# Lab 9

# Real Time Clock

In this week’s lab we will build a real time clock using the timer0 code from Lab8.

Use the functions created in Lab8 to design a real time clock. Make sure that your code conforms to the following specification:

**Software:**

Implement a real time clock.

* The time is displayed to the nearest second on the LCD display.
* You must display Hours, minutes and seconds on the first line in the following format: HH:MM:SS Xm where X = a or p
* Timer 0 is to be used to give a 10ms ' tickCount ' using an interrupt.
* RF3 will flash every second.
* The clock code must be your own.

You must have completely commented code. Also include the following comment block.

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

Student Name: \*\*\*\*\*\*Include this comment block\*\*\*\*\*\*

File Name:

Description:

Demo Date: \*\*\*\*\*\*fill in the marking check list\*\*\*\*\*\*

Marking Check List

Good Comments:

Good Modular Code:

Unsigned Char used correctly:

Format HH:MM:SS Xm correct:

Increments each Second:

10ms Tick correct:

Only valid characters are display:

Clock is accurate:

Am/pm roll over is correct:

Hours per day correct:

Bugs:

Mark:

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

**“RTCmain.c”**

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

Student Name: Aaron Huinink

File Name: RTCmain.c

Description: Implements a real time clock on the lcd display

Demo Date: March 14, 2023

Marking Check List

Good Comments:

Good Modular Code:

Unsigned Char used correctly:

Format HH:MM:SS Xm correct:

Increments each Second:

10ms Tick correct:

Only valid characters are display:

Clock is accurate:

Am/pm roll over is correct:

Hours per day correct:

Bugs:

Mark:

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

#include <xc.h>

#include "timer0.h"

#include "lcd18f.h"

#include “RealTimeClock.h”

#include "timer0config.h"

void main(void){

// ============================ SETUP =================================//

// ---- VARIABLES ---- //

// time variable to store the current time

time current\_time;

current\_time.hours = 12;

current\_time.minutes = 0;

current\_time.seconds = 0;

current\_time.meridian = 'a';

// initialize RTC run flag

nSetRTC = 1;

// ---- HARDWARE INIT ---- //

// initialize LCD

LCDinit();

// ============================ MAIN LOOP =================================//

while(1){

RTCrun(&current\_time); // run the clock

};

}

**“RealTimeClock.h”**

/\*

\* File: RealTimeClock.h

\* Author: Aaron Huinink

\*

\* Created on March 20, 2023, 6:31 PM

\*/

#ifndef REALTIMECLOCK\_H

#define REALTIMECLOCK\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

#include <xc.h>

#include <pic18f57q43.h>

#include "timer0.h"

#include "lcd18f.h"

#include "timer0config.h"

// nSet flag for breaking out of run loop

unsigned char nSetRTC;

// typedef struct for time variables

typedef struct realtime{

char hours;

char minutes;

char seconds;

char meridian;

}time;

// ----- setINT ----- //

/\*

\* attaches an IOC interrupt allows you to set the RTC clock

\* ARGS: [void]

\* RETURNS: [void]

\*/

// interrupt attach prototype

void \_\_interrupt(irq(IOC),high\_priority)setINT(void);

// interrupt service routine protoype

void setINT(void);

// ----- RTCinit ----- //

/\*

\* initializes a real time clock

\* ARGS: [

\* start\_time<\*time> : pointer to time variable containing the time to begin at;

\* ]

\* RETURNS: [void]

\*/

void RTCrun(time \*start\_time);

#ifdef \_\_cplusplus

}

#endif

#endif /\* REALTIMECLOCK\_H \*/

**“RealTimeClock.c”**

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

Student Name: Aaron Huinink

File Name: RealTimeClock.c

Description: Implements a real time clock on the lcd display

Demo Date: March 14, 2023

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

#include "RealTimeClock.h"

void RTCrun(time \*start\_time) {

// array of pointers for printf statement

char\* current\_time[4] = {&start\_time->hours, &start\_time->minutes, &start\_time->seconds, &start\_time->meridian};

// enable RTC to run

nSetRTC = 1;

// initialize timer0

timerInit();

// main loop

while(nSetRTC){

// check 10ms timer0 interrupt variable

if(tick\_count > 99){ // if a second has past

tick\_count -= 100; // remove 100 10ms cycles from the count

(start\_time)->seconds++; // increment seconds

if((start\_time)->seconds > 59){ // if a minute has past

(start\_time)->seconds = 0; // rollover seconds

(start\_time)->minutes++; // increment minutes

if((start\_time)->minutes>59){ // if an hour has past

(start\_time)->minutes = 0; // rollover minutes

(start\_time)->hours++; // increment hours

if((start\_time)->hours>11){ // if it's past 11am or 11pm

if((start\_time)->hours >12){ // if it's past midnight or noon

(start\_time)->hours = 1; // rollover hours

}else{ // if its midnight or noon

(start\_time)->meridian = ( // toggle am/pm

((start\_time)->meridian == 'a')? 'p' : // if am then pm

'a' // otherwise am

);

}

}

}

};

// print time to lcd

LCD\_HOME;

LCDprintf("%i2:%i2:%i2 %cm", current\_time);

}

};

};

**“lcd18f.h”**

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

\* lcd18f.h

\* ECET165 Embedded Micro-controllers

\* Created: 13 Feb 2023

\* Aaron Huinink

\* Provides functionality for pic18f micro-controllers to connect to an lcd on

\* port d.

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

#ifndef LCD18F\_H

#define LCD18F\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

#include <xc.h>

#define LCD\_PORT PORTD

#define LCD\_LAT LATD

#define LCD\_TRIS TRISD

#define LCD\_ANSEL ANSELD

#define LCD\_WPU WPUD

#define LCD\_K LATDbits.LATD7

#define LCD\_EN LATDbits.LATD6

#define LCD\_RW LATDbits.LATD5

#define LCD\_RS LATDbits.LATD4

#define LCD\_BF PORTDbits.RD3

#define LCD\_STROBE LCD\_EN = 1; \_\_delay\_us(1); LCD\_EN = 0

#define LCD\_PORTEN LCD\_ANSEL = 0x0; LCD\_TRIS = 0x0; LCD\_WPU = 0xFF; LCD\_K = 1; \_\_delay\_us(1)

#define LCD\_CLEAR LCDinstruct(0x01)

#define LCD\_HOME LCDinstruct(0x02)

#define LCD\_CURSORBLINK LCDinstruct(0x0F)

#ifndef \_XTAL\_FREQ

#define \_XTAL\_FREQ 64000000

#endif

// ============================ FUNCTION PROTOTYPES ========================== //

// ----- instruct ----- //

/\*

\* sends an instruction to the instruction register

\* ARGS: (instr<unsigned char> : the 2 byte instruction to be sent to the lcd)

\* RETURNS: [void]

\*/

extern void LCDinstruct(unsigned char instr);

// ----- LCDinit ----- //

/\*

\* initializes an LCD display in 4 bit, 2 line display mode

\* ARGS: (void)

\* RETURNS: [void]

\*/

extern void LCDinit(void);

// ----- LCDprint ----- //

/\*

\* prints a character to the lcd

\* ARGS: (c<char> : the character to print to the lcd)

\* RETURNS: [void]

\*/

extern void LCDprintc(char c);

// ----- LCDprint ----- //

/\*

\* prints a string of chars to the lcd

\* ARGS: (\*c<char> : the string to print to the lcd)

\* RETURNS: [void]

\*/

extern void LCDprints(char \*c);

// ----- LCDgoto ----- //

/\*

\* goes to an address in DDRAM

\* ARGS: (pos<unsigned char> : the position to go to in DDRAM)

\* RETURNS: [address<unsigned char> : the current address in the address counter]

\*/

extern void LCDgoto(unsigned char pos);

// ----- LCDreturn ----- //

/\*

\* goes to next line of lcd display and clears it

\* ARGS: (void)

\* RETURNS: [void]

\*/

extern unsigned char LCDreturn(unsigned char pos);

// ----- toString ----- //

/\*

\* converts an integer into a string

\* ARGS: [

\* number<unsigned char> : the number to convert to a string;

\* length<unsigned char> : the number of digits in the number;

\* RETURNS: [string<char\*> : the address of the string containing the integer]

\*/

extern char\* toString(unsigned char number, unsigned char length);

// ----- LCDprintf ----- //

/\*

\* printf function for the LCD display

\* ARGS: [

\* shell<char\*> : string to format variables into;

\* inputs<char\*\*> : array of pointers to the addresses of the variables to input into shell string;

\* ]

\* RETURNS: [void]

\*/

extern void LCDprintf(char\* shell, char\* inputs[]);

#ifdef \_\_cplusplus

}

#endif

#endif /\* LCD18F\_H \*/

**“lcd18f.c”**

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

\* lcd18f.c

\* ECET165 Embedded Micro-controllers

\* Created: 13 Feb 2023

\* Aaron Huinink

\* Provides functionality for pic18f micro-controllers to connect to a 4bit

\* 1 line 5x8 font lcd on port d.

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

// ============================ INCLUDES/DEFINES =================================//

#include <xc.h>

#include "lcd18f.h"

// ============================ FUNCTION DEFS =================================//

void LCDinstruct(unsigned char instr){

LCD\_PORTEN; // enable LCD port

// set rs to instruction register, write mode

LCD\_RW = 0;

LCD\_RS = 0;

// send instruction

LCD\_LAT = (LCD\_LAT & 0xF0)| (instr >> 4 & 0x0F); // send MSnibble

LCD\_STROBE;

\_\_delay\_us(1);

LCD\_LAT = (LCD\_LAT & 0xF0)|(instr & 0x0F); // send LSnibble

LCD\_STROBE;

\_\_delay\_ms(5); // allow busy flag to clear

return;

};

void LCDinit(void){

\_\_delay\_ms(50); // wait for powerup

LCD\_PORTEN;

LCD\_RS = 0;

LCD\_RW = 0;

\_\_delay\_us(1);

// send the function set instruction 3 times

LCD\_LAT = (LCD\_LAT & 0xF0) | 0x03;

LCD\_STROBE;

\_\_delay\_ms(5);

LCD\_LAT = (LCD\_LAT & 0xF0) | 0x03;

LCD\_STROBE;

\_\_delay\_ms(5);

LCD\_LAT = (LCD\_LAT & 0xF0) | 0x03;

LCD\_STROBE;

\_\_delay\_ms(5);

//set 4 bit mode

LCD\_LAT = (LCD\_LAT & 0xF0) | 0x02;

LCD\_STROBE;

\_\_delay\_ms(5);

// run setup functions

LCDinstruct(0x28); // set 4 bit mode, 2 line display, 5x8 font

LCDinstruct(0x08); // display off

LCDinstruct(0x0F); // display on, cursor on, blink on

LCDinstruct(0x01); // display clear

LCDinstruct(0x06); // entry mode set

return;

};

void LCDprintc(char c){

LCD\_PORTEN;

LCD\_RS = 1;

LCD\_RW = 0;

// print character to lcd

LCD\_LAT = (LCD\_LAT & 0xF0)|(c>>4); // send MSnibble

\_\_delay\_us(1);

LCD\_STROBE;

LCD\_LAT = (LCD\_LAT & 0xF0)|(c & 0x0F); // send LSnibble

\_\_delay\_us(1);

LCD\_STROBE;

\_\_delay\_ms(5); // allow busy flag to clear

return;

}

void LCDprints(char \*c){

LCD\_PORTEN; // enable the lcd port

// set rs to data register, r/w to write mode

LCD\_RS = 1;

LCD\_RW = 0;

\_\_delay\_us(1);

while(\*c != 0x0){ // while there are valid chars in the string

// print character from c, increment c

LCDprintc(\*c++);

};

return;

};

void LCDgoto(unsigned char pos){

LCDinstruct(0x80+pos);

return;

}

unsigned char LCDreturn(unsigned char pos){

LCD\_PORTEN; // enable the LCD port

unsigned char next\_line = 0x40; // next line begin address variable

if (pos > 0x0F){

next\_line = 0x00;

}

LCDgoto(next\_line); // go to the beginning of the next line on the lcd

for(unsigned char i = 0; i < 40; i++){

LCDprintc(' ');

}

LCDgoto(next\_line);

return next\_line;

}

char\* toString(unsigned char number, unsigned char length){

char string[4] = ""; // max number is 255, so max string length is 3 + \0

unsigned char copy\_num = number; // copy number for processing

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

string[i] = (copy\_num % 10) + '0';

copy\_num /= 10;

};

return &string;

}

void LCDprintf(char\* shell, char\* inputs[]){

while(\*shell != 0x0){

// if %s

if((\*shell == '%') & (\*(shell+1) == 's')){

// deref inputs, pass pointer, increment inputs

LCDprints(\*inputs++);

shell+=2; //skip "%s"

}

// if %id, where d is the length of the integer to print

else if((\*shell == '%') & (\*(shell+1) == 'i')){

// deref inputs, deref pointer, convert to string, increment inputs

LCDprints(toString(\*\*inputs++, (\*(shell+2)-'0')));

shell+=3; // skip "%id"

// if %c

}else if((\*shell == '%') & (\*(shell+1) == 'c')){

// deref inputs, deref char pointer,

LCDprintc(\*\*inputs++);

shell+=2; // skip "%c"

}else{

// print the char in shell and increment shell

LCDprintc(\*shell++);

}

}

return;

}

**“keypad18f.h”**

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

\* Lab 7 - Matrix Keyboard

\* ECET165 Embedded Micro-controllers

\* keypad18f.h

\* CREATED 28 Feb 2023

\* UPDATED \*\*\*

\* v1.0

\* BY Aaron Huinink

\* Provides 4x4 matrix keypad functionality with a PIC18F uC.

\* Keypad is wired to port f.

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

#ifndef KEYPAD18F\_H

#define KEYPAD18F\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

// ============================ INCLUDES/DEFINES =================================//

#include <xc.h>

#ifndef \_XTAL\_FREQ

#define \_XTAL\_FREQ 64000000

#endif

#define KEY\_LAT LATF

#define KEY\_PORT PORTF

#define KEY\_ANSEL ANSELF

#define KEY\_TRIS TRISF

#define KEY\_WPU WPUF

#define KEY\_PORTEN KEY\_ANSEL = 0x00; KEY\_TRIS = 0xF0; KEY\_WPU = 0xF0; \_\_delay\_us(1);

// ============================ FUNCTION PROTOTYPES ========================== //

// ----- keyScan ----- //

/\*

\* Scans the column keys and returns the column and row values of the pressed key

\* ARGS: (void)

\* RETURNS: [key<unsigned char> : (column | row)]

\*/

extern char keyScan();

#ifdef \_\_cplusplus

}

#endif

#endif /\* KEYPAD18F\_H \*/

**“keypad18f.c”**

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

\* Lab 7 - Matrix Keyboard

\* ECET165 Embedded Micro-controllers

\* keypad18f.c

\* CREATED 27 Feb 2023

\* UPDATED \*\*\*

\* v1.0

\* BY Aaron Huinink

\* Provides 4x4 matrix keypad functionality with a PIC18F uC.

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

// ============================ INCLUDES/DEFINES =================================//

#include <xc.h>

#include "keypad18f.h"

// ============================ FUNCTION DEFINITIONS ========================== //

// ----- keyScan ----- //

/\*

\* Scans the column keys and returns the column and row values of the pressed key

\* ARGS: (void)

\* RETURNS: [key<unsigned char> : {column, row}]

\*/

char keyScan(){

unsigned char colshift = 0x00; // shift variable for checking columns

unsigned char cols = 0x0; // store row pin input

unsigned char rowshift = 0x00; // shift variable for checking rows

// char lookup table

char lookup[4][4] = {

{'1', '2', '3', 'A'},

{'4', '5', '6', 'B'},

{'7', '8', '9', 'C'},

{'\*', '0', '#', 'D'}

};

KEY\_PORTEN; // enable the keypad port

KEY\_LAT = 0x00; // turn on column pins pins

while(KEY\_PORT == 0xF0); // read col pins and wait for a key press

\_\_delay\_ms(15); // debounce

while(!(cols)){ // while there's no reading on the col pins

if(rowshift > 0x3){ // ensure rowshift is in range

rowshift = 0x0;

}

KEY\_LAT = ~(0x08>>rowshift); // cycle a 0 through the row pins

\_\_delay\_us(1);

cols = (~(KEY\_PORT)& 0xF0); // read the 1s complement of the column pins

rowshift++; // increment colshift by one to cycle through next pin

}

rowshift--; // decrement to remove additional column shift

// wait for key release

KEY\_LAT = 0x00; // turn on row pins

\_\_delay\_us(1);

while(KEY\_PORT != 0xF0); // wait for a 0xF reading on the col pins

KEY\_LAT = 0xF0; // turn off column pins

\_\_delay\_ms(15); // debounce

// count the column pins to find the col number

while(!(cols<<colshift & 0x80)){ // while MSB of column pins not 1

colshift++; // add one to the colshift variable and shift again

}

return lookup[rowshift][colshift]; // return the keypad char

}