Due on 11/17, submit a single file as doc, docx, or pdf in Canvas.

Please follow the example code attached in the package.

1. (20%) A motor rotates at 1000 RPM (rotation per minute). We can use timer and delay to generate a PWM signal to slow the motor to 10RPM.

(a) The period of PWM is 20ms. What should be the length of duty in each PWM period to reduce the rotation speed to 10RPM? Assume no mechanical delay in motor on the change of PWM signal.

10 = x\*1000

x = 1/100

==

= .2ms or 200us

(b) Modify the code in hw5.q1 to make a PIC24E program to reduce the motor to 10RPM. Note that you cannot use \_\_delay\_ms(), and need to find another proper delay function from "libpic30.h". Copy your code of main.c and timerint.h in your report and comment your code.

#ifndef TIMERINT\_H

#define TIMERINT\_H

#define TimerInt

#define FCY 4000000UL

#include <timer.h>

extern bool volatile timerFired;

#define period 20 // millisecond

#define duty 4// millisecond

#define MILLISECOND(x) ((FCY / 1000 \* (x)) >> 8)

void configureTimer(unsigned long duration);

#endif /\* TIMERINT\_H \*/

#include <stdbool.h>

#include "timerint.h"

#include "pwm.h"

#include "libpic30.h"

int main() {

initPWM();

configureTimer(period);

timerFired = false;

while (1) {

if (timerFired) {

timerFired = false;

pwmOn();

\_\_delay\_us(200);

pwmOff();

}

}

return 0;

}

2. (25%)

A Queue is designed to be deterministic with a fixed maximum length.

The struct of Queue is below: a currEvent and an array.

{currEvent, {[0: Ex], [1: Ex], [2: Ex], [3: Ex]}}

An example of Queue is below.

{E3, {[0: END], [1: E2], [2: END], [3: E1]}}

It means the current pending event is E3. The following events in order are E1 and E2.

The currEvent E3 points to [3: E1], where 3 is the index of the array and E1 means the next event E1.

Then, the next event E1 points to [1: E2], where 1 is the index of the array and E2 means the next event E2.

Then, the next event E2 points to [2: END], where 2 is the index of the array and END means the end, i.e. no more events.

When the Queue has no events, it has the following:

{END, {[0: END], [1: END], [2: END], [3: END]}}

(a) Assume in the system, the events happened in the following time order:

T0: no events in queue.

T1: E1 is added to the queue.

T2: E0 is added to the queue.

T3: E3 is added to the queue.

T4: E1 is popped from the queue.

T5: E2 is added to the queue.

Fill in the table in hw5.q2.docx to show the queues after the time points, T1, T2, T3, T4, T5. Include the table in your submission (not as a separte file).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time | currEvent | [0: x] | [1: x] | [2: x] | [3: x] |
| T0 | END | [0: END] | [1: END] | [2: END] | [3: END] |
| T1 | E1 | [0: END] | [1: END] | [2: END] | [3: END] |
| T2 | E1 | [0: END] | [1: E0] | [2: END] | [3: END] |
| T3 | E1 | [0: END] | [1: E0] | [2: END] | [3: END] |
| T4 | E0 | [0: E3] | [1: END] | [2: END] | [3: END] |
| T5 | E0 | [0: E3] | [1: END] | [2: END] | [3: E2] |

(b) The Queue is declared in event.h. Implement the four functions of Queue in event.c. main.c is provided for your reference only.

Copy your code in event.c to your report and comment your code.

#include "event.h"

// same event only exists once in que

// initialize all elements in Queue to END

void initQueue(Queue\* que) {

for(size\_t i = 0; i <QLEN;i++)

que->q[i] = END;

}

// test if Queue has pending events

bool isEmpty(Queue\* que) {

Return (que->curEvent == END);

}

// return the current pending event and replace the current with the next event

uint8\_t pop(Queue\* que) {

uint8\_t ev = que->curEvent;

que->curEvent = que->q[ev];

que->q[ev] = END;

return ev;

}

// add a new event to Queue.

// if ev is not in que, alway append ev to the end of que

// if ev is in que already, do not append ev

void append(Queue\* que, uint8\_t ev) {

if(ev < END) {

uint8\_t cev = que->curEvent;

if(cev == END) {

que->curEvent = ev;

} else {

uint8\_t second\_val = 0;

while (cev != END && second\_val < QLEN) {

if(cev == ev) {

break;

}

if(que->q[cev] == END) {

que->q[cev] = ev;

que->q[ev] = END;

break;

}

cev = que->q[cev];

second\_val++;

}

}

}

}

3. (15%)

A system has a 40MHz processor. It needs to process audio signals with a 40000Hz interrupt.

(a) How many cycles are between two interrupts?

1 Cycle = 1/(40\*10^6)

frequency between interrupts = 40\*10^3Hz

(40\*10^3)(1/(40\*10^6)) = 1000 cycles

(b) Assume 20 cycles are needed for switching context. What is the percentage of time for switching context?

(20/1000) \* 100 = 2%

(c) Assume 380 cycles are needed for the ISR. What is the percentage of time in total for processing an interrupt?

20 + 380 = 400 cycles

(400/1000) \* 100 = 40%

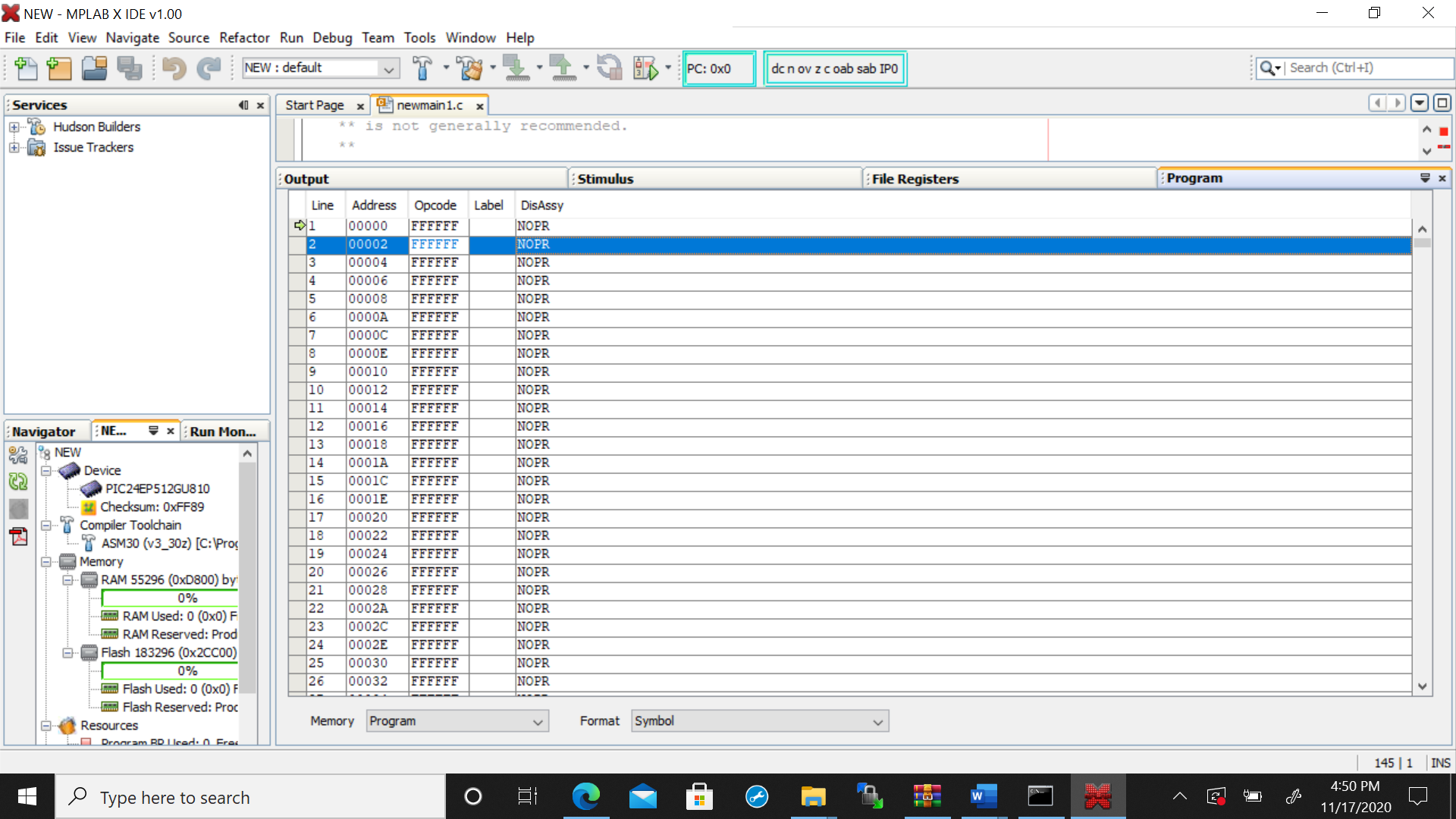
4. (20%)

Read the linker script of pic24ep in hw5.q4. Run a pic24ep program in MPLab X, and watch memory of the program in debugger.

a) Fill the table in hw5.q4.docx and include the table in your submission (not as a separte file).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Memory Section | Start Address | End Address | Length | The First 4 Bytes in HEX |
| data | 0x1000 | 0xDFFF | 0xD000 | 0x000D0000 |
| reset | 0x0 | 0x3 | 0x4 | 0x00000004 |
| ivt | 0x4 | 0x200 | 0x1FC | 0x000001FC |
| program | 0x200 | 0x557FD | 0x555FE | 0x00003FF8 |

b) Show a screen shot of watching memory in debugger.



5. (20%) Use avrgcc to compile the code in hw5.q5 with different optimizations: O0, O1, O2, O3, Os.

Makefile is provided, and you may run "make" to get the results.

Otherwise, you can refer to Makefile to choose proper compiler command and options to get results.

Copy and paste your results in the table in hw5.q5.docx. Include the table in your submission (not as a separte file).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Optimization | Text | Data | Bss | Dec |
| O0 | 1456 | 0 | 77 | 1533 |
| O1 | 828 | 0 | 77 | 905 |
| O2 | 814 | 0 | 77 | 891 |
| O3 | 814 | 0 | 77 | 891 |
| Os | 818 | 0 | 77 | 895 |