Quang Nguyen

CPE 409 Lab 10

# Goals

* To understand and use an RTOS

# Equipment used

## Hardware

* Microchip Explorer 16 board
* PIC kit 3

## Software

* MPLAB X IDE 2.00

# Design Specifications

* The program must use FreeRTOS
* The program must use the three provided tasks:
  + static void ButtonPushCounter( void \*pvParameters)
    - This task must scan for the user’s input of a button press on RD6 (S3) and RD7 (S6) every 10 ms.
    - When RD6 is pressed:
      * The task must increment a 16 bits counter and load the queue for writeLCD task
        + The content of the queue for the writeLCD task must take the form of “Count = XXXXX”
        + XXXXX – the value for of the counter in decimal
    - When RD7 is pressed:
      * The task must throw a flag indicating that the time of the RTC must be display on the LCD
    - Once any of these buttons are pressed:
      * The task must change the scan period from 10 ms to 100 ms.
      * The task must keep this 100 ms period until the button is released.
      * Once the button is released, the task will switch back to a 10 ms scan period.
  + void RTCTimer(void \*pvParameters)
    - This task must run every 0.5 second
    - Once the flag indicating that the time needs to be printed on the LCD, the task will load the queue for the writeLCD task with the printout for the time.
      * The print out for the LCD must take the form of “RTC HH:MM:SS.Z”
        + Z indicates the half second resolution
        + SS indicates the value of seconds
        + MM indicates the value of minutes
        + HH indicates the value of hours
        + The indicated time must be in military time.
  + void WriteLCD(void \*pvParameters)
    - This task must print the content of the queue onto the LCD
      * It must print one character at a time.
      * Upon completion, it will block itself until the next message comes in.

# Design

* Refer to Figure 1 for the block diagram of the ButtonPushCounter task

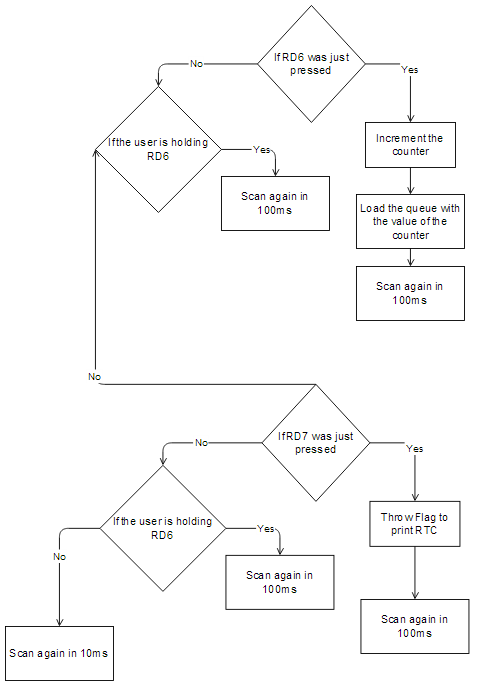


Figure 1: Block diagram for ButtonPushCounter task

* Refer to Figure 2 for the block diagram of the RTCTimer task

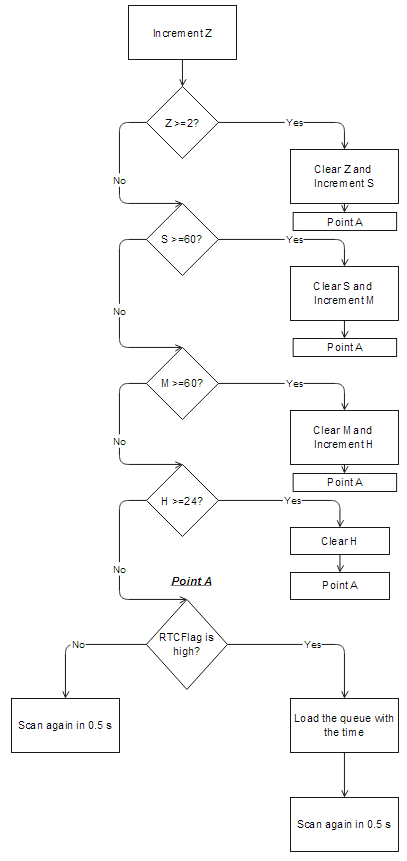


Figure 2: Block diagram for RTCTimer task

* Refer to Figure 3 for the block diagram of the WriteLCD task

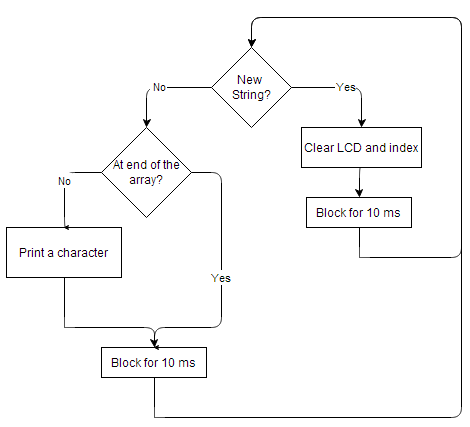


Figure 3: Block diagram for WriteLCD task

* See attached zip file for the code of the design

# Verification

The program was first tested by pressing RD6. After 100 presses, the LCD did indeed display “Count = 00100”, which confirm that the counter portion of the program is working. The program was then tested by pressing RD7. Once pressed, the LCD did indeed switched to the display for the RTC. RD6 was then pressed to confirm that the LCD display switches back to the counter display. The program was then allow to run for over 1 hour. Once this 1 hour was up, it was confirmed that the LCD output for the RTC display does display the correct value.

# Conclusions and Limitations

* The program worked as it should and I did not see any limitation to the program

# Programming Code

/\* Standard includes. \*/

#include <stdio.h>

/\* Scheduler includes. \*/

#include "FreeRTOS.h"

#include "task.h"

#include "queue.h"

#include "croutine.h"

#include <portmacro.h>

/\* Demo application includes. \*/

#include "BlockQ.h"

#include "crflash.h"

#include "blocktim.h"

#include "integer.h"

#include "comtest2.h"

#include "partest.h"

#include "lcd.h"

#include "timertest.h"

#define TRUE 1

#define FALSE 0

/\*-----------------------------------------------------------\*/

//\*\*\*\*\*\*\*\* configuration bits

\_FGS(GSS\_OFF & GWRP\_OFF);

\_FOSCSEL(FNOSC\_PRIPLL & IESO\_ON);

\_FOSC(FCKSM\_CSDCMD & POSCMD\_XT & OSCIOFNC\_OFF);

\_FWDT(FWDTEN\_OFF);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

\* prototypes

\*/

static void prvSetupHardware( void );

char IntToChar(unsigned int Digit);

static void ButtonPushCounter( void \*pvParameters);

void RTCTimer(void \*pvParameters);

void WriteLCD (void \*pvParameters);

void vApplicationIdleHook(void);

void displayCounter(int Count, char \*LCD);

void digitCheck(int \*FirstDigit, int \*SecondDigit);

void BinaryToDecimal ( int Value, char \*TenthPlace, char \*OnePlace);

/\*-----------------------------------------------------------\*/

/\*------------------GLOBAL-----------------------------------------\*/

xQueueHandle LCDDisplayinfo;

char RTCFlag; //This global variable can be used to signal that the button to display the RTC

//has been pressed. Ideally we could have done this with a semiphore.

/\*

\* Create the demo tasks then start the scheduler.

\*/

int main( void )

{

/\* Configure any hardware required for this demo. \*/

PLLFBD = 0x001E; //external osc is 8MHz using defaults Fosc=8MHZ /8 (0x1E+2)=32 MHz

//RTOS setup of hardware

prvSetupHardware();

Init\_LCD();//Initialize the LCD this must be done first before the LCD is to be used.

while (check\_busy())

{

}

home\_clr(); // start display at top line left most character

/\* Create Tasks\*/

xTaskCreate(ButtonPushCounter,"Task Read Swtchs", 200, NULL, 2, NULL);

xTaskCreate(RTCTimer, "Task RTC", 200, NULL, 2, NULL);

xTaskCreate(WriteLCD, "Write LCD",200, NULL, 1, NULL); //notice that this task has lower priority.

/\* create the queue for the string to be displayed. Length = 2 each chunk holds 16 char string\*/

LCDDisplayinfo = xQueueCreate(2,16);

/\* start the scheduler. \*/

vTaskStartScheduler();

/\* Will only reach here if there is insufficient heap available to start

the scheduler. \*/

for( ;; );

return 0;

}

/\*-----------------------------------------------------------\*/

//This is required by the RTOS, DO NOT MOTIFY

static void prvSetupHardware( void )

{

vParTestInitialise();

}

/\*-----------------------------------------------------------\*/

static void ButtonPushCounter( void \*pvParameters)

{

/\* This task will read the pushbuttons, since most pushes of the button will take more than

10 ms it needs to only run every 10 ms or l00ms if in the middle of a button push read.

The API for a delay is: vTaskDelay(10); to delay for 10ms before being active again \*/

char LCDDisplay[17] = "Count = 00000 ";

int Rd6ChangeStateFlag = 0;

int Rd7ChangeStateFlag = 0;

unsigned int Rd6Counter = 0;

int BlockingFlag = 0; // 0 = 10 ms

// 1 = 100 ms

xQueueSendToBack (LCDDisplayinfo, LCDDisplay, 0);

xQueueSendToBack (LCDDisplayinfo, LCDDisplay, 0);

/\* start endless loop\*/

for( ;; )

{

// if RD6(S3) is pressed

// Display count on LCD

if ( (\_RD6 == 0) && (Rd6ChangeStateFlag == 0))

{

RTCFlag = 0;

Rd6ChangeStateFlag = 1;

//Increment Counter

Rd6Counter++;

// Load the LCD with the count display

displayCounter(Rd6Counter, LCDDisplay);

//Put the updated count into the queue

xQueueSendToBack (LCDDisplayinfo, LCDDisplay, 100);

//Scan again every 100 ms

BlockingFlag = 1;

}

// If the user just released the button

else if ( (\_RD6 == 1) && (Rd6ChangeStateFlag == 1))

{

Rd6ChangeStateFlag = 0;

BlockingFlag = 0;

}

// if RD7(S6) is pressed

// Display RTC time on LCD

if ( (\_RD7 == 0) && (Rd7ChangeStateFlag == 0))

{

RTCFlag = 1;

Rd7ChangeStateFlag = 1;

//Scan again every 100 ms

BlockingFlag = 1;

}

// If the user just released the button

else if ( (\_RD7 == 1) && (Rd7ChangeStateFlag == 1))

{

Rd7ChangeStateFlag = 0;

BlockingFlag = 0;

}

if (BlockingFlag == 0)

vTaskDelay(10);

// vTaskDelay( 100 / portTICK\_RATE\_MS);

else

vTaskDelay(100);

// vTaskDelay( 1000 / portTICK\_RATE\_MS);

} // end of for loop

}

void WriteLCD(void \*pvParameters)

{

// This is lower priority task it will write charaters to the LCD screen in response to the button

//pushes an example API of reading the queue is given

int LCDCounter = 0;

portBASE\_TYPE NewString;

char LCDDisplayString[16];

for ( ; ; )

{

NewString = xQueueReceive( LCDDisplayinfo,LCDDisplayString,10);

// If we recieve a new message.

// Need to bring the display back to the beginning and start displayin the new data

if (NewString == pdPASS)

{// get new string and start displaying

LCDCounter = 0;

home\_it();

vTaskDelay( 10 / portTICK\_RATE\_MS);

}

// if not, we are going to print one character at a time

// We are going to wait for a new queque once we reached the end

// of the string

else

{

//Check if busy

if (check\_busy() == 0)

{

// if LCDCounter is not at the end of the string

if (LCDCounter < 17)

{

// Printer Character on LCD

lcd\_data( LCDDisplayString[LCDCounter] );

LCDCounter++;

}

vTaskDelay( 10 / portTICK\_RATE\_MS);

}

}

}//end of for loop

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void RTCTimer(void \*pvParameters)

{ //This is where you place your code to keep track of the RTC and the display that it will show.

//Initialize and declare

/\* In order to get true timing you should use the periodic delay function to set this up

some code is given... see lecture 17 for details

in addition you will want to write a string to queue for either RTC display or count display

sample API function call is given to do this. \*/

portTickType xLastWakeTime;

xLastWakeTime = xTaskGetTickCount();

char LCDDisplay[16];

int Z = 0, S = 0, M = 0, H = 0;

for( ;; )

{

// Z counts in half a second increment.

// If Z overflowed

Z++;

if ( Z == 2)

{

Z = 0;

S++;

}

// If S overflowed

if ( S > 60)

{

S = 0;

M++;

}

// if M overflowed

if ( M > 60)

{

M = 0;

H++;

}

// if H overflowed

if ( H > 24)

{

H = 0;

}

// Load the LCDDispaly array with the time if the RTCFlag is HIGH

if (RTCFlag == 1)

{

char Temp[5] = "RTC ";

int Counter;

for (Counter =0 ; Counter <= 3; Counter++)

{

LCDDisplay[Counter] = Temp[Counter];

}

BinaryToDecimal( H, &LCDDisplay[4], &LCDDisplay[5]);

LCDDisplay[6] = ':';

BinaryToDecimal( M, &LCDDisplay[7], &LCDDisplay[8]);

LCDDisplay[9] = ':';

BinaryToDecimal( S, &LCDDisplay[10], &LCDDisplay[11]);

LCDDisplay[12] = '.';

if ( Z == 1)

LCDDisplay[13] = '5';

else LCDDisplay[13] = '0';

xQueueSendToBack(LCDDisplayinfo, LCDDisplay, 0);

}

vTaskDelayUntil(&xLastWakeTime,500); // wait 0.5 seconds

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void vApplicationIdleHook( void)

{

//this is a function that should return quickly and will run as part of the idle task.

// You don't need to put any code in here you can leave as is.

// this function returns so if you want a variabe to be

//non volatile declare it static.

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Name: displayCounter

\* Parameters:

\* int Count

\* Return: void

\* Pins used: None

\* Global Var used:

\* char LCDDisplay[17];

\* Function:

\* Similiar to the binaryToDecimal Function.

\* This function will take the value of count and convert

\* it to decimal.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void displayCounter(int Count, char \*LCD)

{

//Goal: to display "Count = XXXXX "

char TempChar[9] = "Count = ";

int Counter;

for (Counter =0 ; Counter < 8; Counter++)

{

LCD[Counter] = TempChar[Counter];

}

int Ten = 0, Hundred = 0, Thousand = 0, TenThousand = 0;

if (Count > 9)

{

while(Count > 9)

{

Count -= 10;

Ten++;

digitCheck(&Ten, &Hundred);

digitCheck(&Hundred, &Thousand);

digitCheck(&Thousand, &TenThousand);

}

}

LCD[12] = Count + 48;

LCD[11] = Ten + 48;

LCD[10] = Hundred + 48;

LCD[9] = Thousand + 48;

LCD[8] = TenThousand + 48;

LCD[13] = ' ';

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Name: digitCheck

\* Parameters:

\* int \*FirstDigit

\* int \*SecondDigit

\* Return:

\* void

\* Pins used: None

\* Global Var used:

\* Function:

\* This function will look at the pointer for the first digit

\* and increment the second digit if it is greater than 9.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void digitCheck(int \*FirstDigit, int \*SecondDigit)

{

if ((\*FirstDigit) > 9)

{

\*FirstDigit -= 10;

(\*SecondDigit)++;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Name: BinaryToDecimal

\* Parameters:

\* int value : The value of time

\* char \*TenthPlace : Pointer to the variable for the tenth place

\* char \*OnePlace : Pointer to the variable for the one place

\* Return:

\* void

\* Pins used: None

\* Function:

\* Function will convert will convert a binary value to decimal value

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void BinaryToDecimal ( int Value, char \*TenthPlace, char \*OnePlace)

{

int Temp = 0;

if (Value > 9)

{

while(Value > 9)

{

Value -= 10;

Temp++;

}

}

// Assign value to the variable and convert it to ASCII

\*TenthPlace = 48 + Temp;

\*OnePlace = 48 + Value;

}