**ECE 414 – Embedded Systems**

**Report for Lab 04 – UART**

GuoYuan Li (Scribe)

Harry Zhu

Sep 30, 2020

**1. Introduction**

The objective of this lab is to learn the basics of UART by programming the UART in PIC32 microcontroller with blocking I/O. In addition, functions for configuring data transmission and receiving are also essential in this lab. Furthermore, learning to use the interrupt-driven “ztimer” module discussed in class to work with longer timing intervals than allowed by the “timer1” module. Finally, it is equally important to learn how design choices impact code performance.

**2. Requirements**

For this task you will create a module named uart1 that provides a clean interface for writing and reading characters to/from UART1.

1. The UART driver shall be named “uart1” and include a header file “uart1.h” and code file “uart1.c”.

2. The module shall include a function to initialize the UART to communicate using eight data bits and no parity bits at a specific baud rate: void uart1\_init(uint32\_t baudrate);

3. The module shall include a function that returns a true value when the UART transmitter is ready to accept a character for transmission: uint8\_t uart1\_txrdy();

4. The module shall include a function to write a character to the UART: void uart1\_txwrite(uint8\_t c);

5. The module shall include a function to write a null-terminated string to the UART: void uart1\_txwrite\_str(char \*c); This function must not overflow the transmitter – if the transmitter is not ready, it should suspend execution and wait until ready before sending more characters (this is known as blocking I/O).

6. The module shall include a function that returns true when the UART receiver has a character ready for reading: uint8\_t uart1\_rxrdy();

7. The module shall include a function that reads a received character from the UART. uint8\_t uart1\_rxread();

8. The module shall be tested by a unit test that performs the following tasks:

a. Print out a greeting message including the names of the lab partners.

b. Enter an infinite loop where it waits for a character to be received; reads the character, inverts the case of alphabetic characters (i.e. ‘A’->’a’ and viceversa) waits for the transmitter to be ready, and writes the character back to the transmitter.

c. Unit test shall be performed using input strings ranging in length from a single character up to 100 characters.

d. Unit test shall be performed using baud rates ranging from 1200-19200 baud.

9. Calculate the time taken for different operators for different data types and print the results to the terminal. Summarize results into the performance table. (Task 2)

**3. Architectural Description**

This section describes the planned architecture for a design, generally partitioned into a hardware architecture and a software architecture.

**3.1 Hardware Implementation**

No hardware implementation for this lab because it is for testing the UART on PIC 32 and only 3 pins for transmission(U1TX), receive(U1RX), and ground(GND) should be connected properly. For this lab, the PPS input pin selected for U1RX is RPA0 and the PPS output pin selected for U1Tx is RPA2. GND is connected to the 8th pin on the PIC 32.

**3.2 Software Implementation**

For both task1 and task2, the main software block consists of two modules: the UART1 module and a Test module. The terminal tool SerialTools on mac sends input characters for testing to UART1 and UART1 receives each character bit-by-bit serially. Once a full character is received, required operation is performed on the character by the Test module, in this case inverting the case of the alphabetic character while leaving the non-alphabetical characters unchanged, and it is transmitted to the UART1 module serially, or bit-by-bit. Meanwhile, UART1 is receiving another character from the terminal. For task2 exclusively, zTimer module is used to keep track of the time and another test module consists of different test functions for different operators and data types.

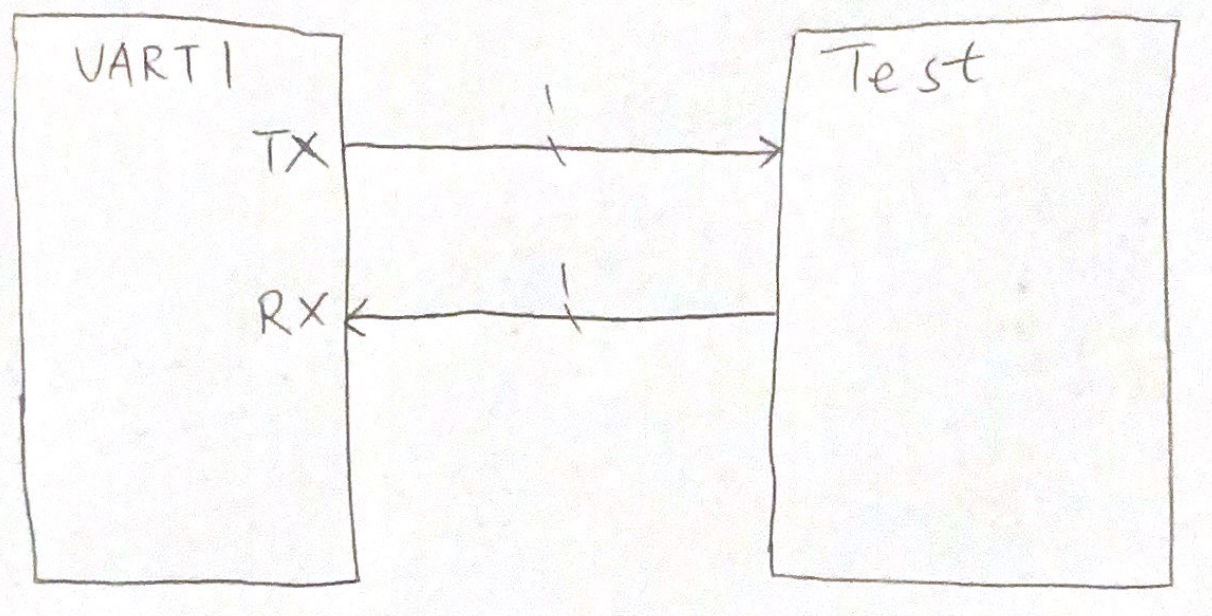


Figure 1. Task 1&2 Software High Level Block Diagram

**4. Test Plan**

**4.1 Unit Test of UART1**

**Unit test is performed by the test module to see if UART1 module behaves correctly. Pass/Fail criterions are included in the implementation plan**

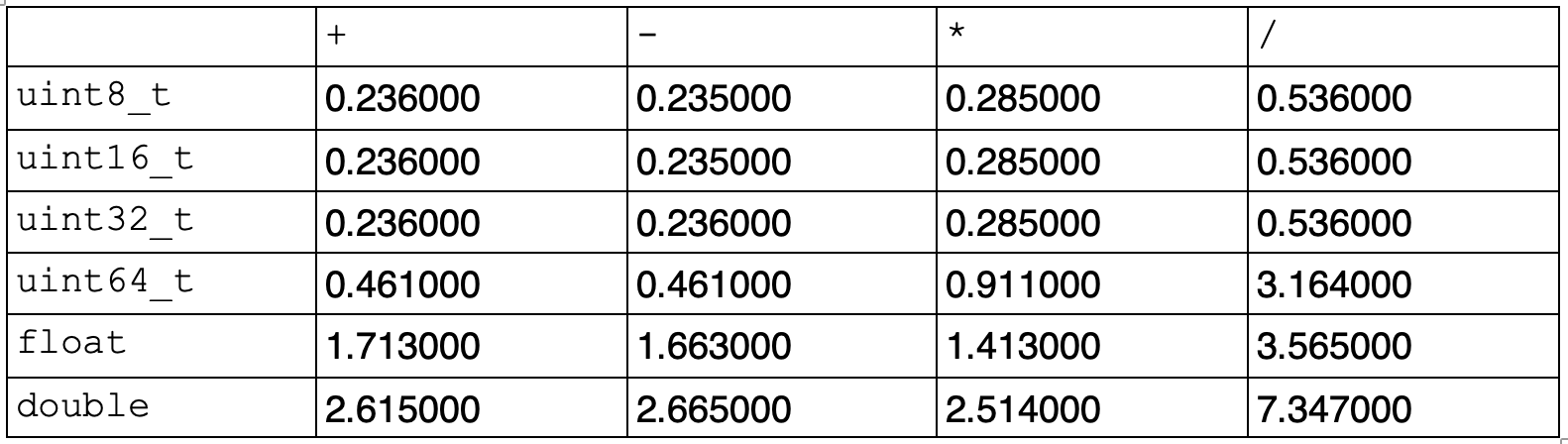
* Print out a greeting message including the names of the lab partners.
  + Result: **Pass**. Greeting messages are successfully printed.
* Enter an infinite loop where it waits for a character to be received; reads the character, inverts the case of alphabetic characters, waits for the transmitter to be ready, and writes the character back to the transmitter.
  + Result: **Pass**. Alphabetical characters have their cases inverted while non-alphabetical characters stay unchanged.
* input strings ranging in length from a single character up to 100 characters. (**Pass**)
* baud rates ranging from 1200-19,200 baud. (**Pass**)

**4.2 Acceptance Test**

* T1: set the baud rate to 9600 baud
  + Result: Baud rate is successfully set.
  + T2: Type ‘B’ on Test module for UART1 reception
  + Result: ‘B’ inverts to ‘b’.
  + T3: print the lab partners’ names
  + Result: lab partners’ names are printed along with the greeting message
  + T4. print other strings
  + Result: other strings are successfully printed
  + T5: type ‘h’ on the terminal
  + Result: ‘h’ is inverted to ‘H’
  + T6: type ‘$’ on the terminal
  + Result: ‘$’ is transmitted to the terminal without changing it
  + T7: calculate time for executing different operations for uint8\_t
  + Result: Operation time for +-\*/ are successfully calculated and printed to terminal
* T8: calculate time for executing different operations for uint16\_t
  + Result: Operation time for +-\*/ are successfully calculated and printed to terminal
* T9: calculate time for executing different operations for uint32\_t
  + Result: Operation time for +-\*/ are successfully calculated and printed to terminal
* T10: calculate time for executing different operations for uint64\_t
  + Result: Operation time for +-\*/ are successfully calculated and printed to terminal
* T11: calculate time for executing different operations for float
  + Result: Operation time for +-\*/ are successfully calculated and printed to terminal
* T12: calculate time for executing different operations for double
  + Result: Operation time for +-\*/ are successfully calculated and printed to terminal

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Requirements** | | | | | | | | | |
| **Tests (Performer)** | **2.1** | **2.2** | **2.3** | **2.4** | **2.5** | **2.6** | **2.7** | **2.8** | **2.9** | **Results** |
| **T1. GuoYuan** | **x** | **x** |  |  |  |  |  |  |  | **Pass** |
| **T2. GuoYuan** | **x** | **x** | **x** | **x** |  | **x** | **x** |  |  | **Pass** |
| **T3. GuoYuan** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **x** |  | **Pass** |
| **T4. GuoYuan** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **x** |  | **Pass** |
| **T5. GuoYuan** | **x** | **x** | **x** | **x** |  | **x** | **x** | **x** |  | **Pass** |
| **T6. GuoYuan** | **x** | **x** | **x** | **x** |  | **x** | **x** | **x** |  | **Pass** |
| **T7. Harry** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **Pass** |
| **T8. Harry** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **Pass** |
| **T9. Harry** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **Pass** |
| **T10. Harry** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **Pass** |
| **T11. Harry** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **Pass** |
| **T12. Harry** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **x** | **Pass** |

**Figure 2. Traceability Matrix**



**Figure 3. Performance Table (unit: micro seconds)**

**5. Conclusion**

The lab went successfully as all the tests were passed and all the requirements were satisfied. We’ve learned how to implement the UART interface in C and use the uart1 on the PIC 32 microcontroller for transmitting and receiving characters serially. In addition, we gained familiarity with the different amount of time taken for executing different operators for different data types. We spent approximately 3 hours in this lab.

**6. Software Implementation**

Main.c

#pragma config FNOSC = FRCPLL, POSCMOD = OFF

#pragma config FPLLIDIV = DIV\_2, FPLLMUL = MUL\_20

#pragma config FPBDIV = DIV\_1, FPLLODIV = DIV\_2

#pragma config FWDTEN = OFF, JTAGEN = OFF, FSOSCEN = OFF

#include <xc.h>

#include <inttypes.h>

#include "uart1.h"

#include <stdio.h>

#include "ztimer.h"

#include "test.h"

#define NUM\_ITERATIONS 100000

#define NUM\_REPS 10

const uint32\_t baudrate = 9600;

char c;

char str[] = "Hello: Lab Partners GuoYuan and Harry!\r\n";

uint8\_t buffer[64];

void task1()

{

uart1\_init(baudrate);

uart1\_txwrite\_str(str);

while(1){

c = uart1\_rxread();

if((c >='a' && c<='z') || (c >= 'A' && c<= 'Z')){

if(c > 'Z'){

c -= 0x20;

}else{

c += 0x20;

}

}

uart1\_txwrite\_str('\r\n');

uart1\_txwrite(c);

uart1\_txwrite('\r\n');

}

}

void task2()

{

uint8\_t j, k;

uint32\_t t1, t2;

// !!! Add code: Initialize modules and turn on interrupts

uart1\_init(baudrate);

zTimerOn();

zTimerSet(1);

uart1\_txwrite\_str("Performance Summary: Time per operation statistics\r\n");

// Test multiplying bytes

for(j=0; j<4; j++){

t1 = zTimerReadms();

operation\_d(j,NUM\_ITERATIONS);

t2 = zTimerReadms();

sprintf(buffer, "task double: %.06f us per operation\r\n",(double)(t2-t1)/ (double)NUM\_ITERATIONS /(double)NUM\_REPS\*1000.0);

uart1\_txwrite\_str(buffer);

}

while (1);

}

int main()

{

//task1();

task2();

}

uart1.c

#include <xc.h>

#include <inttypes.h>

uint8\_t T;

const uint32\_t freq = 40000000; //PBCLK frequency

void uart1\_init(uint32\_t baudrate)

{

ANSELA = 0; // turn portA pin analog off

TRISA = 0x04; // turn portA pin digital input on

U1RXR = 0x0000; //connect RPA2 to U1RX

RPA0R = 0x0001; //connect U1TX to RPA0

U1BRG = 259; //(freq/(16\*baudrate))-1; //set baudrate

U1STA = 0x1400; // enable transmitter and receiver on UART1, 5120

U1MODE = 0x8000; // turn on the UART

}

uint8\_t uart1\_txrdy()

{

return !U1STAbits.UTXBF;

}

void uart1\_txwrite(uint8\_t c)

{

U1TXREG = c;

}

void uart1\_txwrite\_str(char \*c)

{

while(\*c != '\0'){

while(!uart1\_txrdy()){}

uart1\_txwrite(\*c);

c++;

}

}

uint8\_t uart1\_rxrdy()

{

return U1STAbits.URXDA;

}

uint8\_t uart1\_rxread()

{

while(!uart1\_rxrdy()) {}

return U1RXREG;

}

Uart1.h

#ifndef UART1\_H

#define UART1\_H

#include <xc.h>

#include <inttypes.h>

extern void uart1\_init(uint32\_t baudrate);

extern uint8\_t uart1\_txrdy();

extern void uart1\_txwrite(uint8\_t c);

extern void uart1\_txwrite\_str(char \*c);

extern uint8\_t uart1\_rxrdy();

extern uint8\_t uart1\_rxread();

#endif

Test.c (Task 2)

#include <stdio.h>

#include <inttypes.h>

#include "test.h"

uint8\_t a = 15, b = 26 , c = 0;

/\*\*

\* operation:

\* 0: add

\* 1: sub

\* 2: mult

\* 3: divide

\*/

void operation\_8(uint8\_t operation, uint32\_t iterations)

{

uint32\_t i;

uint8\_t i1, i2, i3;

i1 = a;

i2 = b;

i3 = c;

switch(operation){

case 0:

for (i=0; i<iterations; i++){

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

}

break;

case 1:

for (i=0; i<iterations; i++){

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

}

break;

case 2:

for (i=0; i<iterations; i++){

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

}

break;

case 3:

for (i=0; i<iterations; i++){

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

}

break;

default:

break;

}

}

void operation\_16(uint8\_t operation, uint32\_t iterations)

{

uint32\_t i;

uint16\_t i1, i2, i3;

i1 = (uint16\_t)a;

i2 = (uint16\_t)b;

i3 = (uint16\_t)c;

switch(operation){

case 0:

for (i=0; i<iterations; i++){

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

}

break;

case 1:

for (i=0; i<iterations; i++){

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

}

break;

case 2:

for (i=0; i<iterations; i++){

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

}

break;

case 3:

for (i=0; i<iterations; i++){

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

}

break;

default:

break;

}

}

void operation\_32(uint8\_t operation, uint32\_t iterations)

{

uint32\_t i;

uint32\_t i1, i2, i3;

i1 = (uint32\_t)a;

i2 = (uint32\_t)b;

i3 = (uint32\_t)c;

switch(operation){

case 0:

for (i=0; i<iterations; i++){

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

}

break;

case 1:

for (i=0; i<iterations; i++){

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

}

break;

case 2:

for (i=0; i<iterations; i++){

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

}

break;

case 3:

for (i=0; i<iterations; i++){

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

}

break;

default:

break;

}

}

void operation\_64(uint8\_t operation, uint32\_t iterations)

{

uint32\_t i;

uint64\_t i1, i2, i3;

i1 = (uint64\_t)a;

i2 = (uint64\_t)b;

i3 = (uint64\_t)c;

switch(operation){

case 0:

for (i=0; i<iterations; i++){

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

}

break;

case 1:

for (i=0; i<iterations; i++){

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

}

break;

case 2:

for (i=0; i<iterations; i++){

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

}

break;

case 3:

for (i=0; i<iterations; i++){

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

}

break;

default:

break;

}

}

void operation\_f(uint8\_t operation, uint32\_t iterations)

{

uint32\_t i;

float i1, i2, i3;

i1 = (float)a;

i2 = (float)b;

i3 = (float)c;

switch(operation){

case 0:

for (i=0; i<iterations; i++){

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

}

break;

case 1:

for (i=0; i<iterations; i++){

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

}

break;

case 2:

for (i=0; i<iterations; i++){

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

}

break;

case 3:

for (i=0; i<iterations; i++){

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

}

break;

default:

break;

}

}

void operation\_d(uint8\_t operation, uint32\_t iterations)

{

uint32\_t i;

double i1, i2, i3;

i1 = (double)a;

i2 = (double)b;

i3 = (double)c;

switch(operation){

case 0:

for (i=0; i<iterations; i++){

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

i3 = i2 + i1;

}

break;

case 1:

for (i=0; i<iterations; i++){

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

i3 = i2 - i1;

}

break;

case 2:

for (i=0; i<iterations; i++){

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

i3 = (i2) \* (i1);

}

break;

case 3:

for (i=0; i<iterations; i++){

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

i3 = (i2) / (i1);

}

break;

default:

break;

}

}

Test.h (Task2)

#ifndef TEST.H

#define TEST.H

#include <xc.h>

#include <inttypes.h>

void operation\_8(uint8\_t operation, uint32\_t iterations);

void operation\_16(uint8\_t operation, uint32\_t iterations);

void operation\_32(uint8\_t operation, uint32\_t iterations);

void operation\_64(uint8\_t operation, uint32\_t iterations);

void operation\_f(uint8\_t operation, uint32\_t iterations);

void operation\_d(uint8\_t operation, uint32\_t iterations);

#endif

zTimer.c (task2)

#include <xc.h>

#include <plib.h>

#include "ztimer.h"

static uint32\_t count, elapsedCount;

static uint32\_t period, targetCount;

static uint8\_t zTimerFlag = 0;

#pragma interrupt InterruptHandler ipl1 vector 0

void InterruptHandler(void)

{

// add code to set flag when we reach target

count++;

elapsedCount++;

if (count == period) {

zTimerFlag = 1;

count = 0;

}

mT2ClearIntFlag();

} // Interrupt Handler

// initialize timer to set a flag every given period (in ms)

void zTimerSet(uint32\_t pdms) {

period = pdms;

}

// enable the timer and turn on the interrupt

void zTimerOn() {

// 1. initialize timer

PR2 = 4999; // Timer2 periodic reset every 5000 ticks (1ms)

T2CON = 0x8030; // Timer2 scaling factor of 8 (200ns)

TMR2 = 0; // start out at 0

count = 0;

elapsedCount = 0;

zTimerFlag = 0;

// 2. initialize interrupts

mT2SetIntPriority(1);

INTEnableSystemSingleVectoredInt();

mT2IntEnable(1);

}

void zTimerOff() {

mT2IntEnable(0);

T2CON = 0x0; // turn off timer

}

// read and return the timer flag value

// SIDE EFFECT: clear the flag

uint8\_t zTimerReadFlag() {

if (zTimerFlag) {

zTimerFlag = 0;

return 1;

} else return 0;

}

// return elapsed time in milliseconds since last call to zTimerOn

uint32\_t zTimerReadms() {

return elapsedCount;

}

zTimer.h (Task 2)

#ifndef ZTIMER\_H

#define ZTIMER\_H

#include <xc.h>

#include <inttypes.h>

#include <plib.h>

extern void zTimerSet();

extern void zTimerOn();

extern uint8\_t zReadTimerFlag();

#endif