ECE-4273- Digital Design Laboratory - Spring 2020
School of Electrical and Computer Engineering
University of Oklahoma

Final Project

April 24th, 2020

Matthew Johnson
Talia Nguyen
(Tuesday-Thursday Section)

User Manual

Labeled Diagram of Display and Switches

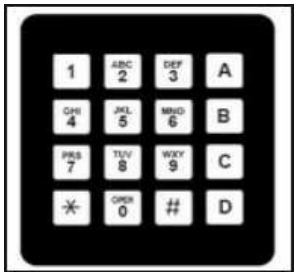
HD44780 (LCD Display)

NHD-0420DZ-FL-YBW-33V3

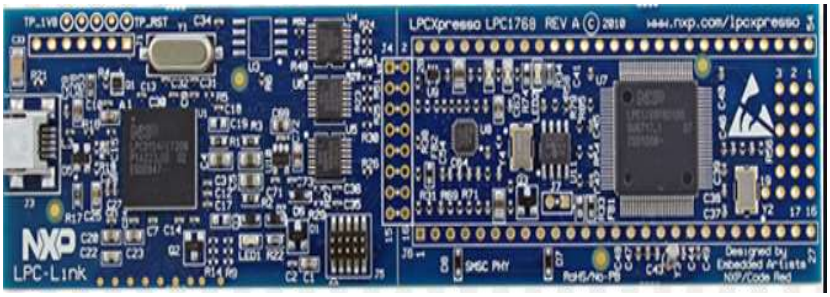
Quad-Encoder

4 x 4 Keypad

96BB2-006-R



LPC1769



Piezo Speaker

Instruction for Use

- Plug in the USB power source to turn on the clock
- The first thing to display on the LCD will be the real time clock of the LPC1769, which if battery backup is connected will be showing the current time. Also, you will see a down arrow at the corner of the screen which is showing there is something else at the bottom or next page of the screen.
- Turn the knob of the Quad-Encoder to go to the next screen.
- On the second screen, there will be 4 different options (that can be used on either page first screen or second screen) as below.

- A: Set Time
- B: Set Alarm
- C: On/Off
- D: Set Snooze

- Option A is for setting the current time. In order to change the time as you want, you can push button A on the keypad. Then, a pop-up screen will show up as below.

- Change Current Time
- XX : XX AM/PM
- Press # to Exit

You can type in the time you want into the place of XX. After the XX's are filled, there will appear two options, which are 1 for AM or 2 for PM. You can choose 1 or 2 depending on your demand.

- Option B is for setting the alarm clock. You can set the alarm by pushing button B on the keypad. A pop-up screen will show up as below.

- Change Alarm Time
- Current: YY : YY
- XX : XX AM/PM
- Press # to Exit

Similar to the process for option A, you can type the time you want the alarm to ring by filling the space of XX. Then, you can choose AM or PM by choosing number 1 or number 2. Setting the alarm will automatically turn the alarm on for use.

- Option C is to turn on or off the alarm. This is a simple one, so you just need to push the C button to turn the alarm off if it is on and vice versa.
- Option D is for setting up the snooze time. You can choose to limit the snoozing time to how many minutes (1-9 max) you want it to be by pushing the button D on the keypad. A pop-up screen will show up as below.

- Snooze Time: X min
- Rotate QEI Knob
- Press * to Accept
- Press # to Exit

Unlike the other 2 processes above of option A and optionB, we will use the QuadEncoder knob this time to control the value which will be filled in X space for the quantity of snooze time (note the keypad here works just as well for entering snooze time).

- For more detailed instructions, you can go to YouTube with this link attached: https://www.youtube.com/watch?v=lFlnLrqMCCE&fbclid=IwAR2oGriKLhIftusCXq-7ttVuvs9d2d5ET92B7BKkBTECaTKO01coYklW9mg

Explanation of Design

High level description of software

For the functionality of our alarm program we utilize multiple concepts that were learned during Digital Design Lab. Our program utilizes the real time clock registers (RTC) of the LPC1769 for keeping time. Every time the while(1) loop reiterates the register values of the RTC are stored into a variable and then displayed through the LCD. The time is stored into a variable before being displayed to prevent having an anomaly where the time displays as 4:01:59 → 4:01:00 → 4:02:01.

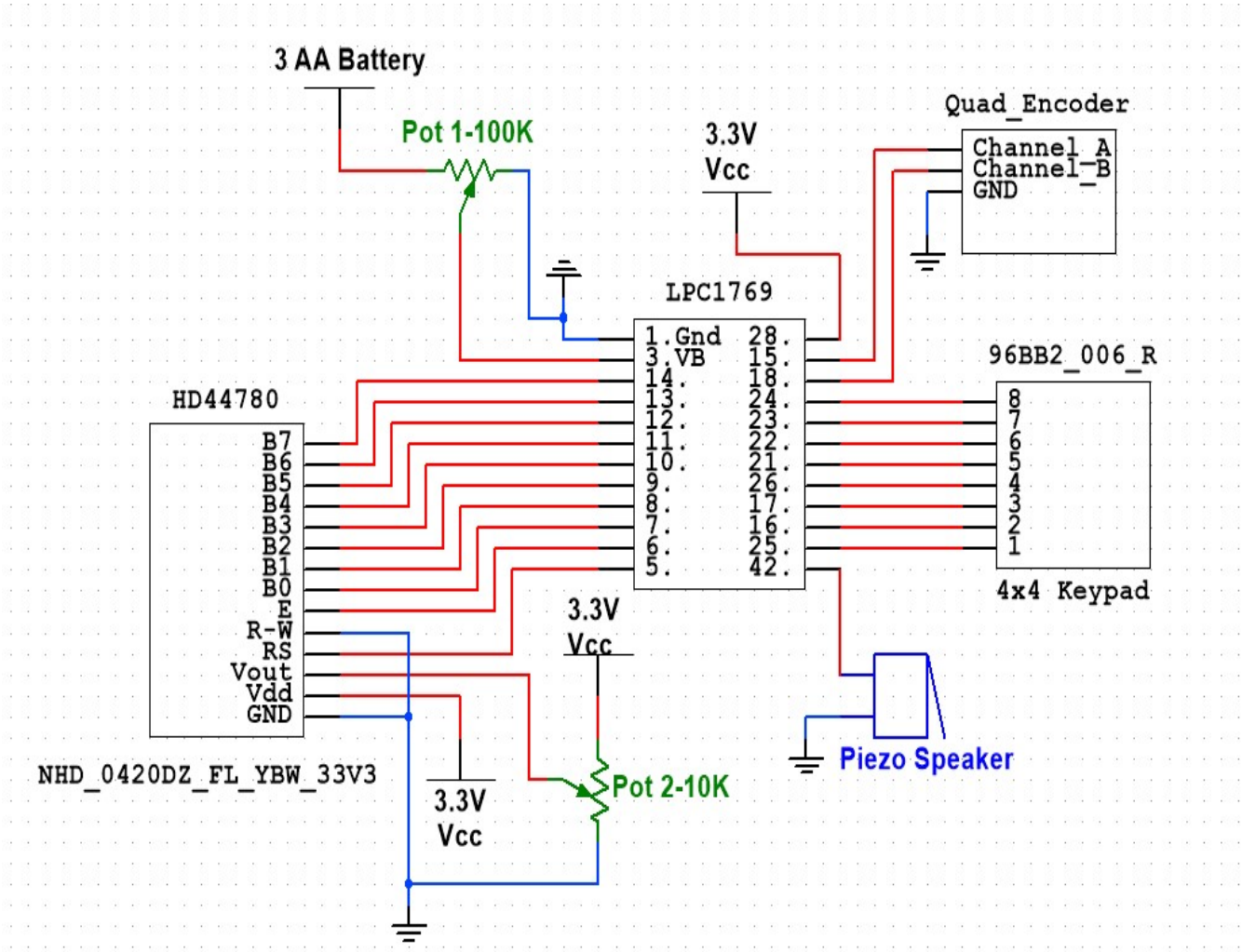
The 4x4 keypad used to send inputs is queried for status updates every 500 us by utilizing timed interrupts. The 8 pins of the keypad are separated into 4 inputs, 4 outputs. Voltage is applied to one output at a time, with each time a voltage is applied the status of the 4 inputs is checked to determine which keypad key has been pressed. If no key has been pressed, then char value 0 is returned (not '0' or char 48). In the main menu of the program the only acceptable returns from the keypad are the buttons A, B, C, D, or char values 65-68.

For the quadrature encoder (rotary knob) we utilize the interrupt feature of the LPC1769. When a interrupt is set a count variable is indexed, then an if statement is executed to see if count % 4 = 0 (through testing the counter, for every one turn causes 4 interrupts to be generated, 2 for rising, 2 for falling, so to prevent a statement from being executed 4 times we use modulus to run once for every 4 interrupts). If this is true, then we next look to see if a flag has been set to adjust snooze time, or if we are just turning 'pages' on the menu screen.

One of the limitations of the LPC found is that while with battery backup the RTC registers are saved, the alarm and alarm mask registers are not saved. This is not good as an alarm clock that can only save the time during power outages and not retain the alarm when it comes back on is essentially worthless. This issue was handled with the GPREG registers that are provided by the LPC1769 (it has four 32-bit registers that it provides a user to program to). These registers retain the data even without battery backup. By doing some and'ing, or'ing, and shifting we pack the alarm mask and the alarm time into one 32-bit word and then store it to the GPREG0 register. On bootup of the LPC it checks the values in this register and writes these values back to the alarm register and the alarm mask register, thereby restoring the alarm in case of power outage.

Detailed Design

Schematic



Software Listings

❖ Main

```
/*
        ============================================================================
        Name        : main.c
        Author      : Matthew Johnson & Talia Nguyen
        Version     :
        Copyright   : $(copyright)
        Description : https://youtu.be/lFlnLrqMCCE
        ============================================================================
        */

        #include "TimerLPC.h"
        #include "HD44780.h"
        #include "KeyPad.h"
        #include "LPC1769_RTC_DEFINITIONS.h"
        #include "PWM.h"

        #define START_SECONDS 0
        #define START_MINUTES 0
        #define START_HOUR 12

        bool PM = false;                          // Display AM/PM status
        bool ALARM_ON;                            // Set if Alarm is ON
        bool ALARM_ACTIVE = false;          // Set of Alarm is ACTIVE
        bool SNOOZE = false;                // Set if Snooze is ON
        bool SNOOZE_ALREADY = false; // Set if Snooze has ran ONCE
        int SNOOZE_TIME = 1;                // Time to SNOOZE
        unsigned int OLD_ALARM;                   // Saves old alarm
        unsigned int oldState = 0;                // Old QEI State
        unsigned int newState = 0;                // New QEI State
        unsigned volatile int change = 0;   // Change in State
        volatile bool PAGE1 = true;               // Set if PAGE1 display
        volatile int QEICOUNT = 0;                // Index for QEI
        char KEYPUSHED = 0;                       // Returns 4x4 keypushed
        volatile bool SNOOZE_QEI = false;
        void MAIN_MENU(void);               // Change time on display
        void printTime(unsigned int);// Print sub-routine
        void setTime(void);                       // Set time routine
        void setAlarm(void);                // Set alarm routine
        void setAlarmSub(unsigned int);     // Sub-routine for setAlarm
```

```
void toggleAlarm(void);                    // Toggle alarm ON/OFF
void alarm(void);                          // ALARM function
void setSnooze(void);              // Set Snooze function
void clockSetup(void);                     // RTC Clock Setup function
void _IRQ_SETUP(void);                     // IRQ Setup for 4x4 & QEI

extern "C" void TIMER0_IRQHandler(void){
	// Setup timer interrupt every 500 us
	if((T0IR >> 0) & 1){
		T0MR0 = T0MR0 + 500;
		KEYPUSHED = keyPress();
		T0IR = (1 << 0);
	}
}

extern "C" void EINT3_IRQHandler(void){
	// If Interrupt is detected on GPIO0
	// New state of ports 23/26 are detected
	// & combined as XX, compared to oldstate
	if((IOIntStatus >> 0) & 1){
		newState = ((FIO0PIN >> 22) & 0x2) | ((FIO0PIN >> 26) & 0x1);
		change = (((oldState << 1) | (oldState >> 1)) & 0x3) ^ newState;
		switch (change) {
			case 0b01:	QEICOUNT++;
					if((QEICOUNT%4) == 0){
						if(SNOOZE_QEI){
							if (SNOOZE_TIME == 9)
								SNOOZE_TIME = 1;
							else
								SNOOZE_TIME++;
						}
						else
							PAGE1 = !PAGE1;
					}
					break;
			case 0b10:	QEICOUNT--;
					if((QEICOUNT%4) == 0){
						if(SNOOZE_QEI){
							if(SNOOZE_TIME == 1)
								SNOOZE_TIME = 9;
							else
								SNOOZE_TIME--;
						}
```

```
                                                else
                                                        PAGE1 = !PAGE1;
                                        }
                                        break;
                }
                oldState = newState;
                IO0IntClr = (0b1001 << 23);
        }
}

int main(void) {
        setupHD44780();       // Setup HD44780 display
        setupKeyPad(); // Setup 4x4 Keypad
        _IRQ_SETUP();  // Setup IRQ for Keypad & QEI
        clockSetup();  // Setup RTC

        while(1) {
        if(ALARM_INTERRUPT)
                alarm();
        if(KEYPUSHED == 'A')
                setTime();
        if(KEYPUSHED == 'B')
                        setAlarm();
        if(KEYPUSHED == 'C')
                toggleAlarm();
        if(KEYPUSHED == 'D')
                setSnooze();
        MAIN_MENU();
        wait(.3);
    }
    return 0 ;
}

void setSnooze(void){
        do{
                unsigned int backup = SNOOZE_TIME;
                while(KEYPUSHED == 'D'){};
                char snoozeDig1temp = 0;

                commandLed(1);
                wordWrite("Snooze time: ");
                charWrite(SNOOZE_TIME + 48);
                wordWrite(" min");
```

```
                commandLed(0xC0);
                wordWrite("Rotate QEI knob");
                commandLed(0x94);
                wordWrite("Press * to accept");
                commandLed(0xD4);
                wordWrite("Press # to exit");
                commandLed(0xD);
                commandLed(0x8D);

                SNOOZE_QEI = true;
                unsigned int oldDigit = 0;
                while(snoozeDig1temp < '0' || snoozeDig1temp > '9'){
                        snoozeDig1temp = KEYPUSHED;
                        if(oldDigit != SNOOZE_TIME)
                                charWrite((SNOOZE_TIME) + 48);
                        oldDigit = SNOOZE_TIME;
                        commandLed(0x8D);
                        if(snoozeDig1temp == '#')
                                break;
                        if(snoozeDig1temp == '*'){
                                break;
                        }
                }
                SNOOZE_QEI = false;
                if(snoozeDig1temp == '*'){
                        wait_ms(500);
                        break;
                }
                if(snoozeDig1temp == '#'){
                        SNOOZE_TIME = backup;
                        break;
                }

                if(snoozeDig1temp == '0')
                        snoozeDig1temp = '1';
                charWrite(snoozeDig1temp);
                wait_ms(500);
                while(KEYPUSHED == snoozeDig1temp){}
                SNOOZE_TIME = snoozeDig1temp - 48;
        }while(KEYPUSHED == 'D');
        commandLed(0x0c);
}
```

```
void _IRQ_SETUP(void){
	T0TCR |= 1;    // Start Timer
	T0MR0 = T0TC + 500; // Interrupt 1000 us into future
	T0MCR |= (1 << 0);    // enable interrupt on MR0 match
	ISER0 = (1 << 1);     // Enable interrupts for Timer 0

	IO0IntEnR |= (0b1001 << 23); // Setup GPIO0 Interrupt on 23/26 Rising
	IO0IntEnF |= (0b1001 << 23); // Setup GPIO0 Interrupt on 23/26 Falling
	IO0IntClr = (0b1001 << 23);          // Clear GPIO0 Interrupts
	oldState = ((FIO0PIN >> 22) & 0x2) | ((FIO0PIN >> 26) & 0x1);     // Save
oldState
	ISER0 = (1 << 21);                          // Enable GPIO0/EINT3 Interrupt
}

void toggleAlarm(void){
	while(KEYPUSHED == 'C'){};
	if(!ALARM_ON){
		AMR = 0;
		AMR |= (0x1f << 3);   // Mask Alarm Registers year, month, doy, dow,
dom
		commandLed(1);
		commandLed(0xC4);
		wordWrite("Alarm ON");
		wait(1.5);
	}
	else{
		if(OLD_ALARM != ((ALHOUR << 6) | ALMIN)){
			ALHOUR = (OLD_ALARM >> 6);
			ALMIN = OLD_ALARM & 0x3f;
		}
		SNOOZE = false;
		AMR = 0x7f;
		commandLed(1);
		commandLed(0xC4);
		wordWrite("Alarm OFF");
		wait(1.5);
	}
	ALARM_ON = !ALARM_ON;
	GPREG0 = (AMR << 11) | (ALHOUR << 6) | ALMIN;
}

void MAIN_MENU(void){
```

```
        unsigned int time = CTIME0;
        #define timeInSeconds ((time) & 0x3F)
        #define timeInMinutes ((time >> 8) & 0x3F)
        #define timeInHours ((time >> 16) & 0x1F)

        if(timeInHours >= 12){
                if(timeInHours!=12)
                        time = (time & 0xFFE0FFFF) | ((timeInHours - 12) << 16);
                PM = true;
        }
        else
                PM = false;
        unsigned int digit = (1 << 20) | (PM << 19) | (timeInHours/10 << 18) |
(timeInHours%10 << 14) | (timeInMinutes/10 << 11) | (timeInMinutes%10 << 7) |
(timeInSeconds/10 << 4) | (timeInSeconds%10);

        commandLed(1);
        if(!ALARM_ACTIVE){
                if(PAGE1){
                        commandLed(0xC4);
                        printTime(digit);
                        if(SNOOZE){
                                commandLed(0xE3);
                                wordWrite("Zzz");
                        }
                        if(ALARM_ON){
                                commandLed(0xE6);
                                charWrite(2);
                        }
                        commandLed(0xE7);
                        charWrite(0);
                }
                else{
                        wordWrite("A: Set Time");
                        commandLed(0xC0);
                        wordWrite("B: Set Alarm");
                        commandLed(0x94);
                        wordWrite("C: On/Off (");
                        if(ALARM_ON)
                                wordWrite("ON)");
                        else
                                wordWrite("OFF)");
                        commandLed(0xD4);
```

```
                        wordWrite("D: Set Snooze");
                        if(SNOOZE){
                                commandLed(0xE3);
                                wordWrite("Zzz");
                        }
                        if(ALARM_ON){
                                commandLed(0xE6);
                                charWrite(2);
                        }
                        commandLed(0xE7);
                        charWrite(1);
                }
        }
        else{
                commandLed(0xC4);
                printTime(digit);
                unsigned static int counter = 0;
                unsigned static int stickman = 6;
                commandLed(0x80 + counter++);
                charWrite(stickman++);
                if(counter == 20)
                        counter = 0;
                if(stickman == 8)
                        stickman = 6;

                commandLed(0x94);
                wordWrite("Press D for OFF");
                commandLed(0xD4);
                wordWrite("Press OTHER 4 SNOOZE");
        }
}

void alarm(void){
        ALARM_ACTIVE = true;
        PWM _alarm(PWM1_1);
        _alarm.setFrequency(770);
        _alarm = 0.5;
        char off = 0;

        while(off == 0){
                commandLed(1);
                _alarm.setFrequency(960);
                wait(0.2);
```

```
                if((off = KEYPUSHED) != 0)
                        break;
                MAIN_MENU();
                _alarm.setFrequency(770);
                wait(0.2);
                off = KEYPUSHED;
        }
        _alarm = 0;

        if(off != 'D'){
                if(!SNOOZE_ALREADY){
                        OLD_ALARM = (ALHOUR << 6) | ALMIN;
                        SNOOZE_ALREADY = true;
                }
                if(ALMIN + SNOOZE_TIME >= 60){
                        if(ALHOUR + 1 == 24)
                                ALHOUR = 0;
                        else
                                ALHOUR++;
                        ALMIN = ALMIN + SNOOZE_TIME - 60;
                }
                else
                        ALMIN = ALMIN + SNOOZE_TIME;
                SNOOZE = true;
                off = KEYPUSHED;
        }
        else{
                SNOOZE = false;
                SNOOZE_ALREADY = false;
                ALMIN = OLD_ALARM & 0x3f;
                ALHOUR = (OLD_ALARM >> 6);
                toggleAlarm();
        }
        while(off == KEYPUSHED){}

        ILR |= (1 << 1);      // Clear Alarm
        ALARM_ACTIVE = false;
}

void clockSetup(void){
        AMR = (GPREG0 >> 11);
        ALHOUR = (GPREG0 >> 6) & 0x1f;
        ALMIN = (GPREG0 >> 0) & 0x3f;
```

```
        if(AMR == 0xf8)
                ALARM_ON = true;
        else{
                AMR = 0x7f;
                ALARM_ON = false;
        }
        OLD_ALARM = (ALHOUR << 6) | ALMIN;
        CCR = 0b10010; // Disable clock[0], reset CTC[1], cal counter disabled[4]
        CCR = 0b10000; // Reset CTC[1] is removed
        ALSEC = 0;
        CCR = 0b10001; // Enable clock[0], cal counter disabled[4]
}

void printTime(unsigned int digit){
        if(((digit >> 18)&1) == 0 && ((digit >> 14)&0xf) == 0){
                digit &= 0x83FFF;
                digit|= (1 << 18) | (2 << 14);
        }
        charWrite(((digit >> 18) & 1) + 48);
        charWrite(((digit >> 14) & 0xf) + 48);
        charWrite(58);
        charWrite(((digit >> 11) & 0x7) + 48);
        charWrite(((digit >> 7) & 0xf) + 48);
        if(digit >> 20){
                charWrite(58);
                charWrite(((digit >> 4) & 0x7) + 48);
                charWrite(((digit >> 0) & 0xf) + 48);
        }
        if((digit >> 19) & 1)
                wordWrite(" PM");
        else
                wordWrite(" AM");
}

void setAlarmSub(unsigned int digit){
        commandLed(1);
        wordWrite("Change alarm time");
        commandLed(0xC0);
        wordWrite("Current: ");
        printTime(digit);
        commandLed(0x94);
}
```

```
void setAlarm(void){
	while(KEYPUSHED=='B'){
		while(KEYPUSHED == 'B'){};
		commandLed(0xD);	// Set cursor blinking
		char hrDig10temp = 0;
		char hrDig1temp = 0;
		char minDig10temp = 0;
		char minDig1temp = 0;
		char AMorPM;
		char pauser;
		bool STATUS_OF_TIME_OF_DAY = false;
		unsigned int tempTime = ALHOUR;
		if(tempTime >= 12){
			STATUS_OF_TIME_OF_DAY = true;
			if(tempTime > 12)
				tempTime -= 12;
		}
		unsigned int digit = (STATUS_OF_TIME_OF_DAY << 19) | ((tempTime/10)
<< 18) | ((tempTime%10) << 14) | (ALMIN/10 << 11) | ((ALMIN%10) << 7);
		setAlarmSub(digit);
		wordWrite("XX:XX AM/PM");
		commandLed(0xD4);
		wordWrite("Press # to exit");
		commandLed(0x94);	// Set cursor at first X

		while(hrDig10temp < '0' || hrDig10temp > '1'){
			hrDig10temp = KEYPUSHED;
			pauser = hrDig10temp;
			if(hrDig10temp == '#')
				break;		// Exit time change w/o changes
		}
		if(hrDig10temp == '#')
			break;		// Exit time change w/o changes

		setAlarmSub(digit);
		charWrite(hrDig10temp);
		wordWrite("X:XX AM/PM");
		commandLed(0xD4);
		wordWrite("Press # to exit");
		commandLed(0x95);

		char LimitDigitOne = 0;
		if(hrDig10temp == '1')
```

```
        LimitDigitOne = '2';
else
        LimitDigitOne = '9';
while(hrDig1temp < '0' || hrDig1temp > LimitDigitOne){
        while(KEYPUSHED == pauser){}
        hrDig1temp = KEYPUSHED;
        pauser = hrDig1temp;
        if(hrDig1temp == '#')
                break;                // Exit time change w/o changes
}
if(hrDig1temp == '#')
        break;                // Exit time change w/o changes

setAlarmSub(digit);
charWrite(hrDig10temp);
charWrite(hrDig1temp);
wordWrite(":XX AM/PM");
commandLed(0xD4);
wordWrite("Press # to exit");
commandLed(0x97);

while(minDig10temp < '0' || minDig10temp > '5'){
        while(KEYPUSHED == pauser){}
        minDig10temp = KEYPUSHED;
        pauser = minDig10temp;
        if(minDig10temp == '#')
                break;                // Exit time change w/o changes
}
if(minDig10temp == '#')
        break;                // Exit time change w/o changes

setAlarmSub(digit);
charWrite(hrDig10temp);
charWrite(hrDig1temp);
charWrite(58);
charWrite(minDig10temp);
wordWrite("X AM/PM");
commandLed(0xD4);
wordWrite("Press # to exit");
commandLed(0x98);


while(minDig1temp < '0' || minDig1temp > '9'){
```

```
        while(KEYPUSHED == pauser){}
        minDig1temp = KEYPUSHED;
        pauser = minDig1temp;
        if(minDig1temp == '#')
                break;                // Exit time change w/o changes
}
if(minDig1temp == '#')
        break;                // Exit time change w/o changes

setAlarmSub(digit);
charWrite(hrDig10temp);
charWrite(hrDig1temp);
charWrite(58);
charWrite(minDig10temp);
charWrite(minDig1temp);
wordWrite(" 1:AM 2:PM _");
commandLed(0xD4);
wordWrite("Press # to exit");
commandLed(0xA4);
//commandLed(0x0c);
while(AMorPM < '1' || AMorPM > '2'){
        while(KEYPUSHED == pauser){}
        pauser = AMorPM;
        AMorPM = KEYPUSHED;
        if(AMorPM == '#')
                break;
}
if(AMorPM == '#')
        break;
if(AMorPM == '2'){
        if(((hrDig10temp - 48) == 1) && ((hrDig1temp-48) == 2))
                ALHOUR = 12;
        else
                ALHOUR = (hrDig10temp - 48)*10 + (hrDig1temp-48) + 12;

}
else{
        ALHOUR = (hrDig10temp - 48)*10 + (hrDig1temp-48);
        if(ALHOUR == 12)
                ALHOUR = 0;
}
ALMIN = (minDig10temp - 48)*10 + (minDig1temp-48);
OLD_ALARM = (ALHOUR << 6) | ALMIN;
```

```
                unsigned int amrtemp = AMR;
                amrtemp &= 0x7;
                GPREG0 = (AMR << 11) | (ALHOUR << 6) | ALMIN;
                if(!ALARM_ON)
                        toggleAlarm();
        }
        commandLed(0x0c);
}

void setTime(void){
        while(KEYPUSHED == 'A'){
                while(KEYPUSHED == 'A'){};
                char hrDig10temp = 0;
                char hrDig1temp = 0;
                char minDig10temp = 0;
                char minDig1temp = 0;
                char pauser;

                commandLed(1);
                wordWrite("Change current time");
                commandLed(0xC0);
                wordWrite("XX:XX AM/PM");
                commandLed(0x94);
                wordWrite("Press # to exit");
                commandLed(0xD);      // Set cursor blinking
                commandLed(0xC0);     // Move cursor to X
                while(hrDig10temp < '0' || hrDig10temp > '1'){
                        hrDig10temp = KEYPUSHED ;
                        pauser = hrDig10temp;
                        if(hrDig10temp == '#')
                                break;                // Exit time change w/o changes
                }
                if(hrDig10temp == '#')
                        break;                // Exit time change w/o changes

                commandLed(1);
                wordWrite("Change current time");
                commandLed(0xC0);
                charWrite(hrDig10temp);
                wordWrite("X:XX AM/PM");
                commandLed(0x94);
                wordWrite("Press # to exit");
                commandLed(0xC1);     // Move cursor to second X
```

```
char LimitDigitOne = 0;
if(hrDig10temp == '1')
	LimitDigitOne = '2';
else
	LimitDigitOne = '9';
while(hrDig1temp < '0' || hrDig1temp > LimitDigitOne){
	while(KEYPUSHED == pauser){};
	hrDig1temp = KEYPUSHED ;
	pauser = hrDig1temp;
	if(hrDig10temp == '#')
		break;		// Exit time change w/o changes
}
if(hrDig1temp == '#')
	break;		// Exit time change w/o changes

commandLed(1);
wordWrite("Change current time");
commandLed(0xC0);
charWrite(hrDig10temp);
charWrite(hrDig1temp);
wordWrite(":XX AM/PM");
commandLed(0x94);
wordWrite("Press # to exit");
commandLed(0xC3);	// Move cursor to X

while(minDig10temp < '0' || minDig10temp > '5'){
	while(KEYPUSHED ==pauser){}
	minDig10temp = KEYPUSHED ;
	pauser = minDig10temp;
	if(minDig10temp == '#')
		break;		// Exit time change w/o changes
}
if(minDig10temp == '#')
	break;		// Exit time change w/o changes

commandLed(1);
wordWrite("Change current time");
commandLed(0xC0);
charWrite(hrDig10temp);
charWrite(hrDig1temp);
charWrite(58);
charWrite(minDig10temp);
wordWrite("X AM/PM");
```

```
commandLed(0x94);
wordWrite("Press # to exit");
commandLed(0xC4);     // Move cursor to second X

while(minDig1temp < '0' || minDig1temp > '9'){
        while(KEYPUSHED == pauser){}
        pauser = 'x';
        minDig1temp = KEYPUSHED ;
        if(minDig10temp == '#')
                break;                // Exit time change w/o changes
}
if(minDig1temp == '#')
        break;                // Exit time change w/o changes
pauser = minDig1temp;
commandLed(1);
wordWrite("Change current time");
commandLed(0xC0);
charWrite(hrDig10temp);
charWrite(hrDig1temp);
charWrite(58);
charWrite(minDig10temp);
charWrite(minDig1temp);
wordWrite(" AM/PM");
commandLed(0x94);
wordWrite("1: AM      2: PM");
commandLed(0xD4);
wordWrite("Press # to exit");
commandLed(0x0C);
char TimeOfDay = 0;   // AM or PM

while(TimeOfDay < '1' || TimeOfDay > '2'){
        while(KEYPUSHED == pauser){}
        pauser = 'x';
        TimeOfDay = KEYPUSHED;
        if(TimeOfDay == '#')
                break;
}
if(TimeOfDay == '#')
        break;

CCR = 0b10010;
CCR = 0b10000;
```

```
                if(TimeOfDay == '1'){
                        if(hrDig10temp == '1' && hrDig1temp == '2'){
                                hrDig10temp = '0';
                                hrDig1temp = '0';
                        }
                        HOUR = (hrDig10temp - 48)*10 + (hrDig1temp-48);
                }
                else if(TimeOfDay == '2'){
                        if(hrDig10temp == '1' && hrDig1temp == '2'){
                                HOUR = (hrDig10temp - 48)*10 + (hrDig1temp-48);
                        }
                        else
                                HOUR = (hrDig10temp - 48)*10 + (hrDig1temp-48) + 12;
                }


                MIN = (minDig10temp - 48)*10 + (minDig1temp-48);
                SEC = 0;
                CCR = 0b10001;
        }
        commandLed(0x0c);
}
```

❖ LCP1769 Definitions

```
/*
 * LPC1769_MEM_DEFINITIONS.h
 *
 *  Created on: Apr 3, 2020
 *      Author: mjrx7
 */

#ifndef LPC1769_RTC_DEFINITIONS_H_
#define LPC1769_RTC_DEFINITIONS_H_

#define SEC (*(volatile unsigned int *) 0x40024020)
#define MIN (*(volatile unsigned int *) 0x40024024)
#define HOUR (*(volatile unsigned int *) 0x40024028)
#define CTIME0 (*(volatile unsigned int *) 0x40024014)
#define CCR (*(volatile unsigned int *) 0x40024008)

#define ALSEC (*(volatile unsigned int *) 0x40024060)
#define ALMIN (*(volatile unsigned int *) 0x40024064)
```

```
#define ALHOUR (*(volatile unsigned int *) 0x40024068)
#define AMR (*(volatile unsigned int *) 0x40024010)
// Mask year,mon,doy,dow,dom,hour,min,sec
#define ILR (*(volatile unsigned int *) 0x40024000)
#define ALARM_INTERRUPT ((ILR >> 1) & 1)

#define T0TCR (*(volatile unsigned int *) 0x40004004)
#define T0MR0 (*(volatile unsigned int *) 0x40004018)
#define T0MCR (*(volatile unsigned int *) 0x40004014)
#define T0TCR (*(volatile unsigned int *) 0x40004004)
#define ISER0 (*(volatile unsigned int *) 0xE000E100)
#define T0TC (*(volatile unsigned int *) 0x40004008)
#define T0IR (*(volatile unsigned int *) 0x40004000)

#define FIO0DIR (*(volatile unsigned int *)0x2009c000)
#define FIO0PIN (*(volatile unsigned int *)0x2009c014)

#define ISER0 (*(volatile unsigned int *) 0xE000E100)
#define IO0IntEnR (*(volatile unsigned int *) 0x40028090)
#define IO0IntEnF (*(volatile unsigned int *) 0x40028094)
#define IO0IntClr (*(volatile unsigned int *) 0x4002808C)
#define IO0IntStatR (*(volatile unsigned int *) 0x40028084)
#define IO0IntStatF (*(volatile unsigned int *) 0x40028088)
#define IOIntStatus (*(volatile unsigned int *) 0x40028080)

#define GPREG0 (*(volatile unsigned int *) 0x40024044)

#endif /* LPC1769_RTC_DEFINITIONS_H_ */
```

❖ HD44780

```
/*
 * HD44780.h
 *
 *  Created on: Apr 1, 2020
 *      Author: mjrx7
 */

#ifndef HD44780_H_
#define HD44780_H_
```

```
void BusoutWrite(unsigned int);
void commandLed(unsigned int);
void charWrite(unsigned int);
void setupHD44780(void);
void wordWrite(char*);

#endif /* HD44780_H_ */
```

```
/*
 * HD44780.cpp
 *
 *  Created on: Apr 1, 2020
 *      Author: mjrx7
 */
#include "TimerLPC.h"

// Registers for port 0
#define FIO0DIR (*(volatile unsigned int *)0x2009c000)
#define FIO0PIN (*(volatile unsigned int *)0x2009c014)

void BusoutWrite(unsigned int);
void commandLed(unsigned int);
void charWrite(unsigned int);
void setupHD44780(void);
void wordWrite(char*);

/*
 * RS p0.9
 * E p0.8
 * bits lsb - msb
 * p0.7,p0.6,p0.0,p0.1,p0.18,p0.17,p0.15,p0.16
 *
 */
const int BusoutBits[8] = {7,6,0,1,18,17,15,16};

void wordWrite(char* word){
	for(int i = 0; word[i] != '\0'; i++)
		charWrite(word[i]);
}
```

```
void setupHD44780(void){
	/*
	 * Setup Pins
	 * RS: p0.9
	 * En: p0.8
	 * data bits: lsb 7,6,0,1,18,17,15,16 msb
	 */
	wait_ms(20);          // Display needs 10 ms to come online
	FIO0DIR |= (0x783C3);
	commandLed(0x38);     // Function Set
	commandLed(0x0C);     // Display ON/OFF control
	commandLed(0x06);     // Entry mode set
	commandLed(0x01);     // Clears display
	wait_ms(400);


	// Custome character setup
	commandLed(0x40);
	//Up Arrow(0)
	charWrite(0x4);
	charWrite(0x4);
	charWrite(0x4);
	charWrite(0x4);
	charWrite(0x1f);
	charWrite(0xE);
	charWrite(0x4);
	charWrite(0x0);
	// Down Arrow(1)
	charWrite(0x0);
	charWrite(0x4);
	charWrite(0xE);
	charWrite(0x1f);
	charWrite(0x4);
	charWrite(0x4);
	charWrite(0x4);
	charWrite(0x4);
	// Alarm Bell(2)
	charWrite(0x4);
	charWrite(0xe);
	charWrite(0xe);
	charWrite(0xe);
	charWrite(0xe);
	charWrite(0x1f);
```

```
charWrite(0x4);
charWrite(0x0);
// Cat eye(3)
charWrite(0b00000);
charWrite(0b00000);
charWrite(0b01110);
charWrite(0b01010);
charWrite(0b01110);
charWrite(0b00000);
charWrite(0b00000);
charWrite(0b00000);
// \ character(4)
charWrite(0b00000);
charWrite(0b10000);
charWrite(0b01000);
charWrite(0b00100);
charWrite(0b00010);
charWrite(0b00001);
charWrite(0b00000);
charWrite(0b00000);
// ~ Character(5)
charWrite(0b00000);
charWrite(0b00000);
charWrite(0b01000);
charWrite(0b10101);
charWrite(0b00010);
charWrite(0b00000);
charWrite(0b00000);
charWrite(0b00000);
// Stick man 1 (6)
charWrite(0b01110);
charWrite(0b01010);
charWrite(0b01110);
charWrite(0b00101);
charWrite(0b01110);
charWrite(0b10100);
charWrite(0b01010);
charWrite(0b01001);
// Stick man 1 (7)
charWrite(0b01110);
charWrite(0b01010);
charWrite(0b01110);
charWrite(0b10100);
```

```
        charWrite(0b01111);
        charWrite(0b00100);
        charWrite(0b00111);
        charWrite(0b11001);

        commandLed(0);
        // Save custom character
}

void charWrite(unsigned int data){
        BusoutWrite(data & 0xff);
        FIO0PIN |= (1 << 9);
        FIO0PIN |= (1 << 8);
        FIO0PIN &= ~(1 << 8);
        wait_us(100);
}

void commandLed(unsigned int cmd){
        BusoutWrite(cmd);
        FIO0PIN &= ~(1 << 9);
        FIO0PIN |= (1 << 8);
        FIO0PIN &= ~(1 << 8);
        wait_us(100);
        if(cmd==1)
                wait_ms(2);
}

void BusoutWrite(unsigned int x){
        for(int i=0; i < 8; i++){
                if((x >> i) & 1)
                        FIO0PIN |= (1 << BusoutBits[i]);
                else
                        FIO0PIN &= ~(1 << BusoutBits[i]);
        }
}
```

❖ Keypad

```
/*
 * KeyPad.h
 *
 *  Created on: Apr 1, 2020
 *      Author: mjrx7
 */

#ifndef KEYPAD_H_
#define KEYPAD_H_

char keyPress(void);
void setupKeyPad(void);

#endif /* KEYPAD_H_ */
```

```
/*
 * KeyPad.cpp
 *
 *  Created on: Apr 1, 2020
 *      Author: mjrx7
 */
#include "KeyPad.h"
// Registers for port 0
#define FIO0DIR (*(volatile unsigned int *)0x2009c000)
#define FIO0PIN (*(volatile unsigned int *)0x2009c014)

/*
 *
 * p0.27,p0.24,p0.25,p0.28   Inputs
 * p0.2,p0.3,p0.21,p0.22     Outputs
 *
 */
const int inputs[] = {27,24,25,28};
const int outputs[] = {2,3,21,22};

char keyPress(void){
       // Column 1
       FIO0PIN &= ~(1 << outputs[0]);              // Set p0.2 = 0
       FIO0PIN |= (1 << outputs[1]);// Set p0.21,p0.22,p0.3 = 1
       FIO0PIN |= (1 << outputs[2]);
       FIO0PIN |= (1 << outputs[3]);
```

```
if(!((FIO0PIN >> inputs[0]) & 1))
        return '1';
else if(!((FIO0PIN >> inputs[1]) & 1))
        return '4';
else if(!((FIO0PIN >> inputs[2]) & 1))
        return '7';
else if(!((FIO0PIN >> inputs[3]) & 1))
        return '*';

// Column 2
FIO0PIN &= ~(1 << outputs[1]);              // Set p0.3 = 0
FIO0PIN |= (1 << outputs[2]);// Set p0.21,p0.22,p0.2 = 1
FIO0PIN |= (1 << outputs[3]);
FIO0PIN |= (1 << outputs[0]);
if(!((FIO0PIN >> inputs[0]) & 1))
        return '2';
else if(!((FIO0PIN >> inputs[1]) & 1))
        return '5';
else if(!((FIO0PIN >> inputs[2]) & 1))
        return '8';
else if(!((FIO0PIN >> inputs[3]) & 1))
        return '0';

// Column 3
FIO0PIN &= ~(1 << outputs[2]);              // Set p0.21 = 0
FIO0PIN |= (1 << outputs[3]);// Set p0.2,p0.3,p0.22 = 1
FIO0PIN |= (1 << outputs[0]);
FIO0PIN |= (1 << outputs[1]);
if(!((FIO0PIN >> inputs[0]) & 1))
        return '3';
else if(!((FIO0PIN >> inputs[1]) & 1))
        return '6';
else if(!((FIO0PIN >> inputs[2]) & 1))
        return '9';
else if(!((FIO0PIN >> inputs[3]) & 1))
        return '#';

// Column 4
FIO0PIN &= ~(1 << outputs[3]);      // Set p0.22 = 0
FIO0PIN |= (1 << outputs[0]);// Set p0.21,p0.2,p0.3 = 1
FIO0PIN |= (1 << outputs[1]);
FIO0PIN |= (1 << outputs[2]);
if(!((FIO0PIN >> inputs[0]) & 1))
```

```
                return 'A';
        else if(!((FIO0PIN >> inputs[1]) & 1))
                return 'B';
        else if(!((FIO0PIN >> inputs[2]) & 1))
                return 'C';
        else if(!((FIO0PIN >> inputs[3]) & 1))
                return 'D';

        return 0;
}

void setupKeyPad(void){
        // Setup Keypad Pins
        for(int i = 0; i < 4; i++){
                FIO0DIR |= (1 << outputs[i]);
                FIO0DIR &= ~(1 << inputs[i]);
        }
}
```

❖ PWM

```
/*
 * PWM.h
 *
 *  Created on: Mar 17, 2020
 *      Author: mjrx7
 */

#ifndef PWM_H_
#define PWM_H_

typedef enum {
        PWM1_0,PWM1_1,PWM1_2,PWM1_3,PWM1_4,PWM1_5
} PIN;

class PWM
{
protected:
        static bool runOnce;
        unsigned int _period_check;
        static unsigned int _period;
        static unsigned int _frequency;
        unsigned int _pulseWidth;
```

```
        float _dutyCycle;
        PIN _pin;
        void setupPWM(PIN pin);
public:
        PWM();
        PWM(PIN pin);
        PWM(const PWM& m);
        ~PWM();
        PWM &operator= (float value){setDC(value); return *this;}
        operator float(){return getDC();}
        float getDC();
        unsigned int getPeriod();
        unsigned int getPulseWidth();
        unsigned int getFrequency();
        void setFrequency(unsigned int freq);
        void setPeriod(unsigned int period);
        void setPulseWidth(unsigned int pulseWidth);
        void setDC(float DC);
};



#endif /* PWM_H_ */
```

```
 /*
          * PWM.cpp
          *
          *  Created on: Mar 17, 2020
          *      Author: mjrx7
          */

          #include "PWM.h"

          #define PCONP (*(volatile unsigned int *) 0x400FC0C4)
          #define PCLKSEL0 (*(volatile unsigned int *) 0x400FC1A8)
          #define PINSEL4 (*(volatile unsigned int *) 0x4002C010)
          #define PWM1PR (*(volatile unsigned int *) 0x4001800C)
          #define PWM1MR0 (*(volatile unsigned int *) 0x40018018)
          #define PWM1MR1 (*(volatile unsigned int *) 0x4001801C)
          #define PWM1MR2 (*(volatile unsigned int *) 0x40018020)
          #define PWM1MCR (*(volatile unsigned int *) 0x40018014)
          #define PWM1LER (*(volatile unsigned int *) 0x40018050)
          #define PWM1PCR (*(volatile unsigned int *) 0x4001804C)
```

```
#define PWM1TCR (*(volatile unsigned int *) 0x40018004)
#define PWM1TC (*(volatile unsigned int *) 0x40018008)

unsigned int PWM::_period;
unsigned int PWM::_frequency;
bool PWM::runOnce = false;

PWM::PWM()
{
	_period = 2000;
	_period_check = _period;
	_pulseWidth = 0;
	_frequency = 1000000 / _period;
	_dutyCycle = 0;
	//setupPWM();
}

PWM::PWM(PIN pin)
{
	_pin = pin;
	if(!runOnce){
		_period = 20000;
		if(_period==0)
			_frequency = 0;
		else
			_frequency = 1000000 / _period;
	}
	_period_check = _period;
	_pulseWidth = 0;//_period/2;
	_dutyCycle = 0;//(float)_pulseWidth / _period;
	setupPWM(pin);
}

PWM::~PWM()
{

}

unsigned int PWM::getPulseWidth(void)
{
	return _pulseWidth;
}
```

```
unsigned int PWM::getPeriod(void)
{
	return _period;
}

float PWM::getDC(void)
{
	return _dutyCycle;
}

unsigned int PWM::getFrequency(void)
{
	return _frequency;
}

void PWM::setFrequency(unsigned int freq){
	_frequency = freq;
	if(_frequency==0)
		_period = 0;
	else
		_period = 1000000 / freq;
	_period_check = _period;
	_pulseWidth = _dutyCycle * _period;
	PWM1MR0 = _period;
	PWM1MR1 = _pulseWidth;
	PWM1MR2 = _pulseWidth;
	PWM1LER = (1 << 0) | (1 << _pin);//Load the MR1 new value at
start of next cycle

}

void PWM::setPeriod(unsigned int period)
{
	_period = period;
	_period_check = _period;
	if(_period==0)
		_frequency = 0;
	else
		_frequency = 1000000 / _period;
	_pulseWidth = _dutyCycle * _period;
	PWM1MR0 = _period;
	PWM1MR1 = _pulseWidth;
	PWM1MR2 = _pulseWidth;
```

```
	PWM1LER = (1 << 0) | (1 << _pin);//Load the MR1 new value at
start of next cycle

}

void PWM::setPulseWidth(unsigned int pulseWidth)
{
	_pulseWidth = pulseWidth;
	_dutyCycle = 1. * _pulseWidth / _period;
	PWM1MR1 = _pulseWidth;
	PWM1MR2 = _pulseWidth;
	PWM1LER = (1 << 2) | (1<<1); //Load the MR1 new value at start of
next cycle
}

void PWM::setDC(float DC)
{
	if(DC != _dutyCycle  || _period_check != _period){
		if(DC > 1)
			DC = 1;
		else if(DC < 0)
			DC = 0;
		_dutyCycle = DC;
		_pulseWidth = _dutyCycle * _period;
		if(_pin == PWM1_1)
			PWM1MR1 = _pulseWidth;
		else if(_pin == PWM1_2)
			PWM1MR2 = _pulseWidth;
		PWM1LER = (1 << _pin);	//Load the MR1 new value at
start of next cycle
		if(_period_check != _period){
			PWM1TCR = (1 << 1);   // Reset PWM TC & PR
			PWM1TCR = (1 << 0) | (1 << 3);	// Enable
counters & PWM mode
			_period_check = _period;
		}
	}
}

void PWM::setupPWM(PIN pin)
{
	/*
```

```
        * Need to correct PINSEL4, write now sets pin mode for PWM1.1
and PWM1.2
        * auto to PWM mode. Need to make optional depending on object
being initialized
        *
        */

       if(runOnce == false){
              PCONP |= (1 << 6);    // Power on PWM default ON
              PCLKSEL0 &= ~(1 << 12);      // Divide by 4 default
divide/4
              PINSEL4 |= (0b01 << 0) | (0b01 << 2);       // Set pin 42
to PWM 1.1  pin 43 tp PWM 1.2

              PWM1PCR = 0;   // Sets single edge PWM
              PWM1PR = 0;           // Prescale Register Amount to
increment TC counter by in addition
              runOnce = true;
       }
       PWM1MR0 = _period;
       if(pin==PWM1_1)
              PWM1MR1 = _pulseWidth;
       else if(pin==PWM1_2)
              PWM1MR2 = _pulseWidth;
       PWM1MCR = (1 << 1);
       // Reset PWM TC on PWM1MR0 match
       PWM1LER = (1 << 0) | (1 << pin);                    // Update
values in MR0 & MR1

       PWM1PCR |= (1 << (pin+8));
       /*if(pin==PWM1_1)
              PWM1PCR |= (1 << 9);                                 //
Enable PWM output
       else if(pin==PWM1_2)
              PWM1PCR |= (1 << 10);*/

       PWM1TCR = (1 << 1);
       // Reset PWM TC & PR
       PWM1TCR = (1 << 0) | (1 << 3);                              //
Enable counters & PWM mode
}
```

❖ Timer

```
/*
 * TimerLPC.h
 *
 *  Created on: Feb 24, 2020
 *      Author: Matthew Johnson & Talia Nguyen
 */

#ifndef TIMERLPC_H_
#define TIMERLPC_H_

void timerStart(void);
void timerStop(void);
void timerReset(void);
int timerRead_us(void);
float timerRead_ms(void);
float timerRead(void);
void wait_us(int);
void wait_ms(float);
void wait(float);

#endif /* TIMERLPC_H_ */
```

```
/*
 * TimerLPC.cpp
 *
 *  Created on: Feb 24, 2020
 *      Author: Matthew Johnson & Talia Nguyen
 */
#define T0TCR (*(volatile unsigned int *) 0x40004004)
#define T0TC (*(volatile unsigned int *) 0x40004008)

void timerStart(void){
	T0TCR |= (1 << 0);
}

void timerStop(void){
	T0TCR &= ~(1 << 0);
}
```

```
void timerReset(void){
	T0TCR |= (1 << 1);
	while(T0TC != 0){}
	T0TCR &= ~(1 << 1);
}

int timerRead_us(void){
	return T0TC;
}

float timerRead_ms(void){
	return (float)timerRead_us()/1000;
}

float timerRead(void){
	return timerRead_us()/1000000.0;
}

void wait_us(int us){
	int startTime;
	timerStart();
	startTime = timerRead_us();
	while((timerRead_us() - startTime) < us){}
}

void wait_ms(float s){
	wait_us(s * 985.0);//1000.0);
}

void wait(float s){
	wait_us(s * 985000.0);//1000000.0);
}
```