTA = ADC_getTenBitValueInterrupt(0) >> 2;
    }
}

/* NOTE: Function implementations */
void IO_init(void)
{
    // set portA as an output
    DDRA  = 0xFF;
    PORTA = 0x00;
}

```

```c
/*
 * FileName: main.c
 * Version: 1
 *
 * Created: 10/26/2022 2:46:31 PM
 * Author: Ethan Zeronik
 *
 * Operations: Encoder module
 *
 * Hardware:
 *   Atmega2560          micro controller
 *   PORTA               LED bar
 *   PORTD1              Direction encoder pin
 *   PORTD2              Encoder clock pin
 */

/* NOTE: Includes */
#include <avr/io.h>

#include "Encoder.h"

/* NOTE: Custom Macros */
// TODO: None

/* NOTE: Global Variables */
// TODO: None

/* NOTE: Function prototypes */
// inits IO ports
void IO_init(void);

/* NOTE: Application implementation */
// the main loop of the function, provided to us
int main(void)
{
    ENC_init();

    IO_init();

    sei();

    while(1)
    {
        // get the flagged byte
        uint8_t directionFlag = ENC_getDirection();

        // if the flag has been set
        if(directionFlag & 0x10)
        {
            // get the driection flag
            if((directionFlag & 0x01))
            {
                // move the bar left one
                PORTA = PORTA << 1 | 0x01;
            }
            else
            {
```

```
58                  // move the bar right one
59                  PORTA = PORTA >> 1;
60              }
61          }
62      }
63  }
64
65  /* NOTE: Function implementations */
66  void IO_init(void)
67  {
68      // set portA as an output
69      DDRA  = 0xFF;
70      PORTA = 0x00;
71  }
```

```c
/*
 * FileName: main.c
 * Version: 1
 *
 * Created: 10/28/2022 4:32:43 PM
 * Author: Ethan Zeronik
 *
 * Operations: Mode enabled combination of previous parts
 *
 * Hardware:
 *   Atmega2560          micro controller
 *   PORTA               LED bar
 *   PORTF.0             Potentiometer in
 *   PORTD1              Direction encoder pin
 *   PORTD2              Encoder clock pin
 *   PORTD2              Mode pushbutton
 */

/* NOTE: Includes */
#include <avr/io.h>

#include "Encoder.h"
#include "AnalogToDigital.h"
#include "Delay.h"

/* NOTE: Custom Macros */
// TODO: None

/* NOTE: Global Variables */
static uint8_t modeFlag = 0;

/* NOTE: Function prototypes */
// inits IO ports
void IO_init(void);

/* NOTE: Application implementation */
// the main loop of the function, provided to us
int main(void)
{
    IO_init();

    ENC_init();
    ADC_initInterrupt();
    DLY_initInterrupt();

    sei();

    while(1)
    {
        // the watchdog led
        if(DLY_getTick() > 500)
        {
            PORTB = ~PORTB & 0x80;

            if(modeFlag == 0)
            {
                PORTA = ~PORTA & 0x01;
```

```
            }

            DLY_setTick(0);
        }

        // the mode switcher
        if(modeFlag == 1)
        {
            // get the value and bit shift it right 2
            PORTA = ADC_getTenBitValueInterrupt(0) >> 2;
        }
        else if(modeFlag == 2)
        {
            // get the flagged byte
            uint8_t directionFlag = ENC_getDirection();

            // if the flag has been set
            if(directionFlag & 0x10)
            {
                // get the driection flag
                if((directionFlag & 0x01))
                {
                    // move the bar left one
                    PORTA = PORTA << 1 | 0x01;
                }
                else
                {
                    // move the bar right one
                    PORTA = PORTA >> 1;
                }
            }
        }
    }
}

// the mode button interupt
ISR(INT3_vect)
{
    if(modeFlag < 2)
    {
        modeFlag++;
    }
    else
    {
        modeFlag = 0;
    }

    PORTA = 0x00;
}

/* NOTE: Function implementations */
void IO_init(void)
{
    // set port B.7 as an output
    DDRB  = 0x80;
    PORTB = 0x00;

```

```
115     // set port A.7 as an output
116     DDRA  = 0xFF;
117     PORTA = 0x00;
118
119     // set port D.3 as an input
120     DDRD  = 0x00;
121     PORTD = 0x08;
122
123     // set interrupt 3 to on rising edge
124     EIMSK = (1 << INT3);
125     EICRA = (1 << ISC31);
126 }
127
```

```c
/*
 * FileName: Delay.c
 * Version: 1
 *
 * Created: 10/26/2022 2:28 PM
 * Author: Ethan Zeronik
 *
 * Operations: encoder definition
 */

#ifndef Encoder_h_INCLUDED
#define Encoder_h_INCLUDED

#if defined(__cplusplus)
extern "C" {
#endif

#pragma message("WARNING: this module uses PORTD1:2 for interrupt")

#include <avr/interrupt.h>
#include <stdio.h>

/* NOTE: Custom Types */
// TODO: None

/* NOTE: Function prototypes */
// init for the encoder
void ENC_init(void);
// gets the raw value for the port
uint8_t ENC_getValue(void);
// returns the flagged direction
uint8_t ENC_getDirection(void);


#if defined(__cplusplus)
} /* extern "C" */
#endif

#endif // Encoder_h_INCLUDED
```

```c
/*
 * FileName: Delay.c
 * Version: 1
 *
 * Created: 10/26/2022 2:28 PM
 * Author: Ethan Zeronik
 *
 * Operations: encoder implementation
 */

/* NOTE: Includes */
#include "Encoder.h"

#include <avr/io.h>

/* NOTE: Local declarations */
// TODO: None

/* NOTE: Global Variables */
static uint8_t value     = 0;
// upper byte is flag lower is direction
static uint8_t direction = 0;

/* NOTE: Local function implementations */
void ENC_init(void)
{
    // turn on the clock and direction inputs
    DDRD  |= 0x06;
    PORTD |= 0x06;

    // interupt 2 to enabled falling edge
    EIMSK |= (1 << INT2);
    EICRA |= (1 << ISC21);
}

uint8_t ENC_getValue(void)
{
    return value;
}

uint8_t ENC_getDirection(void)
{
    // cache the flag state
    uint8_t dir = direction;

    // reset the flag in the global
    direction = 0x00;

    // return the cached state
    return dir;
}

ISR(INT2_vect)
{
    // if pin is high
    if(PIND & 0x02)
    {
```

```
58          // set the flag
59          direction = 0x11;
60
61          // increment if it won't overflow
62          if(value < 255)
63          {
64              value++;
65          }
66      }
67      else
68      {
69          // set the flag
70          direction = 0x10;
71
72          // decrement if it won't overflow
73          if(value > 0)
74          {
75              value--;
76          }
77      }
78  }
```

```h
/*
 * FileName: Delay.h
 * Version: 1
 *
 * Created: 10/18/2022 7:26 PM
 * Author: Ethan Zeronik
 *
 * Operations: header for the delay submodule
 */

#ifndef Delay_h_INCLUDED
#define Delay_h_INCLUDED

#if defined(__cplusplus)
extern "C" {
#endif

#pragma message("WARNING: this module uses Timer 0 for delays")

#include <avr/interrupt.h>
#include <stdio.h>

#define F_CPU 16000000UL

/* NOTE: Custom Types */
// TODO: None

/* NOTE: Function prototypes */
// init registers for delay
void     DLY_init(void);
// init for the 1ms interrupt service
void     DLY_initInterrupt(void);
// delay for an amount of ms
void     DLY_ms(double ms);
// gets the ISR tick value
uint16_t DLY_getTick(void);
// sets the ISR tick value
void     DLY_setTick(uint16_t t);

#if defined(__cplusplus)
} /* extern "C" */
#endif

#endif // Delay_h_INCLUDED
```

```c
/*
 * FileName: Delay.c
 * Version: 1
 *
 * Created: 10/18/2022 7:26 PM
 * Author: Ethan Zeronik
 *
 * Operations: create a custom delay function
 */

/* NOTE: Includes */
#include "Delay.h"

#include <avr/io.h>

/* NOTE: Local declarations */
// TODO: None

/* NOTE: Global Variables */
static uint16_t tick = 0;

/* NOTE: Local function implementations */
void DLY_init(void)
{
    // reset counter to 0
    TCNT0 = 0;

    // normal mode
    TCCR0A = 0x00;
    TCCR0B = 0x00;

    OCR0A = 0;
}

void DLY_initInterrupt(void)
{
    // interrupt mode
    TCCR0A = 0x00;
    TCCR0B = 0x04;
    TCNT0  = 0;
    OCR0A  = 62;
    TIMSK0 = (1 << OCIE0A);
}

void DLY_ms(double ms)
{
    size_t time = (((ms / 1000.0) * F_CPU) / 1024);

    if(ms <= 16)
    {
        OCR0A = time;

        // prescalar of 1024
        TCCR0B = 0x05;

        while((TIFR0 & (1 << OCF0A)) == 0)
        {
```

```
58              // do nothing
59          }
60
61          // stop the timer
62          TCCR0B = 0x00;
63          // clear the overflow flag
64          TIFR0 |= (1 << OCF0A);
65          TCNT0 = 0;
66          OCR0A = 0;
67      }
68      else
69      {
70          OCR0A = (((1 / 1000.0) * F_CPU) / 1024);
71
72          for(size_t i = 0; i < ms; i++)
73          {
74              // prescalar of 1024
75              TCCR0B = 0x05;
76
77              while((TIFR0 & (1 << OCF0A)) == 0)
78              {
79                  // do nothing
80              }
81
82              // stop the timer
83              TCCR0B = 0x00;
84              // clear the overflow flag
85              TIFR0 |= (1 << OCF0A);
86              TCNT0 = 0;
87          }
88
89          OCR0A = 0;
90      }
91  }
92
93  uint16_t DLY_getTick(void)
94  {
95      return tick;
96  }
97
98  void DLY_setTick(uint16_t t){
99      tick = t;
100 }
101
102 ISR(TIMER0_COMPA_vect)
103 {
104     TCNT0 = 0;
105     tick++;
106 }
107
```

```c
/*
 * FileName: AnalogToDigital.h
 * Version: 1
 *
 * Created: 10/19/2022 12:47 AM
 * Author: Ethan Zeronik
 *
 * Operations: header for the adc submodule
 */

#ifndef AnalogToDigital_h_INCLUDED
#define AnalogToDigital_h_INCLUDED

#if defined(__cplusplus)
extern "C" {
#endif

#include <avr/interrupt.h>
#include <stdio.h>

#define F_CPU 16000000UL

/* NOTE: Custom Types */
// TODO: None

/* NOTE: Function prototypes */
// init registers for adc
void   ADC_init(void);
// init adc for interrupt mode
void   ADC_initInterrupt(void);
// returns the value of the given channel
double ADC_getTenBitValue(uint16_t channel);
// gets the 10 bit value on channel o
uint16_t ADC_getTenBitValueInterrupt(uint16_t channel);

#if defined(__cplusplus)
} /* extern "C" */
#endif

#endif // AnalogToDigital_h_INCLUDED
```

```c
/*
 * FileName: AnalogToDigital.c
 * Version: 1
 *
 * Created: 10/19/2022 12:47 AM
 * Author: Ethan Zeronik
 *
 * Operations: basic adc implementation
 */

/* NOTE: Includes */
#include "AnalogToDigital.h"

#include <avr/io.h>

/* NOTE: Local declarations */
// TODO: None

/* NOTE: Global Variables */
static uint16_t readInterrupt = 0;

/* NOTE: Local function implementations */
void ADC_init(void)
{
    // ten bit one way mode
    ADCSRA = (1 << ADEN) | (1 << ADPS1) | (1 << ADPS0);

    // 5v reference
    ADMUX = (1 << REFS0);

    ADCSRB = 0x00;
}

void ADC_initInterrupt(void)
{
    ADC_init();

    ADCSRA |= (1 << ADIE);
}

double ADC_getTenBitValue(uint16_t channel)
{
    uint16_t result = 0;

    // select the channel
    ADMUX  = (ADMUX & 0xe0) | channel;
    ADCSRB = (ADCSRB & 0xf7) | (channel >> 2);

    // start conversion
    ADCSRA |= (1 << ADSC);

    // wait for conversion
    while((ADCSRA & (1 << ADSC)) == 1)
    {
        // do nothing
    }

```

```
58      // save result
59      result = ADCL;
60      result = result | (ADCH << 8);
61
62      return result / 1024.0;
63  }
64
65  uint16_t ADC_getTenBitValueInterrupt(uint16_t channel)
66  {
67      // select the channel
68      ADMUX  = (ADMUX & 0xe0) | channel;
69      ADCSRB = (ADCSRB & 0xf7) | (channel >> 2);
70
71      // start conversion
72      ADCSRA |= (1 << ADSC);
73
74      return readInterrupt;
75  }
76
77  ISR(ADC_vect)
78  {
79      readInterrupt = ADCL;
80      readInterrupt = readInterrupt | (ADCH << 8);
81  }
```