# ET4047 Project 2 (Lab Weeks 9-13)

(Version 1.0. 02 November 2024)

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

## Description:

Write a single ATmega4809 program that to run on an Arduino Nano Every with a Micro Servo Motor and a UL OER Shield that will:

* Move a servomotor at a speed determined by the user, where the servo motor position is set using PWM.
* Measure the period, and high and low pulse widths of an input signal applied to Port E bit 3 (used as 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 Bits 4, 5 & 6 of the LED array based on the time measured on PortE bit 3 (the Input Capture pin)
* Read the analog voltage on the **AIN3** input and report the reading to the user. Set or clear Bit 7 of the LED array based on the ADC reading.

## Operation of the Program:

The program controls the movement rate of a servomotor, using PWM where the Timer/Counter TCA0 output WO-0, sets the servomotor position. A PWM ‘on-time’ value of 1ms with 20ms period sets a position of -90 degrees and a PWM ‘on-time’ value of 2ms sets a position of +90 degrees. The program should move between -90 degrees and +90 degrees in 25 steps.

For servomotor control, the PWM frequency should be 50Hz (Period = 20ms).

The PWM value is changed in a TCB2 Interrupt Service Routine at a rate determined by a user value sent over the Serial Port.

The **WO-0** signal from the ATmega4809 is shared with PORTA Bit 0. The PWM output on WO-0 is controlled by the data value written to the TCA0.SINGLE.CMP0 register.

The AIN3 input is driven by a potentiometer on the Shield. The ADC0 should be initialised so that it runs in ***periodic conversion mode***, where the conversion period is set by the TCB2 Timer/Counter.

The TCA0 WO-0 is used as the Servomotor PWM output, but TCA0 interrupts are **not** enabled.

The 555 Timer produces a periodic signal in a frequency range from approximately 2 kHz to 13 kHz, controlled by a potentiometer on the shield. It is connected to PORTE Pin 3.

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

**‘0’ to ‘9’:**

Set the servomotor rotation speed. The servomotor position is set in the TCB2 ISR, so this time should be translated into a software interrupt count that determines when to move the servomotor to its next position.

Servomotor rotation speed timings:

|  |  |  |  |
| --- | --- | --- | --- |
| ‘0’ | No movement | ‘1’ | 1.0s per step |
| ‘2’ | 0.75s per step | ‘3’ | 0.5s per step |
| ‘4’ | 0.4s per step | ‘5’ | 0.25s per step |
| ‘6’ | 0.2s per step | ‘7’ | 0.15s per step |
| ‘8’ | 0.1s per step | ‘9’ | 0.05s per step |

‘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. Set Continuous Timer Mode.

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

‘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.

'M' or 'm': Continuously report the ADC conversion result in mV. You must convert the ADC value to mV. Set Continuous ADC Mode.

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

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:

Call Clock\_Init() to ensure 20 MHz clock for CLK\_CPU and CLK\_PER

Initialise Pin 0 of PORTA to output (needed as Servomotor PWM output), also set PORTC Pin 6 (LED bit 4), PORTB Pin 2 (LED bit 5), PORTF Pin 4 (LED bit 6), and PORTA Pin 1 (LED bit 7) as outputs.

Initialise Timer/Counter TCA0 for Single Slope PWM with TCA0 clock set to CLK\_PER Divided by 16 using WO-0 output on PORTA Pin 0

Initialise Timer/Counter TCB0 for input capture combined Frequency and Pulse Width Measurement

Initialise Timer/Counter TCB1 to detect TCB0 timeouts (Advanced feature).

Initialise Timer/Counter TCB2 for period timer mode to set a period of 5ms.

Initialise USART Serial Port

Initialise ADC, automatic trigger set for Event Control, where the event is generated by TCB2 Timer 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

If the continuous timer value display (Continuous Timer Mode Set) is selected

{

Test New Input Capture data flag to see if new ICP data has been captured.

If Yes {

Report new timer value to the user on the USART

Clear the New Input Capture data flag

}

}

else // Continuous Timer Mode Not Set

{

If the continuous ADC display (Continuous ADC Mode Set) is selected

Test New ADC data flag to see if new ADC data has been captured.

If yes

{

Report new ADC voltage to the user on the USART

Clear the ADC data flag

}

}

} // End of while

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

ADC0 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 PORTA Pin 1 (LED bit 7)

else

Turn off PORTA Pin 1 (LED bit 7)

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

Timer TCB0 Input Capture ISR

Reset the TCB0 Interrupt Flag using TCB0.INTFLAGS

Calculate the period, and high and low times of the input signal in microseconds

Set New Input Capture data flag

if input time period > 200 us

turn on PORTC Pin 6 (LED bit 4)

else

turn off PORTC Pin 6 (LED bit 4)

if input time period > 300 us

turn on PORTB Pin 2 (LED bit 5)

else

turn off PORTB Pin 2 (LED bit 5)

Turn off PORTF Pin 4 (LED bit 6)

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

TimerTCB1 Overflow ISR

**Challenge** use TCB1 to detect if the 555 timer has stopped oscillating and Turn on PORTF Pin 4 (LED bit 6) if yes

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

TimerTCB2 Overflow ISR

Reset the TCB2 Interrupt Flag using TCB2.INTFLAGS

Increment a software interrupt counter and compare with the value selected by the user when entering numbers ‘0’ to ‘9’.

When equal move the servomotor to its next position.

Clear the software interrupt counter.

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

USART Transmit Complete ISR

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

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

**Initialisations in detail**

**Clock\_Init(); and Event System Initialisation, EVSYS\_Init();**

You may use the supplied example code on Brightspace. Clock\_Init() enables 20MHz operation and EVSYSInit() sets PORTE Pin 3 as the Event Generator and TCB0 as the Event User

**EVSYS setup:**

Set the EVSYS Channel 4 generator to be PORTE Bit 3

Set the EVSYS Channel 4 User to be TCB0

Set the EVSYS Channel 2 generator to be TCB2 CAPT (this will be TCB2 overflow in this case)

Set the EVSYS Channel 2 User to be ADC0

**Timer/Counter TCA0 setup:**

Set TCA0 to Single Slope PWM (CTRLB)

Set TCA.SINGLE.PER or PERBUF for 50Hz PWM frequency (24999)

Set TCA0.SINGLE.CMP0 for nominal -90degrees initial position – On time = 1ms

Timer/Counter TCA0 Clock Source: CLK\_PER divided by 16 and TCA0 enabled (CTRLA)

(These are suggested settings – you may use your own if you can make them work)

**Timer/Counter TCB0 setup:**

Timer/Counter TCB0 Clock source: CLK\_PER/2 and Enable TCB0 (CTRLA)

Configure TCB0 in Clock Frequency - Pulse Width Measurement mode (CTRLB)

Enable Capture interrupt (INTCTRL)

Enable Event Input and Falling Event Edge (EVCTRL)

**Timer/Counter TCB1 setup:**

Timer/Counter TCB1 Clock source: CLK\_PER/2 and Enable TCB1 (CTRLA)

The remaining settings are left as a challenge.

**Timer/Counter TCB2 setup:**

Timer/Counter TCB2 Clock source: CLK\_PER/2 and Enable TCB3 (CTRLA)

Configure TCB2 for Periodic Interrupt Mode (CTRLB), TCB2 interrupts enabled.

Set TCB2.CCMP to give a period of 5ms.

**ADC0 setup:**

ADC0 AIN3 used (MUXPOS)

AVDD selected as the ADC0 Reference Voltage

10-bit resolution, Free Running Mode NOT selected, ADC0 enabled

Enable Event controlled start conversion (EVCTRL)

Simple conversion selected (no sample accumulation)

Initial delay set to 0

ADC0 clock prescaler: 64

ADC Interrupt on Result Ready Enabled

**USART setup:**

8-bit data

No parity

115200 baud

RX enabled RX interrupt disabled

TX Enabled. TXC interrupt enabled

**Note on the Serial Port operation:**

Use the sample code in provided on Brightspace 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.

**Marking Scheme:**

Attendance, initialisations, comments, overall program structure and **program demonstration**: 4%

ADC0 reading and reporting using TCB2 timings: 3%.

Servo Motor Control using PWM: 3%

Serial Mode – reporting data in command driven and continuous modes: 2%.

Timer clock period and pulse high and low times reporting: 3%.

LED output bits indication: 1%.

Servomotor rotation speed control: 2%.

Detection of 555 Timeout: 2%.

**Timetable:**

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

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

# ATmega4809 mapping to Arduino Nano and UNO D0-D13 digital I/O

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Arduino UNO | Arduino Nano Every | ATmega4809 Port Pin | Shield Use | ATmega328P |
| D0 | D0 | PORTC bit 5 (PC5) | LED 0 | PORTD bit 0 |
| D1 | D1 | PORTC bit 4 (PC4) | LED 1 | PORTD bit 1 |
| D2 | D2 | PORTA bit 0 (PA0) | LED 2 | PORTD bit 2 |
| D3 | D3 | PORTF bit 5 (PF5) | LED 3 | PORTD bit 3 |
| D4 | D4 | PORTC bit 6 (PC6) | LED 4 | PORTD bit 4 |
| D5 | D5 | PORTB bit 2 (PB2) | LED 5 | PORTD bit 5 |
| D6 | D6 | PORTF bit 4 (PF4) | LED 6 | PORTD bit 6 |
| D7 | D7 | PORTA bit 1 (PA1) | LED 7 | PORTD bit 7 |
| D8 | D8 | PORTE bit 3 (PE3) | Capture | PORTB bit 0 |
| D9 | D9 | PORTB bit 0 (PB0) |  | PORTB bit 1 |
| D10 | D10 | PORTB bit 1 (PB1) |  | PORTB bit 2 |
| D11 | D11 | PORTE bit 0 (PE0) |  | PORTB bit 3 |
| D12 | D12 | PORTE bit 1 (PE1) | Push Btn0 | PORTB bit 4 |
| D13 | D13 | PORTE bit 2 (PE2) | Push Btn1 | PORTB bit 5 |

# ATmega4809 mapping to Arduino Nano and UNO A0-A5 analog Input (and digital I/O)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Arduino UNO | Arduino Nano Every | ATmega4809 Port Pin | ATmega4809 Ain | Shield Use |
| A0 | A0 | PD3 | AIN[3] | Potentiometer Out\* |
| A1 | A1 | PD2 | AIN[2] |  |
| A2 | A2 | PD1 | AIN[1] | Potentiometer Out\* |
| A3 | A3 | PD0 | AIN[0] |  |
| A4 | A4 | PA2/SDA |  | LED 8\*\* |
| A5 | A5 | PA3/SCL |  | LED 9\*\* |
|  |  |  |  |  |
| SDA | SDA | PA2/SDA |  |  |
| SCL | SCL | PA3/SCL |  |  |

\* Depending on JP7 setting

\*\* If JP5 and JP6 are connected