**“timer0.c”**

#include <xc.h>

#include "timer0.h"

void timerInit(void){

T0CON0 = 0b10001001; // tmr0 enable, 8 bit, 1:10 Postscaler

T0CON1 = 0b10110001; // 500 kHz, async, 1:2 prescaler

TMR0H = 0xF9; // TMR0H 0 - 249 count for a period of 250

// enable global interrupts

INTCON0bits.GIE = 1;

// TMR0 interrupt enable (set TMR0IE bit)

PIE3bits.TMR0IE = 1;

// clear timer0 interrupt flag (clear TMR0IF bit)

PIR3bits.TMR0IF = 0;

// TMR0 high priority interrupt

IPR3bits.TMR0IP = 1;

// attach interrupt to TMR0

void \_\_interrupt(irq(TMR0),high\_priority)tickINT(void);

// attach default interrupt

void \_\_interrupt(irq(default),high\_priority)defaultINT(void);

return;

}

// isr for tmr0 interrupt

void tickINT(void){

PIR3bits.TMR0IF = 0; // clear the timer0 interrupt flag

tick\_count++; // increment tick\_count variable

return;

}

// isr for default interrupt

void defaultINT(void){

return;

}

**“timer0.h”**

/\*

\* File: timer0.h

\* Author: a\_hui

\*

\* Created on March 14, 2023, 10:44 AM

\*/

#ifndef TIMER0\_H

#define TIMER0\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

#include <xc.h>

#define LEDbit RF3

// global variable for isr

volatile unsigned char tick\_count = 0;

// initializes the 10ms timer0 interrupt

extern void timerInit(void);

// timer0 isr

extern void tickINT(void);

// isr for any other interrupts

extern void defaultINT(void);

// attaches the tickINT isr to timer0 flag

extern void \_\_interrupt(irq(TMR0),high\_priority)tickINT(void);

// default interrupt isr attachment

extern void \_\_interrupt(irq(default),high\_priority)defaultINT(void);

#ifdef \_\_cplusplus

}

#endif

#endif /\* TIMER0\_H \*/

**“timer0config.h”**

/\*

\* File: timer0config.h

\* Author: a\_hui

\*

\* config bits for projects using the timer0 module

\* Created on March 13, 2023, 6:35 PM

\*/

#ifndef TIMER0CONFIG\_H

#define TIMER0CONFIG\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

// PIC18F57Q43 Configuration Bit Settings

// 'C' source line config statements

// CONFIG1

#pragma config FEXTOSC = OFF // External Oscillator Selection (Oscillator not enabled)

#pragma config RSTOSC = HFINTOSC\_64MHZ// Reset Oscillator Selection (HFINTOSC with HFFRQ = 64 MHz and CDIV = 1:1)

// CONFIG2

#pragma config CLKOUTEN = OFF // Clock out Enable bit (CLKOUT function is disabled)

#pragma config PR1WAY = ON // PRLOCKED One-Way Set Enable bit (PRLOCKED bit can be cleared and set only once)

#pragma config CSWEN = ON // Clock Switch Enable bit (Writing to NOSC and NDIV is allowed)

#pragma config FCMEN = ON // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor enabled)

// CONFIG3

#pragma config MCLRE = EXTMCLR // MCLR Enable bit (If LVP = 0, MCLR pin is MCLR; If LVP = 1, RE3 pin function is MCLR )

#pragma config PWRTS = PWRT\_OFF // Power-up timer selection bits (PWRT is disabled)

#pragma config MVECEN = ON // Multi-vector enable bit (Interrupt contoller does not use vector table to prioritze interrupts)

#pragma config IVT1WAY = OFF // IVTLOCK bit One-way set enable bit (IVTLOCKED bit can be cleared and set repeatedly)

#pragma config LPBOREN = OFF // Low Power BOR Enable bit (Low-Power BOR disabled)

#pragma config BOREN = SBORDIS // Brown-out Reset Enable bits (Brown-out Reset enabled , SBOREN bit is ignored)

// CONFIG4

#pragma config BORV = VBOR\_1P9 // Brown-out Reset Voltage Selection bits (Brown-out Reset Voltage (VBOR) set to 1.9V)

#pragma config ZCD = OFF // ZCD Disable bit (ZCD module is disabled. ZCD can be enabled by setting the ZCDSEN bit of ZCDCON)

#pragma config PPS1WAY = ON // PPSLOCK bit One-Way Set Enable bit (PPSLOCKED bit can be cleared and set only once; PPS registers remain locked after one clear/set cycle)

#pragma config STVREN = ON // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)

#pragma config LVP = ON // Low Voltage Programming Enable bit (Low voltage programming enabled. MCLR/VPP pin function is MCLR. MCLRE configuration bit is ignored)

#pragma config XINST = OFF // Extended Instruction Set Enable bit (Extended Instruction Set and Indexed Addressing Mode disabled)

// CONFIG5

#pragma config WDTCPS = WDTCPS\_31// WDT Period selection bits (Divider ratio 1:65536; software control of WDTPS)

#pragma config WDTE = OFF // WDT operating mode (WDT Disabled; SWDTEN is ignored)

// CONFIG6

#pragma config WDTCWS = WDTCWS\_7// WDT Window Select bits (window always open (100%); software control; keyed access not required)

#pragma config WDTCCS = SC // WDT input clock selector (Software Control)

// CONFIG7

#pragma config BBSIZE = BBSIZE\_512// Boot Block Size selection bits (Boot Block size is 512 words)

#pragma config BBEN = OFF // Boot Block enable bit (Boot block disabled)

#pragma config SAFEN = OFF // Storage Area Flash enable bit (SAF disabled)

#pragma config DEBUG = OFF // Background Debugger (Background Debugger disabled)

// CONFIG8

#pragma config WRTB = OFF // Boot Block Write Protection bit (Boot Block not Write protected)

#pragma config WRTC = OFF // Configuration Register Write Protection bit (Configuration registers not Write protected)

#pragma config WRTD = OFF // Data EEPROM Write Protection bit (Data EEPROM not Write protected)

#pragma config WRTSAF = OFF // SAF Write protection bit (SAF not Write Protected)

#pragma config WRTAPP = OFF // Application Block write protection bit (Application Block not write protected)

// CONFIG10

#pragma config CP = OFF // PFM and Data EEPROM Code Protection bit (PFM and Data EEPROM code protection disabled)

// #pragma config statements should precede project file includes.

// Use project enums instead of #define for ON and OFF.

#include <xc.h>

#ifdef \_\_cplusplus

}

#endif

#endif /\* TIMER0CONFIG\_H \*/

**“timer0test.c”**

/\*

\* File: timer0test.c

\* Author: a\_hui

\*

\* Created on March 14, 2023, 10:43 AM

\*/

#include <xc.h>

#include "lcd18f.h"

#include "timer0config.h"

#include "timer0.h"

/\*

\*

\*/

int main(int argc, char\*\* argv) {

signed int secondCount = 990; // seconds count variable

signed int tmp\_secondCount = secondCount; // temporary seconds count variable for processing

char mod = 0; // modulo variable to store modulo divide

char seconds[4] = "000";

// set up port F

TRISF = 0x00;

ANSELF = 0x00;

PORTF = 0x00; // turn off led

TRISC = 0x00;

ANSELC = 0x00;

LATC= 0x00;

// initialize LCD

LCDinit();

// initialize timer0 interrupt

timerInit();

LCDprints(seconds);

// main loop

while(1){

LATC = tick\_count;

if(tick\_count == 50){

LEDbit = 0; // turn on LED

}

if(tick\_count > 99){

tick\_count -= 100;

secondCount++;

LEDbit = 1; // turn off LED

if(secondCount > 999){ // secondCount rollover

secondCount = 0;

};

tmp\_secondCount = secondCount; // copy second count for processing

// convert each digit in tmp\_secondCount to ASCII

for(signed char i = 2; i > -1; i--){

mod = tmp\_secondCount % 10; // get LS digit

seconds[i] = mod + '0'; // convert to ascii

tmp\_secondCount /= 10; // get next LS digit

}

// display second count on LCD

LCD\_HOME; // rewrite lcd

LCDprints(seconds); // print the seconds to the lcd

}

}

return (EXIT\_SUCCESS);

}