## EE4524/ED5502 Spring 2020 Project 2 (Lab Weeks 9-13)

This project is worth **40%** of the total module assessment.

V1.1: Removed Timer/Counter1 setup (used for extra challenge only), added Timer/Counter0 setup

V1.2: Corrected typos ‘M’ or ‘m’ and changed reference to OCR2B

## Description:

Write a single ATMega328P program that reads the LM35 temperature sensor and the Light Dependant Resistor (LDR) on the Multifunction Shield, and also varies the intensity of the BLUE LED that is part of the RGB LEDs on that board. The program will:

Read a character using the serial port. Depending on the character that has been read, the program will vary the brightness of the LED using PWM, switch over the ADC from measuring temperature or brightness (LDR), or report requested data (suitably formatted) over the serial port.

Read the ADC0, ADC1 or ADC2 input (depending on the option selected by the user, temperature is the default after RESET).

## Operation of the program:

The program controls the brightness of the Blue LED that is part of the RGB LEDs, using PWM and the Timer/Counter 2 output OC2A, where a zero PWM value turns OFF the LED and a PWM value of 255 Turns ON the LED at maximum brightness.

The **OC2A** signal from the ATMega328P is shared with PORTB Bit 3. The PWM output on OC2A is controlled by the data value written to the OCR2A register.

The **ADC2** input comes from the LM35 Temperature Sensor on the board. The **ADC1** input comes from the Light Dependent Resistor (LDR) on the board. The ADC should be initialised so that it runs in Auto Trigger Mode, with Timer/Counter 0 Overflow selected as the Trigger event. This allows ADC measurements to be taken at a set time interval **NOT** Free Running Mode as in Project 1.

The Timer/Counter2 output OC2A is used as the PWM output, but Timer/Counter2 interrupts are **not** enabled.

#### The program responds to the following single character commands received over the serial port:

'M' or 'm': Set the ADC to read the output of the LM35 temperature sensor (ADC2 input channel).

'N' or 'n': Set the ADC to read the output of the LDR (ADC1 input channel).

‘P’ or ‘p’: Set the ADC to read the output of the potentiometer (ADC0 input channel).

Note: You must change the ADMUX input selection in the ADC ISR only. So ‘m’, ‘M’, ‘n’, ‘n’, ‘N’, ‘p’ or ‘P’ set status options showing what ADC source the user has selected. After saving the ADC result, the ADC ISR checks these bits against the current ADMUX MUX bits. If they are different from the user selected option, the ADMUX MUX bits are changed. This is to ensure that the ADC source is not changed while a conversion is underway.

‘0’ to ‘9’: Set the Blue LED brightness using the OC2RA register, to a value from 0 to 90% of maximum brightness (fully on). You must convert the char value to an integer and set the PWM based on this value.

‘T’ or ‘t’: Report the Temperature read from the LM35 in degrees Centigrade if the ADC has been set to ADC2 as its input source; give a warning otherwise.

‘L’ or ‘l’: Report the state of light in the environment if the ADC has been set to ADC1 as its input source; give a warning otherwise. Report ‘Bright’ if the ADC reading is greater than 512; report ‘Dark’ otherwise.

‘A’ or ‘a’: Report the ADC conversion result. This is the ADC value; do this no matter what ADC input has been selected.

‘V’ or ‘v’: Report the ADC conversion result in mV. You must convert the ADC value to mV. Do this no matter what ADC input has been selected

'C' or 'c': **Continuously** report the ADC conversion result. If the LM35 (ADC2) is selected, report temperature in degrees Centigrade; if the LDR (ADC1) is selected report ‘Bright’ or ‘Dark’; if the potentiometer (ADC0) is selected report the measured voltage in millivolts,

‘E’ or ‘e’: Stop continuous reporting of the ADC conversion result.

‘S’ or ‘s’: Report the current value of the OCR2A register to the user.

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. “LM35 Temperature = 23 deg C”).

**Code Structure:**

Initialisation Section:

Initialise Bit 3 of PORTB to output (needed as PWM output)

Initialise Timer/Counter 2 for PWM – Note Timer 2 interrupts are not used.

Initialise Timer/Counter 0 to overflow every 16 ms but do NOT enable the Timer0 Overflow interrupts.

Initialise USART Serial Port

Initialise ADC for temperature (LM35) reading and with Timer0 overflow as the ADC trigger

// Use a separate initialisation function for each initialisation task.

Enable global interrupts

while(1)

{

Poll Serial port to check for new character and do action based on the input

Poll 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 result (temp, bright/dark or voltage depending on the ADC source) to the user on the USART

Clear the ADC data flag

}

}

//================================================================

ADC ISR

Read new ADC result into a variable

Set new ADC\_value flag.

Check the state of the user selection of ADC input, compare with the state of the ADMUX MIX bits. If different change the MUX bits to the user selected option.

Clear the Timer0 Overflow Flag

//=================================================================

USART Transmit Complete ISR

Send new data byte from transmit print queue (see example program)

==================================================================

## Initialisations in more detail

**Timer/Counter0 setup:**

All Timer/Counter0 interrupts disabled

Timer/Counter0 Clock Source: CLKIO/1024

TCNT0 set to an initial value that will cause timer overflow after 16 ms

**Timer/Counter2 setup:**

All Timer/Counter2 interrupts disabled

Timer/Counter2 Clock Source: CLKIO/256

Timer/Counter2 OC2A output enabled (Clarification: this means set DDRB Bit 3 as output)

Phase Correct PWM Mode, TOP = 0xFF

Clear OC2A on Compare Match when Upcounting

OCR2A used to set PWM duty cycle (Clarification: this is the result of the preceding lines)

Set initial OCR2A value to 0 (LED off)

**ADC setup:**

ADC Channel 2 used (initially)

AVCC selected as the ADC Reference Voltage

ADLAR ADC in 10-bit mode (means just make the ADLAR bit = 0)

AutoTrigger enabled for Tier/Counter 0 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.

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

**Marking Scheme:**

Initialisations, comments, and overall program structure: 5%

ADC reading, source selecting under user control and reporting: 15%

LED brightness control using PWM: 5%

Serial Mode – reporting data in command driven and continuous modes: 10%

High Baud Rate Serial option (Highest possible baud rate > 9600): 5%

**Timetable:**

Submit the final versions of your programs using Sulis, before **5pm, Friday 1 May 2020**.

EXTRA CHALLENGE (worth an extra 10% in the final assessment): Using an external wire, connect PORTB bit 3 to PORTB bit 0. Use the techniques in Homework 2 Q3 to find the TimePeriodHigh and TimePeriodLow and the total TimePeriod (=TimePeriodHigh+TimePeriodLow) and report these to the user using the extra commands

‘g’ or ‘G’: Report TimePeriodLow in microseconds

‘h’ or ‘H’: Report TimePeriodHigh in microseconds

‘w’ or ‘W’: Report TotalTimePeriod in microseconds

If you’re doing this extra challenge you will need the following Timer/Counter1 initialisation setup

**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.