Course: **ENGI30000 Embedded Systems Applications**

Project #: **4**

Project Title: **RTOS based project execution**

Professor: **MD – Nazrul Khan**

|  |  |
| --- | --- |
| Marking Scheme | Marks |
| Demonstration | 3 |
| Coding | 3 |
| Report Content | 4 |
|  |  |

|  |  |  |
| --- | --- | --- |
| Submitted by: |  | |
|  | 1):  2): | Rishab Singh  -NA- |

Date of Submission: December 6, 2022

Notes:

Name and Objective (in short):

The name if the project is “RTOS based application”, and the objective of this project is to make the system/application capable of responding in “supposedly (Nothing is instant)” real time scenario and reconstruct the execution of Project 1,2 and 3.

Schematic:

Diagram, schematic

Description automatically generated

Figure : UART

Diagram

Description automatically generated

Figure : Temp sensor

Diagram, schematic

Description automatically generated

Figure : Potentiometer sensing

Diagram, schematic

Description automatically generated

Figure : Oscillator

Graphical user interface, application, table, Excel

Description automatically generated

Figure : T1CON Register

Diagram

Description automatically generatedConfiguration

Figure :LED Diagram

**For Timer**

Text

Description automatically generated

Figure : Timer configuration

We are following Figure.4 reference from the PIC24 datasheet to configure the Timer1 and Interrupt properties.

We set the value of TON = 1;

We have also selected out Prescaler as 1 so our internal frequency would be divided by 2 and by 1 -> from the prescaler.

TCS = 0 as we are using the internal clock.

In the Timer1 configuration, the code is as below:

void TIMER\_Initialize(void)

{

TMR1 = 0x0000;

//Instructions from Manual

// 1. Set the TON bit (= 1).

/\*\*

\* Timer setting done at the last

\*/

// 2. Select the timer pre scaler ratio using the

T1CONbits.TCKPS = 0b00; //1:1 ratio selected

/\*\*

\* We are using the lowest possible pre scaler to get

\* a more precis time interval facilitating more control

\* over the time

\*/

// 3. Set the Clock and Gating modes using the TCS

T1CONbits.TCS = 0; //using internal clock so TCS=0

T1CONbits.TGATE = 0; //TGATE is set to 0 as we want the output from the PreScaler to generate out interrupts

// 4. Set or clear the TSYNC bit to configure synchronous or asynchronous operation.

T1CONbits.T1SYNC = 1; //turning on synchronization of the clock

// 5. Load the timer period value into the PR1

PR1 = (uint16\_t) 0x0078;

IFS0bits.T1IF = 0;

// Interrupt priority setting

// 7 = 0b0111

IPC0bits.T1IP2 = 1;

IPC0bits.T1IP1 = 1;

IPC0bits.T1IP0 = 1;

IEC0bits.T1IE = 1;

T1CONbits.TON = 1;

// Ans: Check the Interrupt\_Initialize for more details

}

**For UART2 and ADC**

For UART2:

* We set the value of U2Mode register to enable the UART (bit15) and set the parity bit (bit1 and bit2) as well as bit13 has been set to 0 so BRG=0 hence.
* Then we set the U2BRG register to get the desired Baud Rate of 9600.

void UART2\_Initialize (void)

{

/\*\*

\* Enable the UART (bit 15) and

\* set parity bit to "even parity"

\* (bit 2=0, bit 1=1)

\*/

U2MODE = 0x8002;

U2STA = 0x0000; //resetting U2STA register

/\*\*

\* Clearing the U2 transmit register (<- lower half of the register)

\* to prepare for transmission

\*/

U2TXREG = 0x0000;

/\*\*

\* Baud Rate = 9600 = 0x0019;

\* Frequency = 4 MHz;

\*/

U2BRG = 0x0019;

/\*\*

\* Enabling Rx Interrupt

\* to detect receiving information

\*/

IEC1bits.U2RXIE = 1;

/\*\*

\* Enabling transmission by

\* setting the enable transmit bit = 1

\*/

U2STAbits.UTXEN = 1;

/\*\*

\* We don't need Tx Interrupts so

\* we disable it

\*/

IEC1bits.U2TXIE = 0; //Tx interrupt disabled

/\*\*

\* Setting the interrupt flag bits for

\* Rx and Tx interrupts

\*/

IFS1bits.U2RXIF = 0;

IFS1bits.U2TXIF = 0;

/\*\*

\* Setting the priority of

\* the Rx interrupts to 5

\*/

IPC7bits.U2RXIP2 = 1;

IPC7bits.U2RXIP1 = 0;

IPC7bits.U2RXIP0 = 1;

}

For ADC:

* In the ADC\_Initialize() function, we set the **AD1PCFG** to 0xFFCF to set the pin4 and pin5 as Analog inputs (1 for temperature and 1 for pot voltage).
* **AD1CHS** value has to be changed to get the value of pin4 and pin5 as analog input so we set the value of the 0x0004 and 0x0005 as needed.
* **AD1CON1bits.ADON** has to be set to 1 to turn on the Analog to Digital Converter.
* **AD1CON3** has been set to 0x1100 so ADCS is set to 0 and the SAMC has been set to 0x0001.

**For RTOS**

We have created 4 Tasks for the RTOS and 2 Semaphores to establish inter-task communication. The Task TCBs, Priorities and Semaphores are as shown in the figure below:

Table

Description automatically generated

The Tasks are created in the main function as shown in the figure below

Text

Description automatically generated

The semaphores and the scheduler as mentioned as below:

Text

Description automatically generated

Apart from this the header file has to be updated, the OSTASKS have to be defined as 4 as we have declared 4 tasks for our RTOS application.

**Calculations (Current/voltage/resistance/Time/Frequency/ etc.,):**

For the timer, we want to RUN LED Task (TASK\_RUN\_LED) to be executed every 3 seconds.

Hence,

The given internal frequency is 8MHz.

As this frequency passes through the pre-scaler, the frequency becomes 4MHz as the pre-scaler is selected to be in the ratio of 1:1 (and divided by 2).

Note: Pre-scaler value ratio is selected 1:1 as the higher the frequency we have, we will have a smaller time unit and the smaller the time unit, we would have much more precis and accurate output.

Hence for time unit (t),

Now, we need the interrupt to call the ISR every 3 s and the relation between the required time, period register and time unit is as mentioned below.

This implies,

**We also have to consider the fact that the RTOS tick rate is also functional and based on this timer interrupt would generate a tick rate so the above formula could be modified as,**

Here,

**30us is selected instead of 3s as buffer\_variable will take multiplier load (buffer\_variable = 100,000) and the tick rate is set to 30us – All using one timer only.**

### Hence,

NOTE: The datatype of buffer\_variable would have to be changed to **unsigned long int** instead of normal **int**.

As normal **int** can only store upto approx. 32000 while an **unsigned long int** can be stored upto approx. 4.2 million which can easily store 100,000.

**CALCULATIONS FOR ADC & UART:**

Converting physical quantities,

In Code:-

Text

Description automatically generated

**Resolution of the Device**

The resolution of this device is

Resolution (R) =

For 10-bit ADC,

Resolution (R) = = 3.22 mV

**UBRGX**

The BRGH value is set to 0.

**TAD, Auto sampling time and Conversion Time**

As ADCS has been set to zero,

As SAMC (bit8-bit12) has been set to 0001,

Critical Observation/Thinking (One paragraph):

**A piece of paper with writing on it

Description automatically generated**

Figure : Logical Diagram for Critical thinking

To make this project functional, we have to created 4 tasks,

* **Task 1->** To read the temp & pot value and signal the beginning of the process to Task 2 & Task 3 and yield back to the processor to allow other tasks to execute.
* **Task 2->** The Knight Raider pattern task waits for a semaphore from Task 1 to begin execution. The values of count1 and count2 are deliberately kept low to allow faster context switching between the tasks without unnecessary delay.
* **Task 3->** The temp & pot print task waits for a semaphore from Task 1 but will execute only after Task 2 due to logical sequence. This task is created separately because although it is an important task, but printing of the variables can wait as knight rider should run first so the printing waits till task 2 is complete.
* **Task 4->** The run lead will execute after all the three tasks and has the lowest priority as interrupts at the end this will call the OS\_Delay service to release the processor.

**IMPORTANT: The tickrate and the Run LED execution time has been configured to work hand-in-hand in Timer1 interrupt please see the configuration and calculations for a detailed overview of the timer calculations.**

We configure the required ADC and UART in the ADC\_Initialize() and UART\_Initalize() functions that configure the respective UART and ADC modules for reading and printing the temp and pot values via UART.

CODE:

//SmitChawda//12062022

//RTOS Project Template

#include <xc.h>

#include <salvo.h>

#include <stdint.h>

#include <stdbool.h>

#define \_XTAL\_FREQ 8000000UL

// CONFIG2 WORD

#pragma config POSCMOD = XT // Primary Oscillator Select->XT Oscillator mode selected

#pragma config OSCIOFNC = OFF // Primary Oscillator Output Function->OSC2/CLKO/RC15 functions as CLKO (FOSC/2)

#pragma config FCKSM = CSDCMD // Clock Switching and Monitor->Clock switching and Fail-Safe Clock Monitor are disabled

#pragma config FNOSC = PRI // Oscillator Select->Primary Oscillator (XT, HS, EC)

#pragma config IESO = ON // Internal External Switch Over Mode->IESO mode (Two-Speed Start-up) enabled

// CONFIG1 WORD

#pragma config WDTPS = PS32768 // Watchdog Timer Postscaler->1:32768

#pragma config FWPSA = PR128 // WDT Prescaler->Prescaler ratio of 1:128

#pragma config WINDIS = ON // Watchdog Timer Window->Standard Watchdog Timer enabled,(Windowed-mode is disabled)

#pragma config FWDTEN = OFF // Watchdog Timer Enable->Watchdog Timer is disabled

#pragma config ICS = PGx2 // Comm Channel Select->Emulator/debugger uses EMUC2/EMUD2

#pragma config BKBUG = OFF // Background Debug->Device resets into Operational mode

#pragma config GWRP = OFF // General Code Segment Write Protect->Writes to program memory are allowed

#pragma config GCP = OFF // General Code Segment Code Protect->Code protection is disabled

#pragma config JTAGEN = OFF // JTAG Port Enable->JTAG port is disabled

////////////////////////////////////////////////////////////////////////////////

//Function Proto type

void SYSTEM\_Initialize(void);

void OSCILLATOR\_Initialize(void);

void PIN\_MANAGER\_Initialize(void);

void TMR1\_Initialize (void);

void UART2\_Initialize (void);

void ADC\_Initialize (void);

void delayFunc(void);

void U2\_TxByte(char value);

void U2\_TxString(char \*s);

//Interrupt Prototype

void \_\_attribute\_\_ ((interrupt, no\_auto\_psv)) \_T1Interrupt(void);

void \_\_attribute\_\_ ((interrupt, no\_auto\_psv)) \_U2RXInterrupt(void);

////////////////////////////////////////////////////////////////////////////////

//Code variables

unsigned int counter = 0;

unsigned int count1 = 200;

unsigned int count2 = 200;

int rcvByte; //we dont need this for this program - as we are not receiving anything

float temperature =0.0f;

float potVoltage = 0.0f;

/\*\*

\* unsigned to get the full positive range of the buffer\_variable

\* and long as normal int cannot go beyond 65000 either but long can

\* go approx 4.2 million before overflowing

\*/

unsigned long int buffer\_variable = 0;

#define TASK\_READ\_TEMP\_POT\_P OSTCBP(1) //Task #1

#define TASK\_KNIGHT\_RIDER\_P OSTCBP(2) //Task #2

#define TASK\_RUN\_LED\_P OSTCBP(3) //Task #3

#define TASK\_PRINT\_TEMP\_POT OSTCBP(4) //Task #4

#define PRIO\_READ\_TEMP\_POT 10 //Task1 priorities

#define PRIO\_KNIGHT\_RIDER 10 //Task2 ,,

#define PRIO\_RUN\_LED 2 //Task3 ,,

#define PRIO\_PRINT\_TEMP\_POT 10 //Task4 ,,

#define BINSEM\_SIGNAL\_BEGIN OSECBP(1) //Semaphore

#define BINSEM\_SIGNAL\_READ\_COMPLETE OSECBP(2) //Semaphore

////////////////////////////////////////////////////////////////////////////////

void TaskReadTempAndPot (void)

{

while(1)

{

OSSignalBinSem(BINSEM\_SIGNAL\_BEGIN);

temperature = ADC\_Operate(4); //using ANI4

temperature = (((temperature \*0.00322) - 0.5)/0.01);

potVoltage = ADC\_Operate(5); //using ANI5

potVoltage = (potVoltage \* 0.00322);

OSSignalBinSem(BINSEM\_SIGNAL\_READ\_COMPLETE);

OS\_Yield();

}

}

////////////////////////////////////////////////////////////////////////////////

void TaskKnightRider (void)

{

while(1)

{

OS\_WaitBinSem(BINSEM\_SIGNAL\_BEGIN, OSNO\_TIMEOUT);

if (PORTDbits.RD6 == 0)

{

/\*\*

\* Run Straight ProjectKnightRider

\*/

PORTAbits.RA0 = 1;

delayFunc();

PORTAbits.RA1 = 1;

delayFunc();

PORTAbits.RA2 = 1;

delayFunc();

PORTAbits.RA3 = 1;

delayFunc();

PORTAbits.RA4 = 1;

delayFunc();

PORTAbits.RA5 = 1;

delayFunc();

PORTAbits.RA6 = 1;

delayFunc();

PORTA &=0x0080;

}

else

{

/\*\*

\* Run reverse ProjectKnightRider

\*/

PORTAbits.RA6 = 1;

delayFunc();

PORTAbits.RA5 = 1;

delayFunc();

PORTAbits.RA4 = 1;

delayFunc();

PORTAbits.RA3 = 1;

delayFunc();

PORTAbits.RA2 = 1;

delayFunc();

PORTAbits.RA1 = 1;

delayFunc();

PORTAbits.RA0 = 1;

delayFunc();

PORTA &=0x0080;

}

}

}

void TaskPrintTempAndPot (void)

{

while(1)

{

OS\_WaitBinSem(BINSEM\_SIGNAL\_READ\_COMPLETE, OSNO\_TIMEOUT);

char TxString[100] = "";

sprintf(TxString,"\n\n#> Voltage: %4.2f Temperature: %4.2f", potVoltage, temperature);

U2\_TxString(TxString);

}

}

////////////////////////////////////////////////////////////////////////////////

void TaskRunLED (void)

{

while(1)

{

/\*\*

\* Flip LED

\*/

PORTA ^= 0x0080;

OS\_Delay(10);

}

}

////////////////////////////////////////////////////////////////////////////////

int main (void)

{

OSInit();

OSCreateTask(TaskReadTempAndPot, TASK\_READ\_TEMP\_POT\_P, PRIO\_READ\_TEMP\_POT);

OSCreateTask(TaskKnightRider, TASK\_KNIGHT\_RIDER\_P, PRIO\_KNIGHT\_RIDER );

OSCreateTask(TaskRunLED, TASK\_RUN\_LED\_P, PRIO\_RUN\_LED);

OSCreateTask(TaskPrintTempAndPot, TASK\_PRINT\_TEMP\_POT, PRIO\_PRINT\_TEMP\_POT);

// OSCreateTask(TaskRead, TASK\_Read, PRIO\_Read);

OSCreateBinSem(BINSEM\_SIGNAL\_BEGIN, 0);

OSCreateBinSem(BINSEM\_SIGNAL\_READ\_COMPLETE, 0);

SYSTEM\_Initialize(); //PIC initialization

// start multitasking/scheduler/despatcher.

while(1){

OSSched(); //pass control to the scheduler(kernel))

}

}

void \_\_attribute\_\_ ( ( interrupt, no\_auto\_psv ) ) \_T1Interrupt ( )

{

if(buffer\_variable < 100000)

{

buffer\_variable++;

}

if(buffer\_variable >= 100000)

{

buffer\_variable = 0;

/\*\*

\* Toggling LED

\*/

if(PORTAbits.RA7 == 0)

{

PORTAbits.RA7 = 1;

}

else

{

PORTAbits.RA7 = 0;

}

}

IFS0bits.T1IF = 0;

static uint8\_t i = 10; //counter = 1 ms x 30 = 30ms

if(IEC0bits.T1IE && IFS0bits.T1IF )

{

IFS0bits.T1IF = false;

if ( !(--i) )

{

i = 10;

OSTimer(); //call salvo service at 10 ms interval, Tick Rate = 10 mS //call salvo service at 10 ms interval, Tick Rate = 10 mS

}

}

}

void \_\_attribute\_\_ ((interrupt, no\_auto\_psv)) \_U2RXInterrupt( )

{

if(buffer\_variable<100000)

if(IEC1bits.U2RXIE == 1)

{

if(IFS1bits.U2RXIF == true) //check the status flag

{

rcvByte = U2RXREG; //Read the received byte

}

}

IFS1bits.U2RXIF = false; //reset the status flag

}

//////////////////////////////////////////////////////////////////////

void SYSTEM\_Initialize(void)

{

PIN\_MANAGER\_Initialize();

OSCILLATOR\_Initialize();

TMR1\_Initialize();

UART2\_Initialize();

ADC\_Initialize();

}

////////////////////////////////////////////////////////////////////////////////

void OSCILLATOR\_Initialize(void)

{

// NOSC PRI; SOSCEN disabled; OSWEN Switch is Complete;

\_\_builtin\_write\_OSCCONL((uint8\_t) (0x0200 & 0x00FF));

// RCDIV FRC/2; DOZE 1:8; DOZEN disabled; ROI disabled;

CLKDIV = 0x3100;

// TUN Center frequency;

OSCTUN = 0x0000;

// WDTO disabled; TRAPR disabled; SWDTEN disabled; EXTR disabled; POR disabled; SLEEP disabled; BOR disabled; IDLE disabled; IOPUWR disabled; VREGS disabled; CM disabled; SWR disabled;

RCON = 0x0000;

}

////////////////////////////////////////////////////////////////////////////////

void PIN\_MANAGER\_Initialize(void)

{

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

\* Setting the Output Latch SFR(s)

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

LATA = 0x0000;

LATB = 0x0000;

LATC = 0x0000;

LATD = 0x0000;

LATE = 0x0000;

LATF = 0x0000;

LATG = 0x0000;

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

\* Setting the GPIO Direction SFR(s)

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

TRISA = 0x0000;

TRISB = 0xFFFF;

TRISC = 0xF01E;

TRISD = 0xFFFF;

TRISE = 0x03FF;

TRISF = 0x31FF;

TRISG = 0xF3CF;

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

\* Setting the Weak Pull Up and Weak Pull Down SFR(s)

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

CNPU1 = 0x0000;

CNPU2 = 0x0000;

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

\* Setting the Open Drain SFR(s)

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

ODCA = 0x0000;

ODCB = 0x0000;

ODCC = 0x0000;

ODCD = 0x0000;

ODCE = 0x0000;

ODCF = 0x0000;

ODCG = 0x0000;

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

\* Setting the Analog/Digital Configuration SFR(s)

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

AD1PCFG = 0x00C0;

}

////////////////////////////////////////////////////////////////////////////////

void TMR1\_Initialize (void)

{

TMR1 = 0x0000;

//Instructions from Manual

// 1. Set the TON bit (= 1).

/\*\*

\* Timer setting done at the last

\*/

// 2. Select the timer pre scaler ratio using the

T1CONbits.TCKPS = 0b00; //1:1 ratio selected

/\*\*

\* We are using the lowest possible pre scaler to get

\* a more precis time interval facilitating more control

\* over the time

\*/

// 3. Set the Clock and Gating modes using the TCS

T1CONbits.TCS = 0; //using internal clock so TCS=0

T1CONbits.TGATE = 0; //TGATE is set to 0 as we want the output from the PreScaler to generate out interrupts

// 4. Set or clear the TSYNC bit to configure synchronous or asynchronous operation.

T1CONbits.T1SYNC = 1; //turning on synchronization of the clock

// 5. Load the timer period value into the PR1

PR1 = (uint16\_t) 0x0078; //The value of the pre scaler goes here register.

//58000 = E290

IFS0bits.T1IF = 0;

// Interrupt priority setting

// 7 = 0b0111

IPC0bits.T1IP2 = 1;

IPC0bits.T1IP1 = 1;

IPC0bits.T1IP0 = 1;

IEC0bits.T1IE = 1;

T1CONbits.TON = 1;

// Ans: Check the Interrupt\_Initialize for more details

}

////////////////////////////////////////////////////////////////////////////////

void UART2\_Initialize (void)

{

// STSEL 1; IREN disabled; PDSEL 8N; UARTEN enabled; RTSMD disabled; USIDL disabled; WAKE disabled; ABAUD disabled; LPBACK disabled; BRGH enabled; RXINV disabled; UEN TX\_RX;

U2MODE = 0x8002; //0x8008;

// OERR NO\_ERROR\_cleared; URXISEL RX\_ONE\_CHAR; UTXBRK COMPLETED; UTXEN disabled; ADDEN disabled; UTXISEL0 TX\_ONE\_CHAR; UTXINV disabled;

U2STA = 0x0000;

// U2TXREG 0;

U2TXREG = 0x0000;

// BaudRate = 9600; Frequency = 2000000 Hz; BRG 51;

U2BRG = 0x0019;

IEC1bits.U2RXIE = 1; //Rx interrupt Enabled

U2STAbits.UTXEN = 1;

IEC1bits.U2TXIE = 0; //Tx interrupt disabled

IFS1bits.U2RXIF = 0;

IPC7bits.U2RXIP2 = 1;

IPC7bits.U2RXIP1 = 0;

IPC7bits.U2RXIP0 = 1;

}

void ADC\_Initialize ( void )

{

/\*\*

\* Setting pin4 and pin5 as a analog input

\*/

AD1PCFG = 0xFFCF;

/\*\*

\* Start the sample enable bit

\* from bit2 and set the SSRC<2:0>

\* to 111 - (Internal counter ends sampling

\* and starts conversion (auto-convert))

\*/

AD1CON1 = 0x00E2;

AD1CON2 = 0x0000;

/\*\*

\* AD1CON3 value bit7 to bit0 has been set to 0 so

\* that ADCS is set to zero and TAD = Tcy

\*/

AD1CON3 = 0x1100;

AD1CSSL = 0x0000;

/\*\*

\* Setting the initial value for AD1CHS to 0x0004

\*/

AD1CHS = 0x0004;

AD1CON1bits.ADON = 1;

}

int ADC\_Operate (int value)

{

while (1) {

AD1CHS = value;

AD1CON1bits.SAMP = 1;

while (!AD1CON1bits.DONE);

return ADC1BUF0;

}

}

int ADC\_Temp(int value)

{

AD1CHS = value;

AD1CON1bits.SAMP = 1;

while (!AD1CON1bits.DONE);

if (IFS0bits.AD1IF == 1) {

return ADC1BUF0;

}

}

void U2\_TxByte(char value)

{

while (!U2STAbits.TRMT);

U2TXREG = value;

Nop();

Nop();

Nop();

}

void U2\_TxString(char \*s)

{

char nextChar;

while(\*s != '\0')

{

nextChar = \*s++;

U2\_TxByte(nextChar);

}

U2\_TxByte(0x0d); //CR

U2\_TxByte(0x0a); //LF

Nop();

Nop();

}

void delayFunc(void)

{

int j,k;

int a;

for(j = 0; j < count1; j++)

{

for(k=0; k < count2; k++)

{

a = 0; //How many times (frequency) this instruction getting executed?

}

}

}

////////////////////////////////////////////////////////////////////////////////

//Revision History:

//Dec 19/17: original build up.