Homework 4

Due on 11/13, submit in hard copy.

1. (30%) 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

(b) Modify the code in hw4.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.

\*timerint.h

#ifndef TIMERINT\_H

#define TIMERINT\_H

#define TimerInt

#define FCY 4000000UL

#include <timer.h>

extern bool volatile timerFired;

#define period 20000 // millisecond

#define duty 0.2 // millisecond

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

void configureTimer(unsigned long duration);

#endif /\* TIMERINT\_H \*/

\*main.c

#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(duty);

pwmOff();

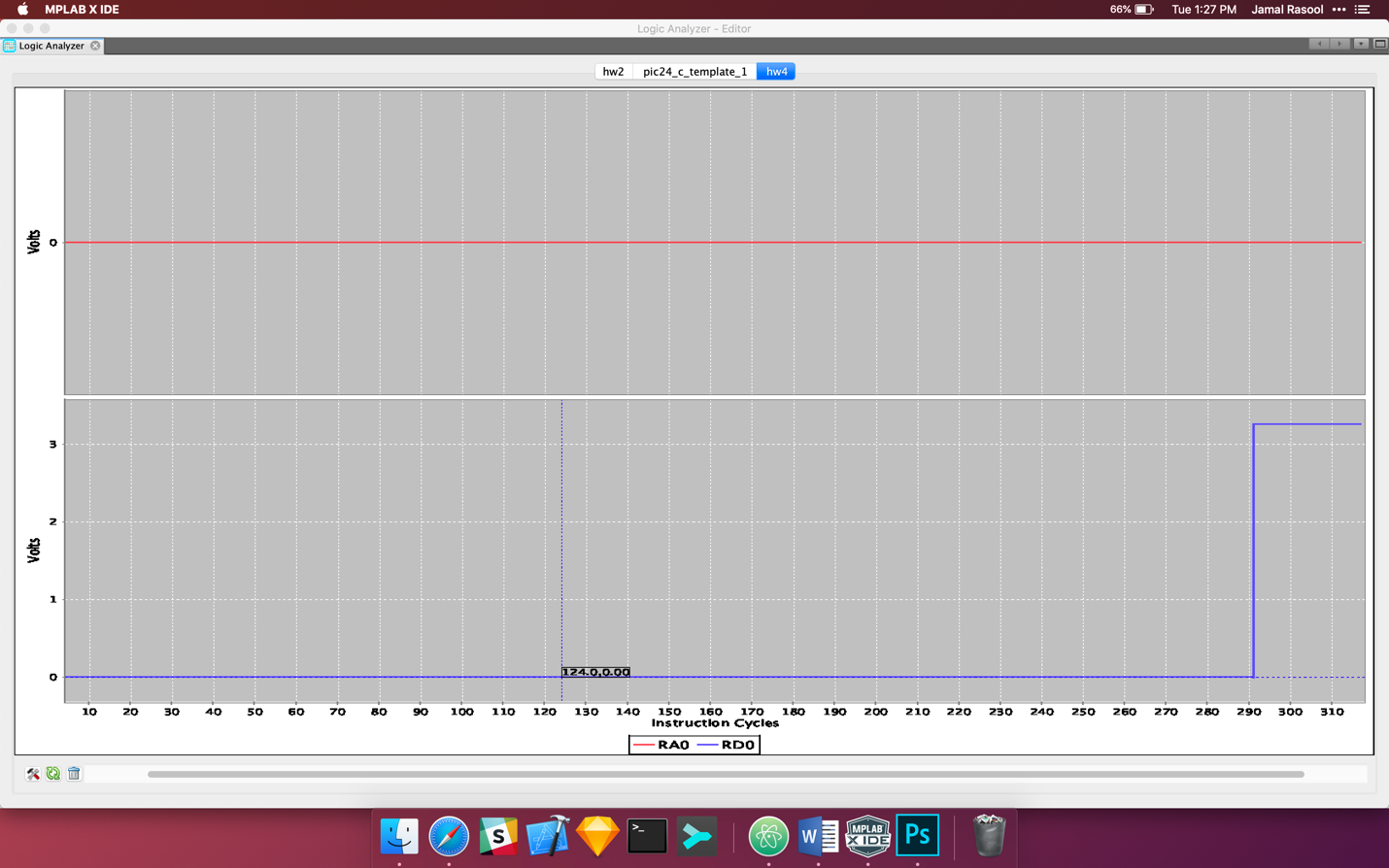
}

}

return 0;

}

(c) Show a screenshot of PWM signal in the Analyzer window and verify that the period and the duty are correct.



2. (25%)

Make a program to use the input interrupt to detect if a button is pressed with the PIC24E chip.

Assume the button is connected to Pin A0. If the button is pressed, turn the LED on. Otherwise, turn the LED off.

Use a state machine to handle the bouncing button issue. The state machine should toggle the LED only if the button changes its state for at least 10ms.

A part of code is in hw4.q2. Complete the code of led.c and button.c in hw4.q2.

1. Draw the diagram of the state machine.



(b) Complete the code of main.c that uses the state-centric approach to implement the diagram. Copy your code of main.c in your report.

#include <stdbool.h>

#include "Compiler.h"

#include "ConfigurationBits.h"

#include "led.h"

#include "delay.h"

#include "button.h"

#include "event.h"

#include "state.h"

#include "macros.h"

Event volatile evButton, evTimer;

int main() {

initLeds();

initButton();

evButton = NONE;

evTimer = NONE;

State st = S0\_Released;

while (1) {

pollButton();

switch (st) {

case S0\_Released:

//ledOff(1);

if (evButton == Button\_Pressed) {

ledOn(1);

startDelay();

st = S1\_Pressed;

}

break;

case S1\_Pressed:

//ledOn(1);

if (evTimer == Delay\_Expired) {

st = S2\_Pressed;

}

break;

case S2\_Pressed:

// ledOn(1);

if (evButton == Button\_Released) {

ledOff(1);

startDelay();

st = S3\_Released;

}

break;

case S3\_Released:

// ledOff(1);

if (evTimer == Delay\_Expired) {

st = S0\_Released;

}

break;

default:

// only for errors to reset to initial state

st = S0\_Released;

evButton = NONE;

evTimer = NONE;

}

}

return 0;

}

3. (30%)

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]}}

1. Assume in the system, the events happened in the following time order:

Follow the style in hw4.q3.docx, show the queues after the time points, T1, T2, T3, T4, T5.

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

(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++;

}

}

}

}

4. (15%)

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

1. How many cycles are between two interrupts?

frequency 40MHz

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

so frequency between interrupts = 40\*10^3Hz

Therefore, number of cycles between two interrupts

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

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

percentage of time for switching context = (20/1000) \* 100 = 2%

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

Total number of cycles = 20 + 380 = 400 cycles (without switching back to normal mode)

(400/1000) \* 100 = 40%

With switching back to normal it would be 42% as 20 cycles more will be required.