EE4524/ED5502 Project 2 (Lab Weeks 10-13)

(Version 1. 23 March 2022)

This project is worth 20% of the total module assessment.

Description:

Write a single ATMega328P program that to run on an Arduino UNO R3 with Arduino Motor Shield and UL OER Shield that will:

- Start, stop and change the direction and speed of a DC motor using PWM, based on data read over the serial port
- Measure the period, and high and low pulse widths of an input signal applied to Port B bit 0 (the Input Capture pin) and report the reading to the user. If the external chip producing this signal has stopped oscillating for a time, report this to the user when pulse widths or signal periods are requested.
- Turn on or off Bit 6 (PORTD bit 6) of the LED array based on the time period measured on PortB bit 0 (the Input Capture pin)
- Read the **ADC2** input and report the reading to the user.
- Turn on or off bit 7 (PORTD bit 7) of the LED array based on the value detected on the ADC input
- Toggle the bit 4 (PORTD bit 4) at 125ms (approximately) using the Timer0 Overflow Interrupt, when requested by the user (same timing as Project 1). When this is selected, in the same Timer0 Overflow Interrupt move a Servomotor position forward and backwards
- Display the current setting of PORTD (output)
- Display the input read from PINB (input)

Operation of the Program:

The program controls the speed of a DC motor using the Arduino Motor Shield, using PWM and the Timer/Counter 2 output OC2B, where a zero PWM value stops rotation and a PWM value of 100% corresponds to the maximum motor speed. PORTB Bit 4, controls the Direction: 1 means Forward and 0 means Reverse.

The program can also start or stop the motor using a Brake signal on PORTB Bit 1, where 1 means Halt or stop motor rotation and 0 means Go or start the motor rotation.

The **OC2B** signal from the ATMega328P is shared with PORTD Bit 3. The PWM output on OC2B is controlled by the data value written to the OCR2B register.

The program changes the position of a servomotor connected to PORTB bit3 (**OC2A** output).

The ADC2 input is driven by a potentiometer on the Shield. The ADC should be initialised so that it runs in Automatic Trigger mode with retriggering from the Timer0 Timer Overflow event.

The Timer/Counter0 is set up to both retrigger ADC data acquisition, to toggle LED bit 4 (PORTD Bit 4), and change the position of the servomotor using a similar technique to that used in Project 1.

The Timer/Counter2 output OC2B is used as the DC Motor PWM output, and OC2A is used as the servomotor PWM output, but Timer/Counter2 interrupts are **not** enabled.

The Timer/Counter1 input, ICP1, is connected to a 555 Timer, which produces a periodic signal in a frequency range from approximately 2 kHz to 13 kHz, controlled by a potentiometer on the shield.

The program continuously reads the serial port for characters and responds to the following single character commands received over the serial port:

'F' or 'f': Set Direction of motor spin to Forward.

'R' or 'r': Set Direction of motor spin to Reverse.

‘B’ or ‘b’: Stop motor irrespective of the motor speed previously selected – ie Turn on Brake

‘G’ or ‘g’: Start the motor (depends on the motor speed previously selected) – ie Turn off Brake

‘0’ to ‘9’: Set DC motor speed using the OC2RB register, to a value from 0 (off) to 90% of full intensity. You must convert the char value to an integer and set the PWM based on this value.

‘S’ or ‘s’: Report the current value of the OCR2B register to the user.

‘T’ or ‘t’: Report the period of the 555 Timer in microseconds.

‘L’ or ‘l’: Report the time taken by the low pulse of the 555 Timer signal in microseconds

‘H’ or ‘h’: Report the time taken by the high pulse of the 555 Timer signal in microseconds

'C' or 'c': Continuously report the Timer input period in microseconds.

‘E’ or ‘e’: Stop continuous reporting of Timer input.

‘A’ or ‘a’: Report the ADC conversion result. This is the ADC value.

‘V’ or ‘v’: Report the ADC conversion result in mV. You must convert the ADC value to mV. You must also display the ADC source.

'M' or 'm':   Continuously report the ADC conversion result in mV. You must convert the ADC value to mV. You must also display the ADC source.

‘N’ or ‘n’: Stop continuous reporting of ADC input.

‘W’ or ‘w’: Toggle the LED bit 4 at 125ms and move the servomotor to its next position

‘U’ or ‘u’: Stop toggling LED bit 4 and stop moving the servomotor

‘D’ or ‘d’: Report to the user the state (in hex) of PORTD outputs.

‘P’ or ‘p’: Report to the user the state (in hex) of PINB inputs.

‘I’ or ‘I’: Select ADC14, Internal 1.1V BG ref as the ADC input. This means changing the ADMUX register and the change must be done in the ADC ISR.

‘j’ or ‘J’: Select ADC2 as the ADC input. This means changing the ADMUX register and the change must be done in the ADC ISR.

All other characters are ignored.

Note: The data reported to the user is in ASCII format, and should be formatted with some text to show what is being displayed. (e.g. “Timer period = 103us” or “Voltage = 2345 mV”).

Code Structure:

Initialisation Section:

```
Initialise Bit 3 of PORTD to output (needed as DC Motor PWM output), also set Bits 7, 6, 4 of PORTD
as outputs
Initialise Bit 3 of PORTB to output (needed as Servo PWM output)
Initialise Timer/Counter 0 for Timer0 Overflow Interrupts
Initialise Timer/Counter 1 for input capture and rollover interrupts (see Homework 2)
Initialise Timer/Counter 2 for Fast PWM – Note Timer 2 interrupts are not used.
Initialise USART Serial Port
Initialise ADC, automatic trigger set as Timer0 Overflow
// Use a separate initialisation function for each initialisation task.
Enable global interrupts
while(1)
        {
        Test Serial port to check for new character and Parse input
        Test New Input Capture data flag to see if new ICP data has been captured.
        If yes
                {
                If the continuous timer value display is selected, report new timer value to the user
                on the USART
                Clear the New Input Capture data flag
                }
                else Test New ADC data flag to see if new ADC data has been captured.
                If yes
                        {
                        If the continuous ADC display is selected, report new ADC voltage to the
                        user on the USART
                        Clear the ADC data flag
                        }
        }  // End of while
//=============================================================
ADC ISR
        Read new ADC result into a variable
        Set new ADC_value flag.
        if (ADC result) > 3.5 volts  // work out the threshold value needed
                Turn on PORTD bit 7
        else
                Turn off PORTD bit 7
        If a change of ADC input is required, write to ADMUX to change the ADC input
//=============================================================
Timer1 Overflow ISR
        Increment the timer1 overflow counter
//=============================================================
Timer1 Input Capture ISR
        Calculate the period, and high and low times of the input signal in microseconds
        Set up any variables for the next interrupt.
        Clear the timer 1 overflow counter
        Set New Input Capture data flag
        if input time period > 150 us
                turn on PORTD bit 6
        else
```

```
                turn off PORTD bit 6
//==================================================================
Timer0 Overflow ISR
        Increment the Timer0 overflow counter
        If it is greater than or equal to the count for 12.48ms
                Clear the Timer0 overflow counter
                If the Toggle LED command is true
                        Toggle PORTD bit 4
                        Move the Servomotor position
//==================================================================
USART Transmit Complete ISR
        Send new data byte from transmit print queue (see example program)
==================================================================
```

Initialisations in more detail

Timer/Counter0 setup:
Timer/Counter0 Clock Source: CLKIO/1024
TCNT0 set for 12.48ms, Timer0 Overflow Interrupts enabled

Timer/Counter1 setup:
Timer/Counter1 Clock source: CLKIO/8
All Timer1 outputs disabled
Timer/Counter1 Input Capture set for falling edge with noise control turned OFF
Timer/Counter1 Input Capture and Timer1 Overflow Interrupts enabled.

Timer/Counter2 setup:
All Timer/Counter2 interrupts disabled
Timer/Counter2 Clock Source: CLKIO/1024
Timer/Counter2 OC2B output enabled (Clarification: this means set DDRD Bit 3 as output)
Timer/Counter2 OC2A output enabled (Clarification: this means set DDRB Bit 3 as output)
Fast PWM Mode, TOP = 0xFF
Clear OC2B on Compare Match when Upcounting
OCR2B used to set PWM duty cycle (Clarification: this is the result of the preceding lines)
OCR2A settings for Servo

ADC setup:
ADC Channel 2 used
AVCC selected as the ADC Reference Voltage
ADLAR ADC in 10-bit mode (means just make the ADLAR bit = 0)
AutoTrigger enabled for Timer/Counter0 Overflow re-trigger mode
ADC Prescaler: 128
ADC Interrupt Enabled

USART setup:
8-bit data
No parity
9600 baud (but try 115200 later)
RX enabled RX interrupt disabled

TX Enabled. TXC interrupt enabled

Also initialise any variables you are using

Note on the Serial Port operation:

Use the sample code in provided on Sulis to receive and transmit data bytes, using the serial TX complete interrupt. I suggest using sprintf for formatted printing.

Note on arithmetic: use integer arithmetic throughout. Do not use floating point arithmetic or formatting unless *absolutely necessary*.

Marking Scheme:

Initialisations, comments, overall program structure and program demonstration: 4%

ADC input changing, ADC reading and reporting: 3%

Timer0 Bit Toggling and ADC retrigger: 2%

DC motor Control using PWM, Direction and Brake: 2%

Serial Mode – reporting data in command driven and continuous modes: 3%

Timer clock period and pulse high and low times reporting: 2%

LED output bit control: 1%

Servo Motor Control: 2%

High Baud Rate Serial option (Highest possible baud rate > 9600, using a standard baud rate): 1%

Timetable:

Demonstrate your program running on Arduino + shields in Week 13 (or before) if possible.

Submit the final versions of your programs using Sulis, before the due date on Sulis.