

Real Time Intelligent Home System

ECE 4220: Real-Time Embedded Computing

Project Report

Yixiang Gao

I. Abstract

In this project, a demo of intelligent housing system is displayed by using TS-7250, an auxiliary board from the Lab, and another auxiliary board with two different sensors. The processor will be able to communicate with both sensors and one button on the auxiliary board and then switch the red LED when the program is triggered. In the same time, the user can also send messages to the processor through the network to achieve certain operation. Therefore, two major parts are implemented in this project. First is the kernel space which reads the value from the sensor and switch the light. Second is the User space that communicates with the kernel and also creates a network communication to receive the command from outside network.

II. Introduction

The goal of this project is to express the idea of using different sensors with a computer system instead of the traditional physical switch to control all electronic devices at people's house in the very future. Imagine whenever someone walks into a room, the light will be turned on automatically. People do not need to use the remote controllers anymore but their voice to command different devices at their house. Moreover, people can also communicate with their home system outside through internet to achieve different activities even when they are not at home. The computer system will be the housekeeper and take care of everything for the house. All different kinds of sensors will be the communication point between the system and people.

III. Background

Similar implementation can be found on campus. For example, in the restroom, instead of pressing the button to turn on and off the hand dryer, a sensor is installed that can detect whether there are hands under the dryer or not. The same implementation for some tap water machine on campus too. Furthermore, the door of each building will have an electronic side that whenever a hand is waving close to the sensor next to the door, the system will open the door for the person automatically. Nowadays, most buildings all have electronic control system with different sensors installed to help people in many different ways.

IV. Implementation

The overall system is composed of a Linux PC, a TS-7250 single board, the auxiliary board, a microphone voice sensor, and another sensor that detects the touching.

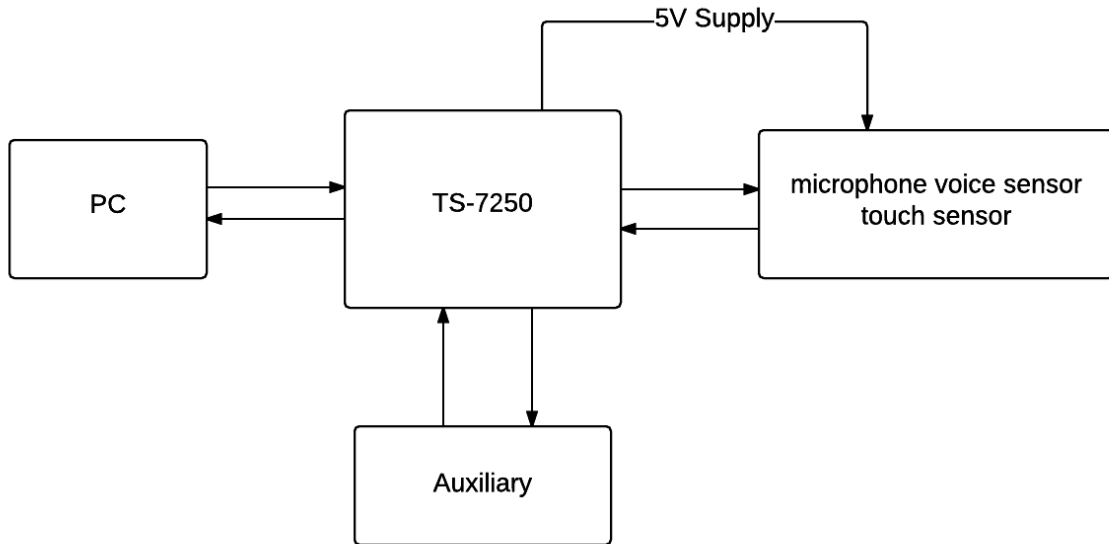


Figure 1: Hardware System Diagram

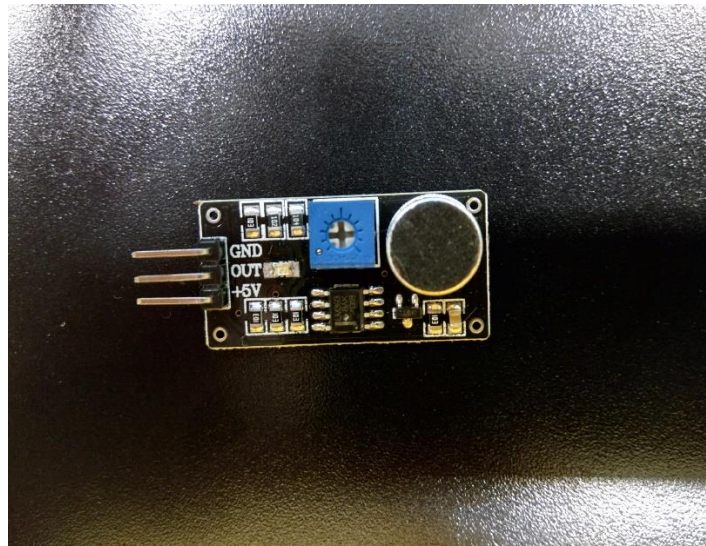


Figure 2: Microphone Sensor



Figure 3: Touch Sensor

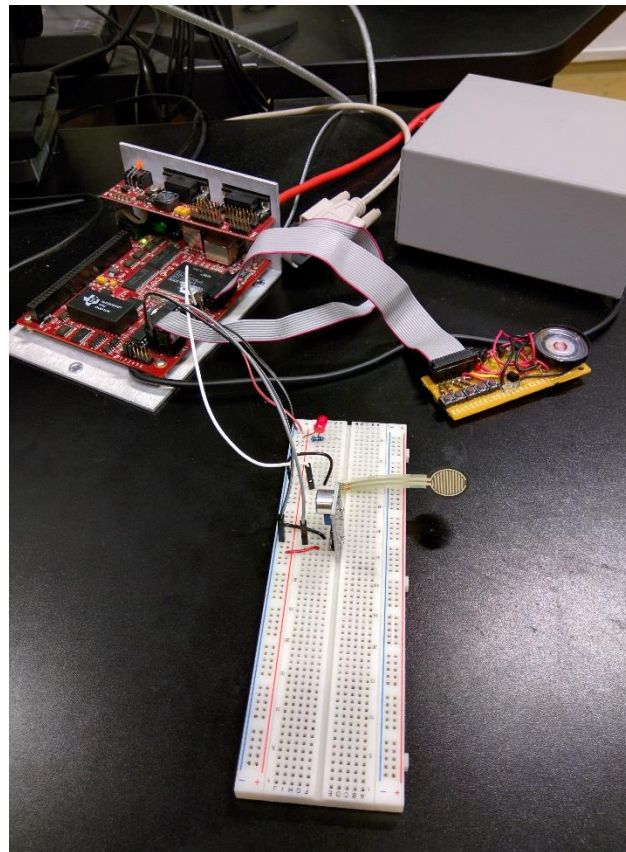


Figure 4: Hardware Implementation

The TS-7250 will constantly read the input from three different places. First is the output value from microphone sensor via the onboard MAX197 A/D Converter. It has an optional eight-channel, 12-bit converter with a conversion time of 12 us. The microphone used has three pins which are +5V, GND, and OUT. The +5V pin is supplied by 5V from the TS-7250 LCD power supply. The GND pin is connected to the ground on channel 0

of the ADC. The OUT pin, which has a voltage from 0V to 5V depends on the volume of the voice, is connected to channel 0 of the ADC.

Second is the value from touch sensor via LCD Header. Two pins on the sensor are normally disconnected. Whenever the round terminal is being touched, it will connect these two pins. For example, assume one pin is connected to the VCC which is 5V, when the sensor is not being touched, the other pin will output 0V. When the sensor is being touched, it will output 5V DC voltage. DIO lines LCD_0 thru LCD_7 are a byte-wide port using Port A on the EP9302. The port is interfacing to a 5V LCD, 1.0K Ohm resistors have been added in series between the EP9302 and the LCD_0 thru LCD_7 pins. In implementation, one pin of the touch sensor is connected to the ground and the other pin is connected to LCD_1 which is PortA1.

The third value is from a button on auxiliary board. The auxiliary board is connected to TS-7250 via the DIO1 Header. It has 5 buttons and 3 LEDs. In the implementation, only PortB0, which is one button, and PortB5, which is the red LED, are being used. Whenever, either of the button, the touch sensor, or the microphone is triggered, it will return the result of red LED switching on or off.

The 5V power supply is from LCD Header pin 1 and 2 with 2 wires connected on the breadboard to supply the sensor which makes the hardware implementation a lot easier.

Both the user space application and kernel module are written in C using Eclipse on the Linux workstations in the lab. The general software implementation may be seen in Figure 5. The client and server communication is based on UDP network connection. The kernel will send the information to the server via two FIFOs.

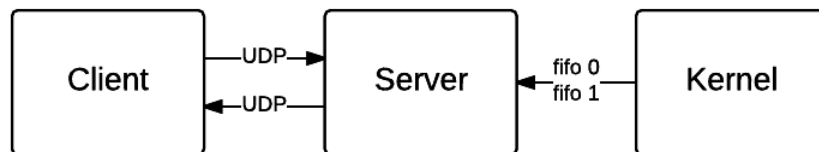


Figure 5: Software Implementation

For Client implementation, this project actually uses “*Lab6_Client*” which is provided in the lab. The reason is for Client, it does not have very much operation but sending different command message to the server. “*Lab6_Client*” is basically a UDP broadcast server which already has the function of sending message. It is much easier to implement in this way since the main focus of this project is to handle the communication between User space and Kernel space.

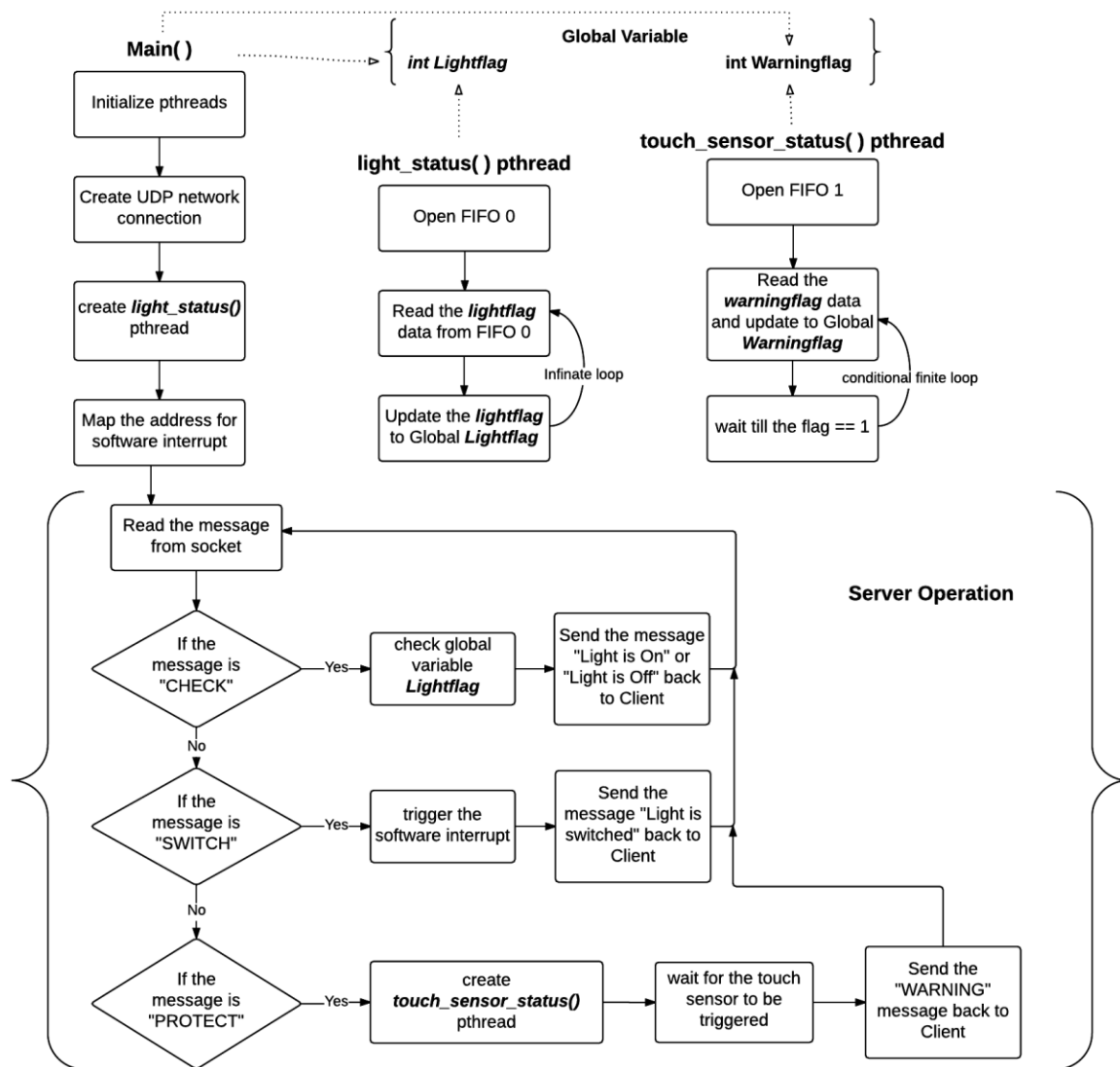


Figure 6: Server Implementation

The server operation uses the same structure from Lab 5 and Lab 6 server program. It will handle three different messages and then run the corresponding function depends on which message is received. Initially, the "SWITCH" function is proposed to be achieved by sending a message to kernel module through FIFO. However, it will require one more real time task running consistently in the kernel and consuming resources. Therefore the decision for using interrupts was made because it utilizes system resources more efficiently and, in fact, it is also easier to approach in this way.

Two different pthreads are created. They both communicate with different real time tasks in the kernel module. **light_status()** will be running for the whole time in the system to keep reading and updating the flag value for red LED via FIFO 0.

touch_sensor_status() will only be created when the server receives a “PROTECT” message, the function is similar to *light_status()* but it checks the flag value for the touch sensor output through a different FIFO. This pthread will destroy itself after the touch sensor is triggered once. In Preliminary Report, it was originally planned to use more pthreads. During the implementation, however, there are some repeat functions between those pthreads and some objects are better achieved by using interrupt. So the total pthread number reduces to two.

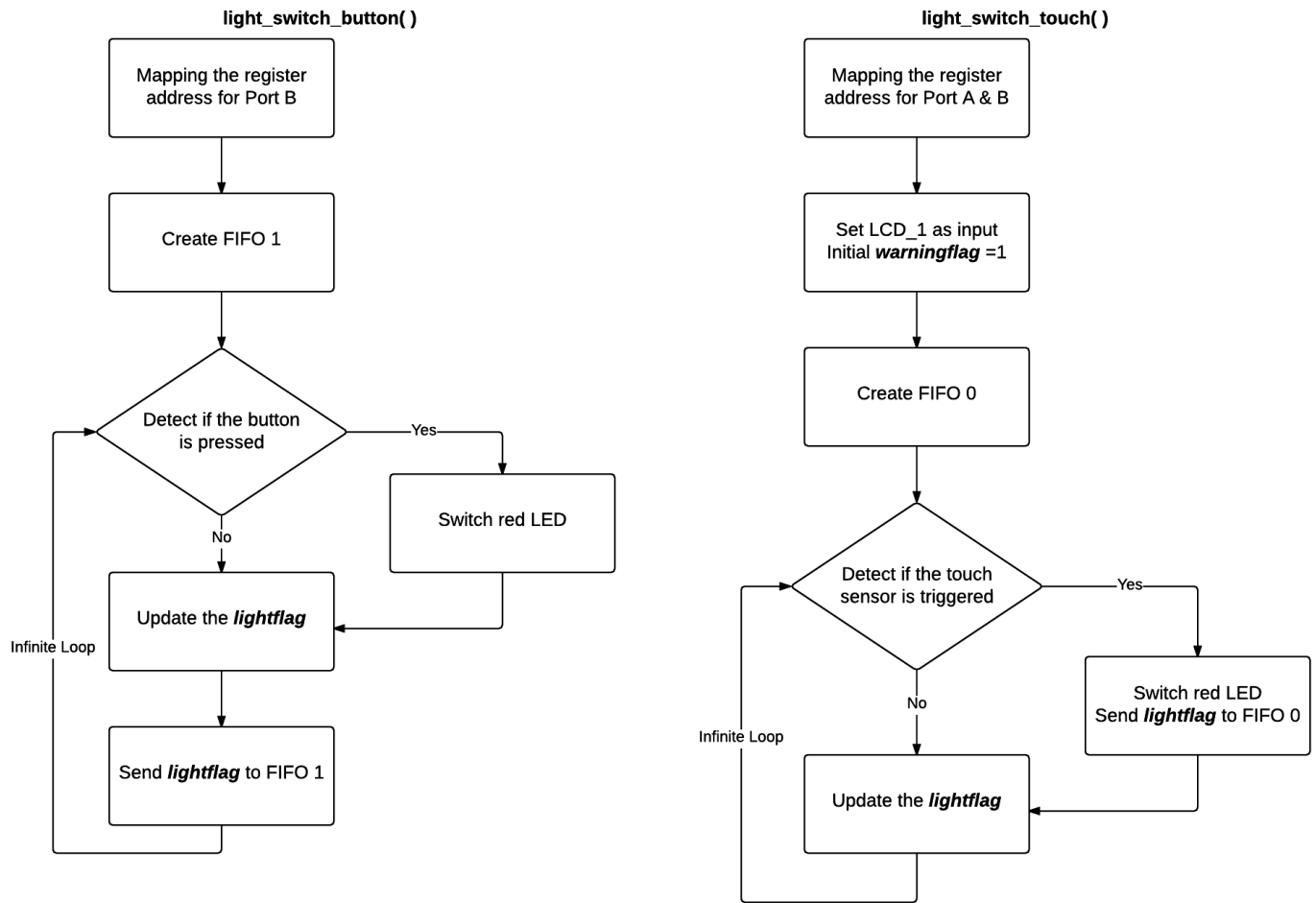


Figure 7: Kernel Implementation 1

Three real time tasks are created in the kernel module which are shown in Figure 7 & 8. Each Task corresponds to detecting either button, touch sensor, or microphone. After detecting operation is done, all tasks will update the *lightflag* information and only task *light_switch_button()* will send the *lightflag* into FIFO 1. This implementation will be further discussed in the Discussion section.

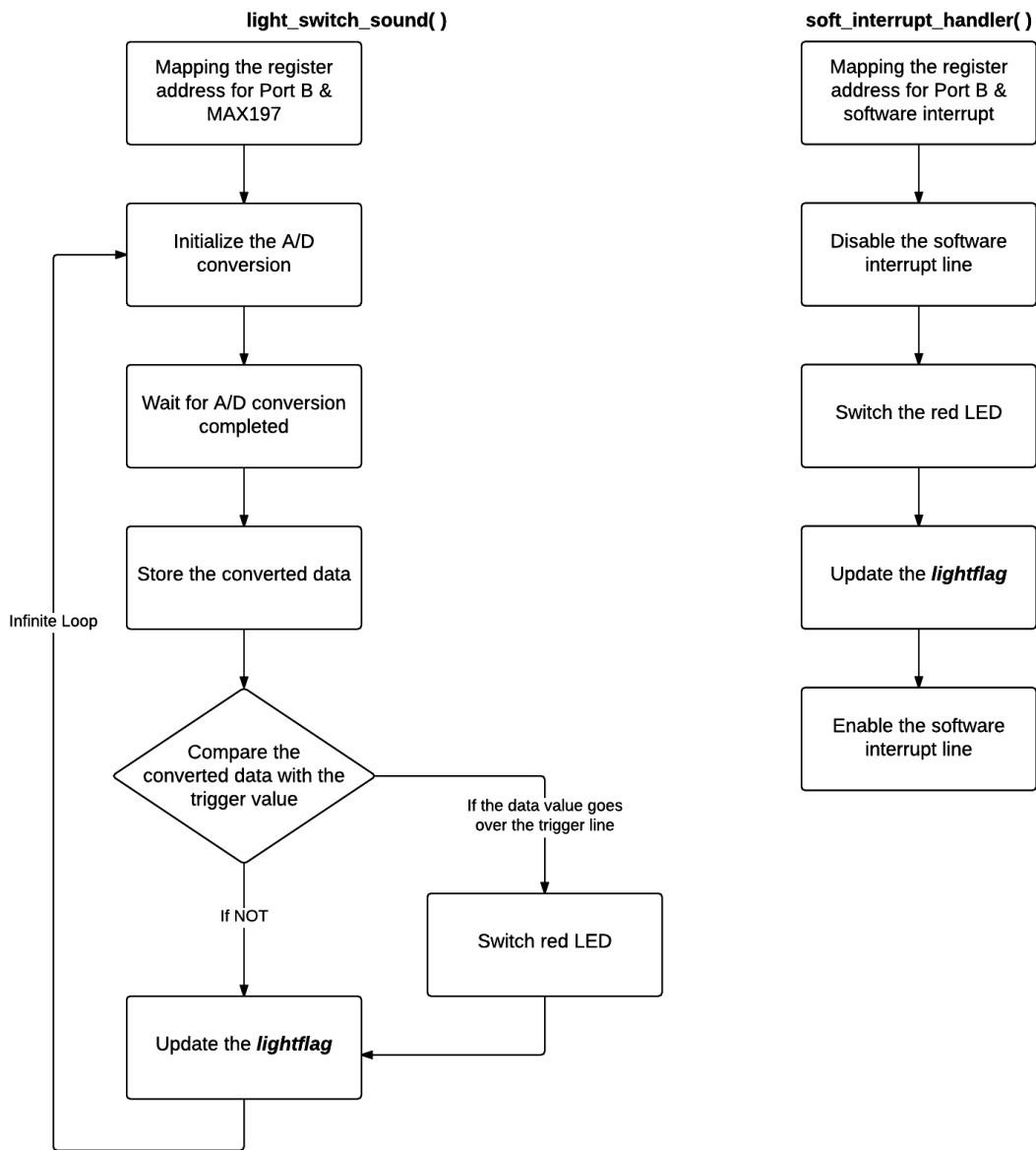


Figure 8: Kernel Implementation 2

The software interrupt handler is very straight forward. Whenever the interrupt is received, it switches the red LED and update the *lightflag* value.

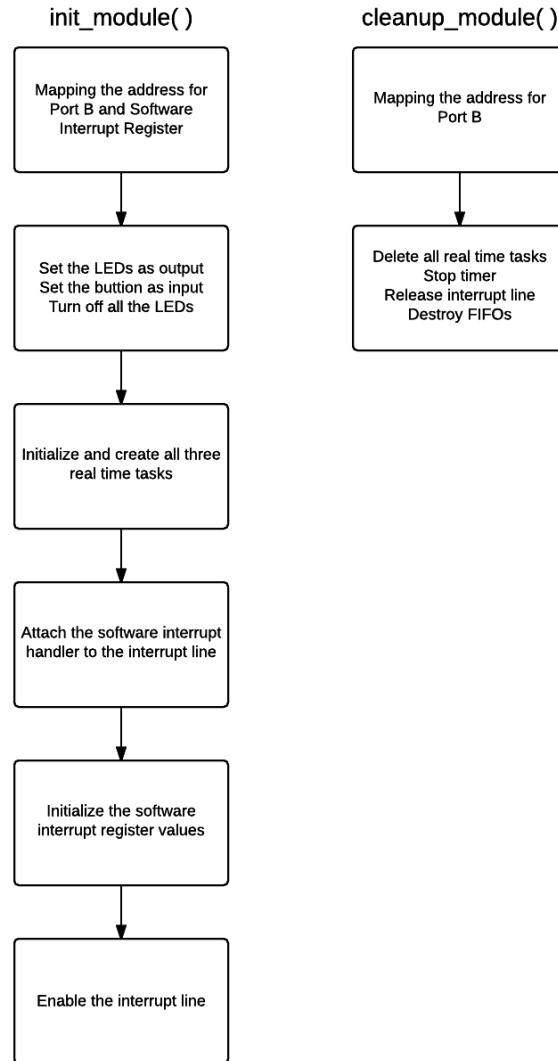


Figure 9: Kernel Implementation 3

V. Experiments and Results

Before the whole system is built on TS-7250, a few testing experiments were made to test the output signal for two sensors, microphone and touch sensor, since they are both assumed to be analog signal at the beginning of this project. And it was also proposed to use the A/D converter for both microphone and touch sensor. Therefore, it is necessary to know what the characteristic of the output analog signal is in order to handle the A/D conversion better.

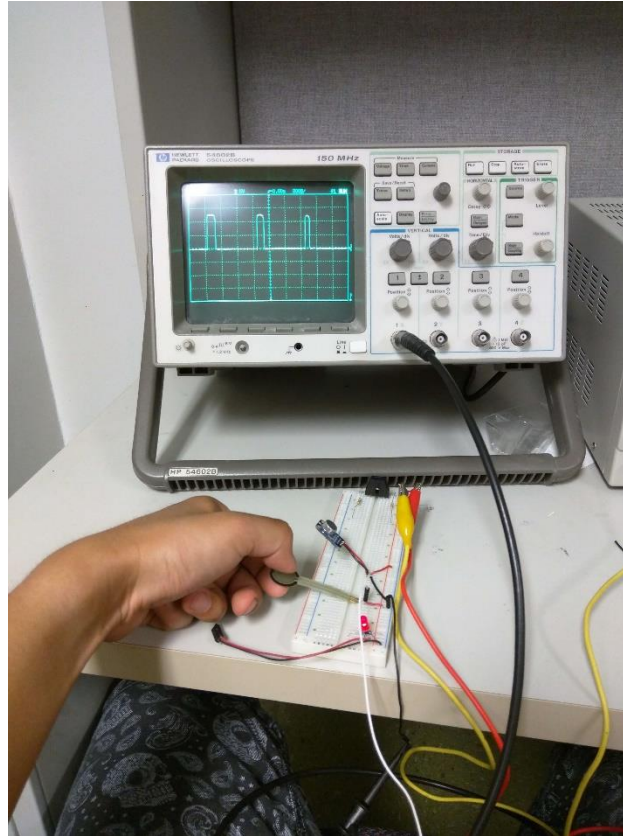


Figure 10: Testing for Touch Sensor

In Figure 10, the one pin of the touch sensor is connected to Vcc which is 5V, another pin is connected directly to the oscilloscope. Normally the output is 0V from the touch sensor, whenever it is being pressed, it will send the output 5V DC voltage. So the decision is made to use the 5V DIO Port to replace the A/D converter which is also much easier to code. During the implementation, at first, the touch sensor is connected to the Vcc and when it is triggered, it should send the 5V signal to the LCD_1. However, since there are 1K Ohm resistors in series between the EP9302 and the LCD_0 thru LCD_7 pins to prevent the LCD from overdriving the EP9302 Port A pins, the output bit from LCD_1 is always HIGH which is not preferred in this case. For modification, change the Vcc pin of touch sensor to GND. So whenever the sensor is triggered, the bit will become LOW which is easier to be detected.

```

fd_map = open("/dev/mem", O_RDWR | O_SYNC );

pAbase = (unsigned long *)mmap(NULL,
                               getpagesize(),
                               PROT_READ|PROT_WRITE,
                               MAP_SHARED,
                               fd_map,
                               0x80840000);

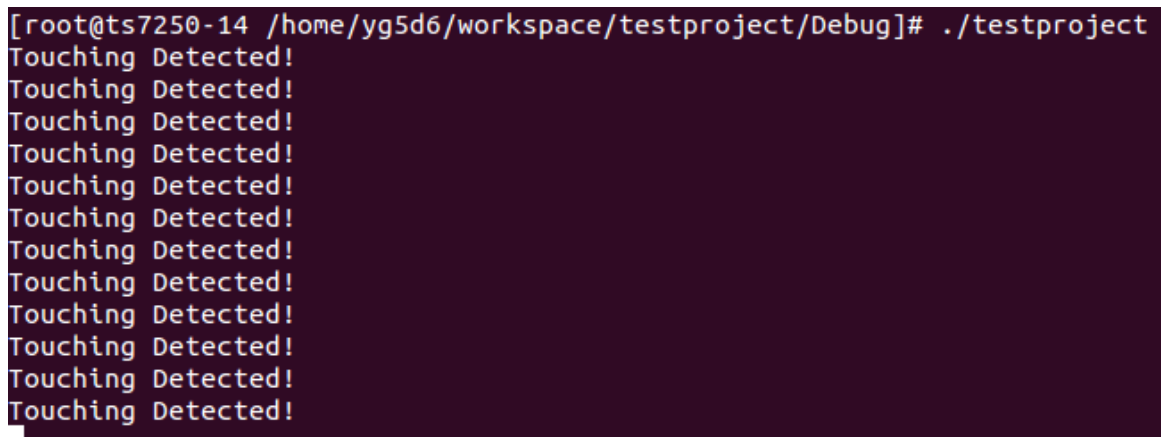
pAddr   = (unsigned long *)((char *)pAbase + 0x10);
pAdr    = (unsigned long *)((char *)pAbase + 0x00);

*pAddr &= 0xFFFFFFF0;

while(1)
{
    if(!(*pAdr & 0x02))
        printf("Touching Detected!\n");
    usleep(1000000);
}

```

Figure 11: Test Code for Touch Sensor



```

[root@ts7250-14 /home/yg5d6/workspace/testproject/Debug]# ./testproject
Touching Detected!
Touching Detected!
Touching Detected!
Touching Detected!
Touching Detected!
Touching Detected!
Touching Detected!
Touching Detected!
Touching Detected!
Touching Detected!
Touching Detected!
Touching Detected!

```

Figure 12: Screen Shot of Testing Touch Sensor

For testing the microphone sensor, a video was taken that shows the difference of its output signal when it detects voice. The video is included with other files. Basically, it will constantly output 5V DC value whenever the environment is quiet. If there are some voice being detected, the output will drop down. The voltage drop range depends on the volume of the voice.

When implement the microphone onto the board with using MAX197, another simple test is also created to overview the conversion data in order to decide the trigger line for the LED switch operation.

```

result: 2830
...
result: 141
...
result: 2818
...
result: 2818
...
result: 2829
...
result: 2830
...
result: 2828
...
result: 2818
...
result: 2829
...
result: 2828
...
result: 2819
...
result: 142

```

Figure 13: Screen Shot of Testing Microphone

Normally the converted value is around 2810 to 2830 which refers to the analog 5V output in a quiet environment. The 142 value (or 141) appears whenever some loud noise is made close to the microphone. Therefore, to make the trigger not very hard, the line is set at 2800.

For testing the whole system, four videos are taken to demonstrate four different parts of the system. Demonstration1 shows all the hardware function including touch trigger, voice trigger, and button press trigger from the kernel module. Demonstration 2,3 and 4 basically display three different responding operation when the server program receives the command from the client.

VI. Discussions and Conclusions

In general, this project is successful. Most main objects proposed have been achieved and the system overall works reasonable well. However, there are still many parts in the project can be improved.

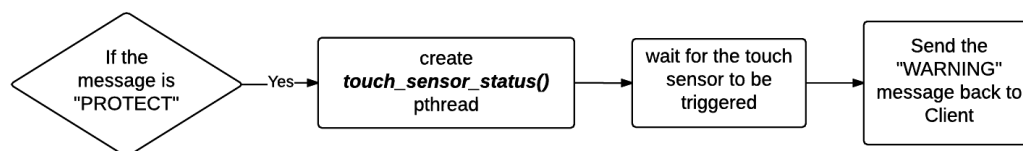


Figure 14: PROTECT Mode Implementation

For “PROTECT” mode implementation. When it is triggered, besides creating a new pthread, the server program also starts waiting for the trigger flag to be received from kernel module. During that time, the server program will not handle any message sent from the client which is not ideal. One solution is to also create another task which used for handing the waiting process to be able to let the server program keep running its other function.

In building the kernel module, there were four real time tasks created to run in the same time with the same priority. The forth real time task is used for updating the *lightflag* and sending the value to FIFO. However, every time the module is installed onto the board, the board system would crush. One guess is that TS-7250 cannot handle this many real time tasks in the same time. It is also possible to solve this problem by making a better schedule for all real time tasks but due to the time limitation for this project. It has not been achieved.

The voice detect via microphone is very insensitive. Usually the noise needs to be kept for a few seconds to be able to trigger the LED switch. Meanwhile, there were also some unexpected trigger from the microphone. One possible solution is to improve the A/D conversion which is from software side. Another guess is to use a voltage follower between output pin on the microphone and MAX197 to send a better analog signal to the A/D converter.

[illegible]

```

while(1)
{
    if(!(*pBdr & 0x1)) //whenever the button is pressed
    {
        *pBdr ^= 0x20; //switch the Red LED
    }

    if( *pBdr & 0x20 ) //when the switch operation is finished, check
        lightflag = 1; //the bit value for Red LED and change the flag
    else //value
        lightflag = 0;
    rt_sleep(nano2count(150000000));

    rtf_put(1, &lightflag, sizeof(lightflag)); //send the lightflag status //to
pthread1 at userspace through //fifo 1
}
}

void light_switch_touch(int t) //task for detecting the touch sensor
{
    unsigned long *pBbase, *pBdr; //registers for Port B
    unsigned long *pAddr, *pAdr; //registers for Port A

    pBbase = (unsigned long *)__ioremap(0x80840000,4096,0); //mapping the address
    pBdr = (unsigned long*)((char *)pBbase + 0x4);

    pAddr = (unsigned long*)((char *)pBbase + 0x10);
    pAdr = (unsigned long*)((char *)pBbase + 0x00);

    *pAddr &= 0xFFFFFFF0; //set PortA bit 1 as input to detect the touch sensor

    int warningflag = 1;
    rtf_create(0,sizeof(warningflag)); //create fifo 0 which used to communicate with
//another pthread in the userspace

    while(1)
    {
        if(!(*pAdr & 0x02)) //whenever the sensor is touchedcd
        {
            *pBdr ^= 0x20; //switch the Red light
            rtf_put(0, &warningflag, sizeof(warningflag)); //send the flag to pthread2
        }
        if( *pBdr & 0x20 ) //when the switch operation is finished, check
            lightflag = 1; //the bit value for Red LED and change the flag value
        else
            lightflag = 0;
    }
}

```

```

        rt_sleep(nano2count(150000000));
    }
}

void light_switch_sound(int t) //task for detecting the sound
{
    unsigned long *pBbase, *pBdr; //registers for Port B

    //Registers for using MAX197 A/D converter, there are two registers being used here, the
    //first is from address 0x10F0_0000 which initiate the A/D conversion and store the
    //converted data. The second is from address 0x1080_0000 which is used for checking if
    //the conversion is completed or not.

    volatile unsigned short *complete; //registers for completion check
    volatile unsigned char *lsb, *msb, *control; //control is for conversion initialization.
                                                //lsb, msb are for data storage.

    int res; //store the decimal number data after conversion

    pBbase = (unsigned long *)__ioremapped(0x80840000,4096,0); //mapping the address
    pBdr = (unsigned long*)((char *)pBbase + 0x4);
    lsb = control = (unsigned char *)__ioremapped(0x10f00000,4096,0);
    msb = lsb + 1; //least and most significant bit for store the binary data
    complete = (unsigned short *)__ioremapped(0x10800000,4096,0);

    while(1)
    {
        *control = 0x40; //Initiate conversion, channel 0,unipolar, 5V

        while((*complete & 0x80)!= 0) //wait for completion
        {
        }
        res = *lsb; //store result on a scale from 0 to 2^12-1
        res |= *msb << 8;
        if( res < 2800 ) //2800 is the base line for the sound signal to be able to trigger
                        //the switch
        {
            *pBdr ^= 0x20; //switch the Red LED
        }
        if( *pBdr & 0x20 ) //when the switch operation is finished, check
            lightflag = 1; //the bit value for Red LED and change the flag value
        else
            lightflag = 0;
        rt_sleep(nano2count(150000000));
    }
}

```

//Software interrupt handler will be trigger when the userspace receive a "SWITCH"

//command, then the interrupt will switch the value for the red LED bit.

```
static void soft_interrupt_handler(unsigned irq_num, void *cookie)
{
    unsigned long *VIC2SoftIntClear, *VIC2base; //software interrupt registers
    unsigned long *pBbase, *pBddr, *pBdr; //registers for Port B

    pBbase = (unsigned long *)__ioremmap(0x80840000,4096,0); //mapping the address
    pBddr = (unsigned long*)((char *)pBbase + 0x14);
    pBdr = (unsigned long*)((char *)pBbase + 0x4);

    VIC2base = (unsigned long *)__ioremmap(0x800C0000,4096,0);
    VIC2SoftIntClear = (unsigned long*)((char *)VIC2base + 0x1C);

    rt_disable_irq(IRQ_SOFT); //disable interrupt

    *pBdr ^= 0x20; //switch the Red LED

    if( *pBdr & 0x20 ) //when the switch operation is finished, check
        lightflag = 1; //the bit value for Red LED and change the flag
    else //value
        lightflag = 0;

    *VIC2SoftIntClear |= 0x80000000; //clear bits in the VIC2SoftInt Register

    rt_enable_irq(IRQ_SOFT);
}
```

//The main function does most of the initialization work

```
int init_module(void)
{
    unsigned long *pBbase, *pBddr, *pBdr; //Port B registers
    unsigned long *VIC2base, *VIC2IntEnable, *VIC2SoftIntClear; //software interrupt
    registers

    pBbase = (unsigned long *)__ioremmap(0x80840000,4096,0); //mapping the address
    pBddr = (unsigned long*)((char *)pBbase + 0x14);
    pBdr = (unsigned long*)((char *)pBbase + 0x4);

    VIC2base = (unsigned long *)__ioremmap(0x800C0000,4096,0);
    VIC2IntEnable = (unsigned long*)((char *)VIC2base + 0x10);
    VIC2SoftIntClear = (unsigned long*)((char *)VIC2base + 0x1C);

    *pBddr |= 0xE0; //set all three lights as output
    *pBddr &= 0xFFFFFFF0; //set the button as input
    *pBdr &= 0xFFFFFFF1; //turn off all the lights at the beginning

    //initialize real time task
```

```

rt_set_periodic_mode();           //set to periodic mode
start_rt_timer(nano2count(1000000));

//initialize real time task
rt_task_init(&mytask1,light_switch_button,0,256,1,0,0);
rt_task_init(&mytask2,light_switch_sound,0,256,1,0,0);
rt_task_init(&mytask3,light_switch_touch,0,256,1,0,0);

//start all real time tasks
rt_task_resume(&mytask1);
rt_task_resume(&mytask2);
rt_task_resume(&mytask3);

//attach the software handler to the interrupt line
rt_request_irq(IRQ_SOFT, soft_interrupt_handler, 0, 1);

*VIC2IntEnable   |= 0x80000000; //enable the interrupt request lines
*VIC2SoftIntClear |= 0x80000000; //Clear bits in the VIC2SoftInt Register

rt_enable_irq(IRQ_SOFT); //enable the software interrupt

return 0;
}

void cleanup_module(void)
{
    unsigned long *pBbase, *pBddr, *pBdr;

    pBbase = (unsigned long *)__ioremap(0x80840000,4096,0);
    pBddr = (unsigned long*)((char *)pBbase + 0x14);
    pBdr = (unsigned long*)((char *)pBbase + 0x4);

    *pBddr |= 0xE0;           //set all three lights as output
    *pBddr &= 0xFFFFFFF0;    //set the button as input
    *pBdr &= 0xFFFFFFF0;     //turn off all the lights at the beginning

    rt_task_delete(&mytask1);           //delete real time task
    rt_task_delete(&mytask2);
    rt_task_delete(&mytask3);
    stop_rt_timer();                     //stop timer
    rt_release_irq(IRQ_SOFT);           //release interrupt line
    rtf_destroy(0);                     //destroy the fifos
    rtf_destroy(1);
}

```

```

/*
=====
=====
Name      : FP_user.c
Author    : Yixiang
Version   :
Copyright :
Description : Final Project
=====
=====
*/

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <string.h>
#include <rtai.h>
#include <rtai_lxrt.h>
#include <rtai_fifos.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netdb.h>
#include <fcntl.h>
#include <semaphore.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <errno.h>

#define MSG_SIZE 40

//variables that are shared by main program and pthread

int Lightflag; //the status of red LED

int Warningflag; //the status for touch sensor, used for a different pthread
                //which created by a specific command

void error(const char *msg) //for printing the error message
{

```

```

        perror(msg);
        exit(0);
    }

void *light_status(void *ptr)    //pthread used for receiving the red LED status from fifo 0
                                //and then it will update the flag value into the global variable
                                //"Lightflag"
{
    int fd_fifo_1;
    int lightflag;

    fd_fifo_1 = open ("/dev/rtf/1", O_RDWR);    //to access fifo 1

    while(1)
    {
        read(fd_fifo_1,&lightflag,sizeof(lightflag));//receive the flag data from kernel module

        Lightflag = lightflag;    //update the lightflag data
    }
    close(fd_fifo_1);

    return NULL;
}

void *touch_sensor_status(void *ptr)    //This pthread is created whenever the program
                                        //receives the "PROTECT" command from network.
                                        //The pthread will destroy itself when the function is
                                        //finished.
{
    int fd_fifo_0;
    int warningflag;    //store the value received from kernel module

    fd_fifo_0 = open ("/dev/rtf/0", O_RDWR);    //to access fifo 0

    while( Warningflag != 1 )    //If the sensor has not been touched, the pthread will keep
                                //reading the data from fifo until it receives the trigger value
                                //which means the touch is detected
    {
        read(fd_fifo_0,&warningflag,sizeof(warningflag));//receive the flag from kernel
module

        Warningflag = warningflag;    //update the value to global variable warning_flag
    }
    close(fd_fifo_0);

    return NULL;
}

```

```

int main(int argc, char *argv[])
{
    int a;    //not using for pthread
    pthread_t thrd1;
    pthread_t thrd2;

    //UDP connection
    int length;
    int boolval = 1;           //for a socket option
    struct sockaddr_in server;
    char buf[MSG_SIZE];        //for store the receiving message
    int sock, n;               //sock used to send the broadcast message
    struct sockaddr_in from;
    socklen_t fromlen;

    char msg1[MSG_SIZE] = "CHECK";
    char msg2[MSG_SIZE] = "SWITCH";
    char msg3[MSG_SIZE] = "PROTECT";

    unsigned long *VIC2base, *VIC2SoftInt; //variable to access the software interrupt
    int fd_map;           //variable for "mmap" function

    //create the network communication
    if (argc < 2)
    {
        fprintf(stderr, "ERROR, no port provided\n");
        exit(0);
    }

    sock = socket(AF_INET, SOCK_DGRAM, 0); // Creates socket. Connectionless.
    if (sock < 0)
        error("Opening socket");
    length = sizeof(server);           // length of structure
    bzero(&server, length);           // sets all values to
    zero      server.sin_family = AF_INET;           // symbol constant for
Internet domain
    server.sin_addr.s_addr = INADDR_ANY;           // IP address of the machine on which
                                                    // the server is running
    server.sin_port = htons(atoi(argv[1]));       // port number

    // binds the socket to the address of the host and the port number
    if (bind(sock, (struct sockaddr *)&server, length) < 0)
        error("binding");

    // set broadcast option
    if (setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &boolval, sizeof(boolval)) < 0)

```

```

{
    printf("error setting socket options\n");
    exit(-1);
}

fromlen = sizeof(struct sockaddr_in); // size of structure

//after the network connection is created, initialize the pthread which keeps updating the
//red LED status
pthread_create(&thrd1, NULL, light_status, &a);

//map the software interrupt register address
fd_map = open("/dev/mem", O_RDWR | O_SYNC );
if(fd_map < 0)
    printf("mmap open error!\n");

VIC2base = (unsigned long *)mmap(NULL,
    getpagesize(),
    PROT_READ|PROT_WRITE,
    MAP_SHARED,
    fd_map,
    0x800C0000);
VIC2SoftInt = (unsigned long*)((char *)VIC2base + 0x18);

//server operation
while (1)
{
    // bzero: to "clean up" the buffer. The messages aren't always the same length.
    bzero(buf, MSG_SIZE);

    // receive the message from client
    n = recvfrom(sock, buf, MSG_SIZE, 0, (struct sockaddr *)&from, &fromlen);
    if (n < 0)
        error("recvfrom");
    printf("\nReceived a datagram: "); //print the received message
    printf("%s\n", buf);

    if( strcmp(buf, "CHECK") == 0 ) //if message received is "CHECK"
    {
        if(Lightflag == 1) //compare the value from Lightflag variable which should be
            //updated by "light_status" pthread. "1" means the light is
            //on. "0" means the light is off.
            n = sendto(sock, "The light is On!\n", 16, 0, (struct sockaddr *)&from, fromlen);
            //send the light status message back to client
    }
}

```

```

else
    n = sendto(sock,"The light is Off!\n",17,0,(struct sockaddr *)&from,fromlen);
}
else if ( strcmp(buf,msg2,6) ==0 ) //if message received is "SWITCH"
{
    *VIC2SoftInt |= 0x80000000; //triggered the software interrupt which is used for
                                //switching the red LED
    n = sendto(sock,"Light is Switched!\n",18,0,(struct sockaddr
*)&from,fromlen);
                                //update to the client that the operation is complete
}
else if( strcmp(buf,msg3,7) ==0) //if message received is "PROTECT"
{
    Warningflag = 0; //initial the warningflag value
    pthread_create(&thrd2,NULL,touch_sensor_status,&a);
    //create the pthread which updates the warningflag value

    while( Warningflag != 1) //wait for the warning flag triggered in terms of
                            //the trigger of touch sensor
    {

    }
    n = sendto(sock,"WARNING!!!\n",10,0,(struct sockaddr *)&from,fromlen);
                                //Whenever the trigger happens, send
                                //the warning message back to the client.
}
else
{
    n = sendto(sock,"Error Input!\n",12,0,(struct sockaddr *)&from,fromlen);
}
if (n < 0)
    error("sendto");
}

return 0;}

```