|  |
| --- |
| University of missouri columbia electrical & computer engineering |
| A GLANCE AT INTERNAL FANS |
| 4220 Embedded Systems Final Project |
|  |
| **Justin Alderton** |
| **5/14/2015** |

|  |
| --- |
|  |

# Abstract:

This study examined the design and implementation of an internal fan controller. Being such a broad topic, to be specific, this project will focus mainly on the implementation of a thermally triggered real time event. Also to help the viewer visualize, the simulation will involve LED lights for representation of a system cool, and system hot state.

# Introduction:

The motivation behind this project comes from personal interest and fascination behind a real time system being able to cool itself based on temperature readings. My main goal of this study was to design a system that demonstrated the system design of a real time event. The short-term goal here was to first, design a system that would successfully implement reading in temperatures and then output a signal based on those temperature readings. Long term/final goal was to demonstrate how the proposition of shorter bursts of fan usage could possibly increase the efficiency of the overall system. Again, the purpose of an internal fan controller is to turn on due to a certain temperature being reached, enabling the system to be cooled back down to the desired temperature. This project was to be a real time event driven project in that there were certain real time constraints to be met based on a real time event. I believe that my project is the perfect case showing a real time event, and also “Hard Real Time” constraints being met.

# Background:

Internal fan controllers are used in many modern embedded systems. A thermostat that would constantly be keeping track of a room temperature would trigger once a temperature was met to either heat, or cool the temperature back to the desired temp. Everyday personal computers have an internal fan in them that again, once a certain system temperature is reached will turn on the fan in order to cool down the internal workings of the microcontroller. Also a similar device is that of which almost everyone has grown to be using these days which is the cellphone. Cellphone design relates to my project due to the similarity of D/A (digital to analog) and A/D(analog to digital) logic. I used a summing amplifier circuit to implement the D/A conversion. Here, I needed to take a digital 4 bit input from my program, and through the summing amplifier circuit would be converted to a voltage to power the fan. Figure 1 shows the circuit that is implemented on my breadboard. A digital-to-analog converter takes a set of binary numbers and creates a single analog value as an output. Based on the pre-determined resistor values, and given a certain set of digital bits, the op-amp summing circuit could output an analog signal.

# Proposed Method/System description/ Implementation:

Figure 1 shows the circuit that is implemented on my breadboard. A digital-to-analog converter takes a set of binary numbers and creates a single analog value. Based on the pre-determined resistor values, by a certain set of bits, the op-amp summing circuit could output an analog signal.

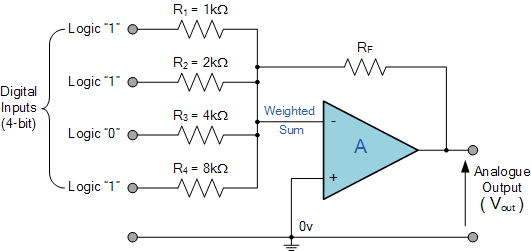
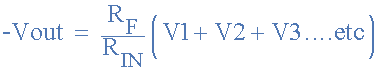


Figure (DAC op-amp summing circuit)



Equation : DAC Summing Amplifier Equation

Given the equation in Equation 1, an output voltage could be calculated based on the resistors chosen. For my DAC circuit, R1 was set to be a 22 kOhm resistor, R2 was 10kOhm, R3 was 3.3 kOhm, and R4 was a 2.2 kOhm resistor. The feedback resistor was set at 4.7 kOhm. After a set of digital inputs was received a certain voltage could be calculated. For my case a digital input of “1” was to represent 5V, and a digital input of “0” was 0V. The fan I bought for my project was a piece of junk to say the least. I was very disappointed in not being able to test my DAC circuit due to after receiving my fan (two weeks later than expected), the power wire was missing. It came with a ground wire attached but nothing else. This was my own mistake thinking I could get away with buying a cheap fan off of Amazon.

Also included in my system design was the capability of taking in and reading real time temperatures. This was done by using the TMP 36 temperature sensor I purchased, along with the MAX197 optional A/D register on the TS 7250 board in the lab.

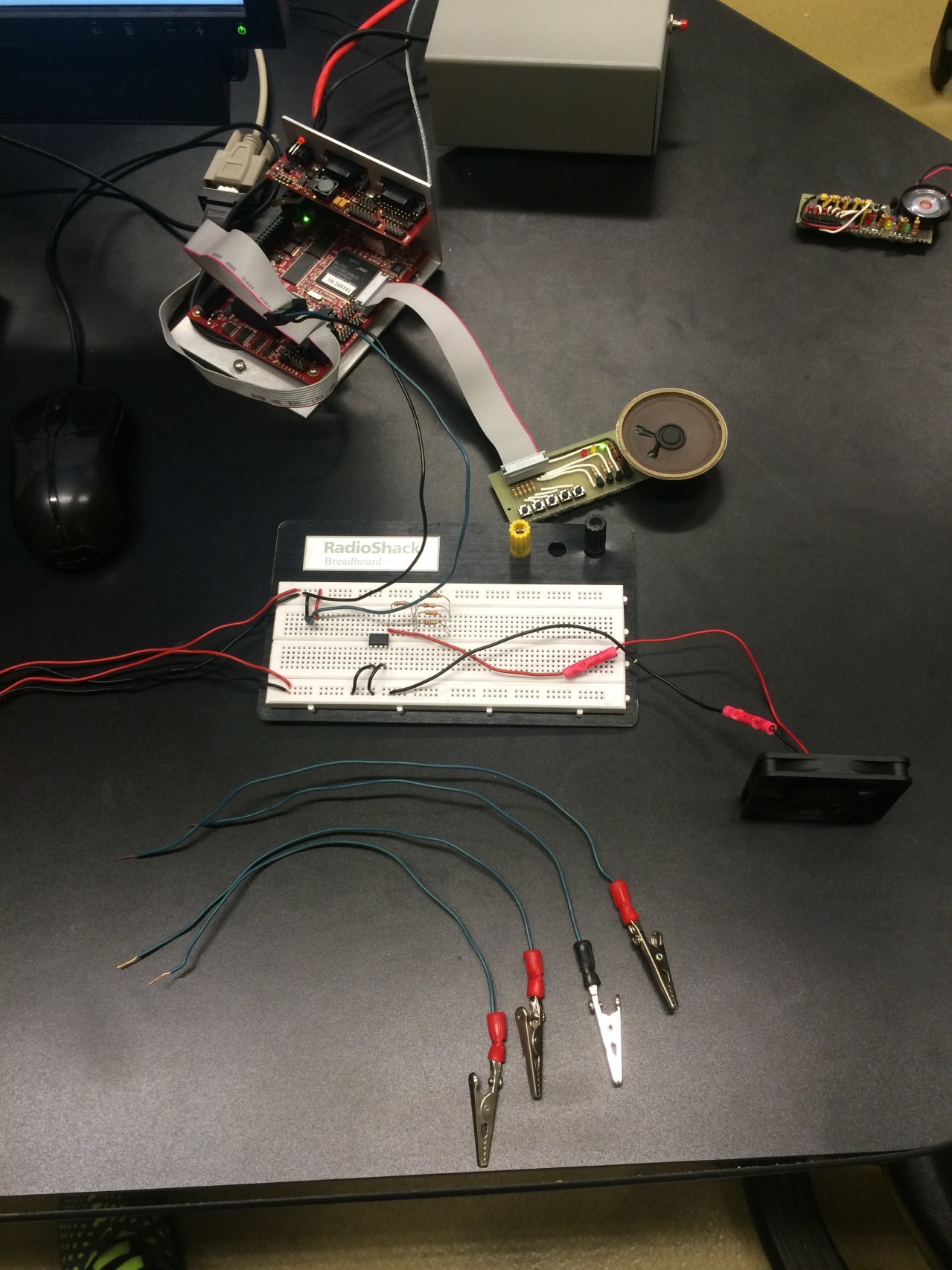


Figure Circuit Design

As you can see in Figure 2, the temperature sensor (see Figure 4) in the top left corner of the circuit was supplied a voltage of 3.3 V from my voltage regulator (see Figure 3), and then was grounded through the black ground wire connected to the breadboard. The four wires with alligator clips connected to them at the bottom of the Figure were the wires that I planned to connect to each resistor and then the DIO pins on register port B. These were to enable a connection to be made in order to implement the DAC circuit.

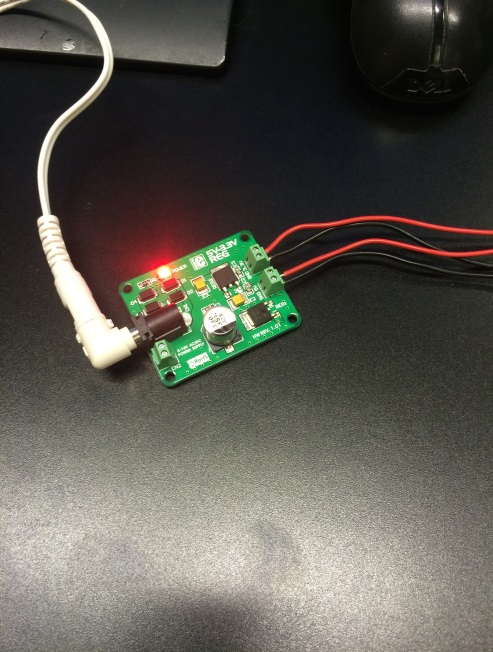
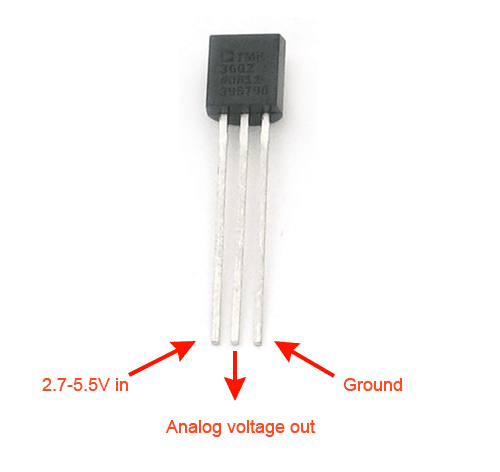
 

Figure : Voltage Regulator Figure 4: TMP 36 Temperature Sensor

Figure 3 shows the MikoElektronika 5V-3.3VReg Board. This board was the voltage regulator used in my system design. It allowed me to plug into a 12V dc source and regulate a voltage of 3.3V and 5V dc to be supplied to the temperature sensor and the 5V fan. This part of my design was ideal in that I had no problems with my voltage calculations and designated dc voltage. After receiving my fan in the mail, I soldered a wire on the back of the fan to try and make it work. However, this was not successful. Instead, as you will see in the project presentation video, I turned a green led light on to represent my system was cool, and then based on a soft interrupt being triggered, I flashed the red led light to indicate that the “hot” real time temperature had been reached. The red light flashing is to represent when the fan would be turned on, until the system had cooled down in which the solid green led light comes back on to represent system was again cool. This will all be better understood after watching my project presentation video. Figure 4 shows the temperature sensor that was picked at my discretion. This was a simple yet accurate temperature sensor that worked well with my system design. As you can see from Figure 4, a voltage supply of 3.3V was supplied to the left pin, the right pin was grounded, and the middle pin is what was connected to the MAX 197 A/D register to receive my real time temperature. This temperature sensor supplies a voltage output that is linearly proportional to the Celsius temperature. It provides accuracy of +- 1 degree Celsius @ 25 degrees Celsius, and +- 2 degrees Celsius over the -40 to +125 degrees Celsius range.

# 

Figure 5: Hardware System Design

Now I will discuss the software side of my system design. My software consisted of two parts: Kernel Space and User Space programs. I will start off with the user space program. My user space program started off by mapping to the software interrupt register of 0x800C0000 that would be triggered when a certain temperature was reached. Next I created a named pipe called “my\_pipe” which was my way of establishing communication between getting the temperature and sending it to the fileWrite thread to be written to an output file. I also mapped to the MAX 197 register. I created 4 different threads that would be running concurrently. The first thread that was created was a thread that would setup the overall system test. Here is where I asked such questions as period of system test, number of seconds to test for, certain temperature the user could be notified of and finally it would start a system test. The second and third threads that were created were to one, write to an output file as the temperatures were read in (if user selected), and then also a thread that was entitled UI. This thread allowed the user to interact with the currently running task. This thread allowed the user to Start/Stop temperature readings, Pause temperature readings, Save a temperature reading, and finally the user could just watch the temperatures collected as my program would print the information to the terminal if this was desired. The reception from MAX 197 A/D registers was collected in a function called getTemp. This function was set up to basically get the temperature readings and pass them accordingly throughout my program. Since my created named pipe was constantly being accessed, the use of semaphores was necessary in that each thread running concurrently might cause race conditions of writing to the named pipe. I used sem\_wait() and sem\_post functions to avoid this. The last thread that was created was called P\_SEND. This thread was spawned out of the getTemp function and was used to constantly send the real time temperatures through a real time fifo over to my kernel space program where a soft interrupt was set and handled appropriately through my soft handler. Each time a temperature was read in in the user space program, it was sent through the real time fifo and checked in the kernel space. Once the temperature reached approximately 75 degrees Fahrenheit, an interrupt occurred in which “My\_Soft\_Handler”, would allow the red light to start flashing. Eventually once the temperature had been cooled back down to below 75 degrees, the green light would turn back on to indicate that the system had returned to a safe temperature. This sensor is a wide range, low voltage temperature sensor that outputted an analog voltage, which was then converted to a Celsius temperature.

# 

Figure 6: Software Flowchart

# Experiments and Results:

Testing was done thoroughly on my project. After finally getting my circuit built, and anA/D conversion setup, I started testing my temperature sensor. This was what I configured first was to read in temperature. Again the temperature sensor I purchased was the TMP 36 and was a fairly easily sensor to setup and get accurate temperature readings from.

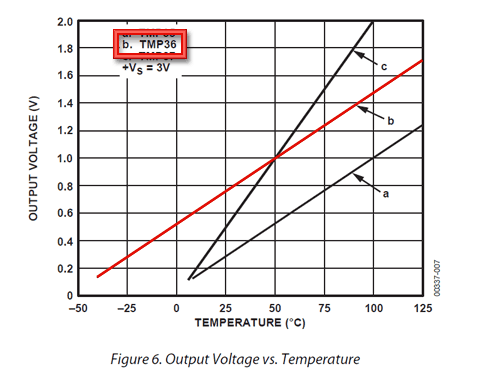


Figure 7: Output Voltage vs. Temperature

Figure 6 is a graph of how the TMP36 sensor was observed to correlate with the temperature readings. After the temperature sensing was configured to be working properly, I moved on to the implementation of my DAC circuit and outputting a voltage to the 5V fan. While waiting for my fan to arrive, I had already configured my DAC circuit on my breadboard and was ready to test the fan once I received it. However, after receiving the fan in the mail two weeks later than expected, it was a piece of junk that someone had torn up. The terminal on the backside of the fan was completely gone so there was no way for me to run a power wire to it (although I went ahead and soldered one on there to see if possibly it would work). It didn’t, and this was very frustrating. I opted to implement and test turning on a green light to indicate “system cool”, and then if the interrupt were triggered, would flash the red light indicating “system hot” state. Although this was very frustrating and quite depressing, the implementation of the lights would achieve everything in my system design that was to represent a real time event driven project.

# Discussion and Conclusions:

To conclude this project, I want to say that I had a lot of fun. Seriously. It was challenging in that tying hardware and software together was a learning curve, but I believe my implementation was spot on and accurate. Concepts such as user space and kernel space programs, named pipes, real time fifos, semaphores, interrupts, ISR, interrupt handler, real time tasks, and P-threads were all used throughout my project. Also, I was able to incorporate electrical devices as my hardware components, which made it a very well rounded project and implementation. Also in conclusion, I was again, very disappointed that I was unable to test a fan enable. However, after finishing my project I am proud and confident in my system design of an internal fan controller. Overall this gives the overview of how this real time system operates. That was the overall goal of this project: “this project will focus mainly on the implementation of a thermally triggered real time event.” The “thermally” indicates of course that taking temperatures was involved. Taking temperatures was successfully implemented and is considered a real time task. Some possible alternatives to the design of my system would have been of course a different fan. Also, programming on an Arduino, which could have possibly simplified things in such a way that I could have gotten away with not having to build a DAC circuit as such. I have no complaints about the A/D conversion used on the MAX 197 as it was fairly easy to understand and implement. Overall this project design and simulation was what I was expecting. The design itself went smooth. I started by writing most of my software while waiting for the hardware components, and finished with the design and testing of the hardware and software components combined. This was a small voltage application, the larger devices would require larger voltage supplied and modifications would have to be made. Therefore my system design would have to be kept in the scope of small voltage gain devices. To improve upon my system design, I could again, try using an Arduino for the software, and also could try and purchase a pre manufactured A/D and D/A converter. This would obviously simplify the circuitry while the software would stay the same. Altogether though, I am happy and definitely satisfied with how this system design turned out!

# Works Cited:

Technologic Systems (2014, May). TS-7250. Retrieved from

<http://wiki.embeddedarm.com/wiki/TS-7250>

Laboratory Electronics II (2009, Sep 28). Digital to Analog. Retrieved from

<http://www.niu.edu/~mfortner/labelec/lect/p575_05b.pdf>

Analog Devices (2015, May). TMP 36. Retrieved from

<http://www.analog.com/en/products/analog-to-digital-converters/integrated-special-purpose-converters/integrated-temperature-sensors/tmp36.html#product-overview>

MikroElektronika (2015, May). 5V-3.3V Reg. Retrieved from

<http://www.mikroe.com/downloads/get/1257/5v3_3v_reg_manual_v1000.pdf>

# Appendices:

**KERNEL MODULE:**

**/\***

\* kernel.c

\*

\* Created on: Apr 21, 2015

\* Author: jlavz6

\*/

**#ifndef** MODULE

**#define** MODULE

**#endif**

**#ifndef** \_\_KERNEL\_\_

**#define** \_\_KERNEL\_\_

**#endif**

**#include** <linux/module.h>

**#include** <linux/kernel.h>

**#include** <asm/io.h>

**#include** <unistd.h>

**#include** <rtai\_fifos.h>

**#include** <rtai\_sched.h>

**#include** <rtai\_sem.h>

MODULE\_LICENSE("GPL");

RT\_TASK task1,task2;

RTIME period;

**unsigned** **long** \*map\_address;

**unsigned** **long** \*pbdr;

**unsigned** **long** \*pbddr;

**unsigned** **long** \*ptr1;

**unsigned** **long** \*softInt;

**unsigned** **long** \*softClear;

**unsigned** **long** \*softEn;

**int** check;

**int** get = 0;

**int** number;

SEM sem1;

**static** **void** **red\_light**(**int** x)

{

**while**(1)

{

**rt\_sem\_wait**(&sem1);//acquire semaphore

**if**(check == 1)

{

\*pbdr &= 0xFFFFFF7F;//Turn off the green led

\*pbdr |= 0x00000020;//Turn on the red led representing system has reached 75 degrees fahrenheit

**rt\_sleep**(period);

check = 0;

}

**rt\_sem\_signal**(&sem1);//release semaphore

}

}

**static** **void** **green\_light**(**int** z)

{

**while**(1)

{

**rt\_sem\_wait**(&sem1);//acquire semaphore

\*pbdr &= 0xFFFFFFDF;//Turn off the red led

**rt\_sleep**(period);

\*pbdr |= 0x000000080;//Turn on the green led to represent system is below 75 degrees fahrenheit

**rt\_sem\_signal**(&sem1);//release semaphore

}

}

**static** **void** **my\_soft\_handler**(**unsigned** irq\_num, **void** \*cookie)//handler for interrupt (63)

{

ptr1 = (**unsigned** **long** \*)**\_\_ioremap**(0x800C0000, 4096, 0);//Map to physical address for soft Interrupt register

//Soft Interrupt Data

softInt = ptr1 + 6;//VIC2SoftInt

softEn = ptr1 + 4;//VIC2IntEnable

softClear = ptr1 + 7;//VIC2SoftIntClear

**rt\_disable\_irq**(63);//disable interrupt

get = **rtf\_get**(0,(**void** \*)&number,**sizeof**(**int**));//Receive temperature in Fahrenheit through real time fifo

**if**(get < 0)

{

**printk**("Error in receiving from the fifo.\n");

}

**if**(number >= 75)//If system is greater that 75 degrees fahrenheit, internal fan on

{

**rt\_task\_resume**(&task2);//Red light

check = 1;//Flag fan

**printk**("Fan will turn on!\n");

}

**else**

{

check = 0;

**printk**("System cool!\n");

}

\*softClear |= 0x80000000;//clear interrupt 63

**rt\_enable\_irq**(63); //enable interrupt

}

**int** **init\_module**(**void**)

{

**rt\_sem\_init**(&sem1,1);//initialize semaphore

map\_address = (**unsigned** **long** \*)**\_\_ioremap**(0x80840000,4096,0);//Map address for port b register

ptr1 = (**unsigned** **long** \*)**\_\_ioremap**(0x800C0000, 4096, 0);//Map to physical address for soft Interrupt register

pbdr = map\_address + 1;//Offset to port B data register

pbddr = map\_address + 5;//Offset to port B data direction register

\*pbddr |= 0x000000E0;//Sets the three led lights to be output

\*pbdr &= 0xFFFFFF1F;//Turn all lights off to begin

**rtf\_create**(0,**sizeof**(**int**));//Create realtime fifo

softInt = ptr1 + 6;//VIC2SoftInt

softEn = ptr1 + 4;//VIC2IntEnable

softClear = ptr1 + 7;//VIC2SoftIntClear

//Set to periodic mode

**rt\_set\_periodic\_mode**();

//Assign period

period = **start\_rt\_timer**(**nano2count**(1000000000));

**rt\_request\_irq**(63, my\_soft\_handler, 0, 1); //attach handler to interrupt line

\*softClear |= 0x80000000;//Clearing Soft Interrupt

\*softEn |= 0x80000000;//Enabling Soft Interrupt

//Initialize real time task

**rt\_task\_init**(&task1,green\_light,0,256,0,0,0);

//Resume real time task

**rt\_task\_resume**(&task1);

//Initialize real time task

**rt\_task\_init**(&task2,red\_light,0,256,0,0,0);

//Resume real time task

**rt\_enable\_irq**(63); //enable soft interrupt

**return** 0;

}

**void** **cleanup\_module**(**void**)

{

**rt\_task\_delete**(&task1);//Delete real time task

**rt\_task\_delete**(&task2);//Delete real time task

**stop\_rt\_timer**();//Stop real time timer

**rtf\_destroy**(0);//Delete Real Time Fifo

**rt\_sem\_delete**(&sem1);//Delete semaphore

**rt\_release\_irq**(63);//release soft interrupt

**printk**("MODULE REMOVED\n");

}

**USER SPACE PROGRAM:**

**/\***

\* user.c

\*

\* Created on: Apr 21, 2015

\* Author: jlavz6

\*/

**#include** <time.h>

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <sys/types.h>

**#include** <sys/mman.h>

**#include** <fcntl.h>

**#include** <unistd.h>

**#include** <termios.h>

**#include** <pthread.h>

**#include** <semaphore.h>

**#include** <rtai\_fifos.h>

**#include** <string.h>

//stores temperature readings

**typedef** **struct** {

**double** temp\_read;

**struct** timeval temp\_time;

}check\_Temp;

//variables declared inside of structure to enable shared information between threads

**typedef** **struct**{

**int** choose\_File;//Varaible choosen by the user regarding which output file they would like to log

**int** choose\_Print;//flag to set printing the screen

**int** set\_Stop;//flag to stop the collection of temps

**int** choose\_Periodic;//Variable to choose the period

**int** set\_Pause;//Flag to pause the collection of temperatures

**int** set\_Save;//Flag to indicate a temperature was saved

**int** fan\_set;//Flag to indicate system has reached temperature

**int** Reboot;//Flag to reset the system

**struct** timeval start\_Time;//System start time

**int** set\_Time;//Flag to indicate that the start time for the system has been set

**int** ask\_file;//Variable chosen by user on which output file is desired

**double** ask\_notify;//Variable

**int** tempNotifyUnit;

}shared\_Info;

**void** **fileWrite**(**void** \*ptr);//Will write to an output file

**void** **getTemp**();//Collects the temperatures being taken

**void** **UI**(**void** \*ptr);//Allows the user to interact with the real time program

**int** **kbhit**(**void**);//Used to allow keyboard usage

**void** **Collect**(**void** \*ptr);//Sets up the software side of the program

RTIME Period1;

sem\_t sem1;

sem\_t sem2;

**double** time\_MS;

**unsigned** **long** \*ptrSoft = NULL;//Pointer to map to for soft Interrupt

**unsigned** **long** \*softInt = NULL;//Pointer for soft Interrupt register

**void** **P\_SEND**(**void** \*ptr)//Thread that will send the temperatures collected through a real time fifo

{

**double** Celsius;

**int** Fahrenheit;

\*softInt |= 0x80000000;//Enables soft Interrupt

check\_Temp \*Temp = (check\_Temp \*)ptr;

**int** k\_pipe = 0;

**if**((k\_pipe = **open**("/dev/rtf/0", O\_RDWR))<0)//Opening real time fifo

{

**printf**("Opening fifo write error.\n");

}

//Conversion from AD signal to Celsius temp, and then from Celsius temp to Fahrenheit temp

Celsius = Temp->temp\_read;

Celsius = Celsius \* 5000 / 4096;

Celsius = (Celsius - 500) / 10;

Fahrenheit = ((Celsius\*1.8) + 32);

**if**(**write**(k\_pipe,(**void** \*)&Fahrenheit,**sizeof**(**int**))<0)//Write to real time fifo

{

**printf**("pipe write error\n");

**exit**(-1);

}

**pthread\_exit**(0);

}

**int** **main**(**void**) {

pthread\_t read\_Temp;//Thread spawned to read in temperatures

shared\_Info info;//Structure defined type to be passed to other threads in order to share stored information

**volatile** **unsigned** **long** \*enable;//will check if MAX197 installed

**unsigned** **long** \*ptr;

**int** fd = **open**("/dev/mem", O\_RDWR|O\_SYNC); //file descriptor for mapping

//Mapping to soft interrupt register

ptrSoft = (**unsigned** **long** \*)**mmap**(NULL,**getpagesize**(),PROT\_READ|PROT\_WRITE,MAP\_SHARED,fd,0x800C0000);//Mapping to physical memory

softInt = ptrSoft + 6;//offsetting to soft Interrupt register

**double** PeriodTemp = 0;//used for setting the real time period

**int** ready;//check to see if user is ready for system start

**system**("mkfifo my\_pipe >& /dev/null"); //create named pipe

**system**("clear");

**sem\_init**(&sem1, 0, 2);

//initialize semaphores

**sem\_init**(&sem2, 0, 1);

//initialize variables of structure

info.set\_Stop = 0;

info.choose\_Print = 0;

info.choose\_File = 0;

info.choose\_Periodic = 0;

info.set\_Pause = 0;

info.set\_Time = 0;

info.ask\_notify = 0;

info.fan\_set = 0;

info.set\_Save = 0;

info.Reboot = 0;

//map address of ptr register

ptr = **mmap**(0, **getpagesize**(), PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0x22400000);

enable = (**unsigned** **long** \*)ptr;//check bit 0 to see if MAX197 ADC is installed

**if**(\*enable & 0x00000001)

{

**printf**("MAX197 Connection\n");

}

**else**

{

**printf**("MAX197 Not Connected\n");

**pthread\_exit**(0);

}

//check if ready to start collecting data

**printf**("\nAre you ready to collect temperatures (y/n)? ");

ready = **getchar**();

**if**(ready == 'y')

{

**printf**("Enter period length in milliseconds? "); //get period value from user

**scanf**("%d", &info.choose\_Periodic);

PeriodTemp = info.choose\_Periodic;

PeriodTemp = PeriodTemp \* 1000000; //convert to nanoseconds

Period1 = start\_rt\_timer(nano2count(PeriodTemp)); //set base period for real time task

//create thread for data collection

**pthread\_create**(&read\_Temp, NULL, (**void**\*)&Collect, (**void** \*)&info);

**pthread\_join**(read\_Temp, NULL);

}

**else**

{

**printf**("\n");

}

**usleep**(5000);

**return** 0;

}

**void** **Collect**(**void** \*ptr)

{

shared\_Info \*info = (shared\_Info \*)ptr;

**int** iterations;//Variable indicating how long user would like to collect temperatures

**printf**("\nEnter how many seconds you would like to collect temperature: ");

**scanf**("%d", &iterations);

//calculation of measurements to be taken

iterations = ((iterations \* 1000) / info->choose\_Periodic) + 1;

//get a temp to indicate system has reached a certain temperature

**printf**("\nWould you like to be notified at a specific temp?\n"

"Specify temperature in degrees Fahrenheit (0 = No): ");

**scanf**("%lf", &info->ask\_notify);

info->ask\_file = **getchar**();

//ask if user would like to save data to files

**printf**("\nWould you like to output data collected to an output file?(y/n)\n");

info->ask\_file = **getchar**();

//ask user what kind of files they would like and set choice

**if**(info->ask\_file == 'y')

{

**printf**("Choose type of file:\n");

**printf**("1 - (.csv) file\n");

**printf**("2 - (.txt) file\n");

**printf**("3 - cancel\n");

**scanf**("%d", &info->choose\_File);

}

//create thread for writing to output files and allow user to interact

pthread\_t write\_it;

**pthread\_create**(&write\_it, NULL, (**void** \*)&fileWrite, (**void** \*)info);

**system**("clear");

pthread\_t user\_It;

**pthread\_create**(&user\_It, NULL, (**void** \*)&UI, (**void** \*)info);

//initialize real time task and make periodic with specified period

RT\_TASK\* rtwrite = rt\_task\_init(nam2num("read\_temp"), 0, 512, 256);

rt\_task\_make\_periodic(rtwrite, rt\_get\_time() + nano2count(1000000000), Period1);

**gettimeofday**(&info->start\_Time, NULL);

//collecting data

**int** i=0;

**for**(i=0; i < iterations; i++)

{

**if**(info->set\_Stop == 1)

{

**pthread\_exit**(0); //if stop has been selected, quit data collection

}

**if**(info->set\_Pause != 1)

{

getTemp(); //get temperature measurement

}

**else**

{

**if**(info->set\_Pause == 1)

{

**printf**("\nPaused...\n"); //shows pause if terminal output enabled and paused

}

}

**if**(info->Reboot == 1)

{

i = 0; //if Reboot as has be selected, Reboot data collection

}

rt\_task\_wait\_period();

}

//System has finished testing

**printf**("\nSystem test complete!\n");

**usleep**(5000);

**pthread\_exit**(0);

}

//gets a temperature reading from channel 0 of the MAX 197 ADC on the TS-7250

**void** **getTemp**()

{

**volatile** **unsigned** **char** \*op, \*control, \*busy; //registers for ADC

check\_Temp \*Temp; //structure for temperature measurement

**int** fd = **open**("/dev/mem", O\_RDWR|O\_SYNC); //file descriptor for mapping

**int** my\_pipe;

**unsigned** **short** \*result;

**unsigned** **char** \*busy\_Val;

**unsigned** **char** \*optionValue;

Temp = (check\_Temp\*)**malloc**(**sizeof**(check\_Temp)); //allocate memory

//map addresses of registers

op = **mmap**(0, **getpagesize**(), PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0x22400000);

control = **mmap**(0, **getpagesize**(), PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0x10F00000);

busy = **mmap**(0, **getpagesize**(), PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0x10800000);

//open named pipe for writing

**if**((my\_pipe = **open**("my\_pipe", O\_WRONLY)) < 0)

{

**printf**("error creating pipe\n");

**pthread\_exit**(0);

}

//get time of temperature measurement

**gettimeofday**(&Temp->temp\_time, NULL);

op = (**unsigned** **char** \*)op;

//get sample from ADC Channel 0, Unipolar, 5V Range

\*control = 0x40;

busy\_Val = (**unsigned** **char** \*)busy;

//wait until conversion complete

**while**(\*busy\_Val & 0x80)

{

busy\_Val = (**unsigned** **char** \*)busy;//bit 7 is 1 when complete

**printf**("");

}

result = (**unsigned** **short** \*)control;

Temp->temp\_read = (**double**)\*result;//store result in structure as a double

pthread\_t send\_It;//Create pthread to send temps to kernel space

**pthread\_create**(&send\_It, NULL, (**void** \*)&P\_SEND, (**void** \*)Temp);

//write to pipe to get temperature structure to fileWrite thread

**sem\_wait**(&sem1);

**if**(**write**(my\_pipe, Temp, **sizeof**(\*Temp)) != **sizeof**(\*Temp))

{

**printf**("error writing to pipe\n");

**pthread\_exit**(0);

}

**sem\_post**(&sem2);

}

//writes to .csv or .txt (if selected) and displays to terminal (if selected)

**void** **fileWrite**(**void** \*ptr)

{

shared\_Info \*info = (shared\_Info \*)ptr;

check\_Temp \*Temp;

Temp = (check\_Temp\*)**malloc**(**sizeof**(check\_Temp));

**int** my\_pipe;

**double** Celsius, Fahrenheit;

FILE \*fd1, \*fd2;

**double** time\_MS;

**if**(info->choose\_File == 1)//if .csv option selected, create .csv file

{

fd1 = **fopen**("dat.csv", "w+");

**fprintf**(fd1, "Time, Celsius, Fahrenheit, Save");

}

**if**(info->choose\_File == 2)//if .txt option selected, create .txt file

{

fd2 = **fopen**("dat.txt", "w+");

**fprintf**(fd2, "Temperature Data\n");

}

//open named pipe for reading

**if**((my\_pipe = **open**("my\_pipe", O\_RDONLY)) < 0)

{

**printf**("error creating pipe\n");

**pthread\_exit**(0);

}

**while**(1)

{

**sem\_wait**(&sem2);

**if**(info->Reboot == 1)//if Reboot selected, Reboot the system

{

**if**(info->choose\_File == 1)

{

**fclose**(fd1);

fd1 = **fopen**("dat.csv", "w+");

**fprintf**(fd1, "Time, Save");

}

**if**(info->choose\_File == 2)

{

**fclose**(fd2);

fd2 = **fopen**("dat.txt", "w+");

**fprintf**(fd2, "Temperature Data\n");

}

info->set\_Time = 0; //reset start time

info->Reboot = 0; //deselect reset

}

**if**((**read**(my\_pipe, Temp, **sizeof**(\*Temp))) < 0)//read from named pipe

{

**printf**("error reading from pipe\n");

**pthread\_exit**(0);

}

**if**(info->set\_Time == 0)//check set\_time, (if 0) set start time

{

info->start\_Time = Temp->temp\_time;

info->set\_Time = 1; //set set\_Time to show start time has been set

}

//convert time to milliseconds

time\_MS = ((Temp->temp\_time.tv\_sec \* 1000.0) - (info->start\_Time.tv\_sec \* 1000.0));

time\_MS += ((Temp->temp\_time.tv\_usec / 1000.0) - (info->start\_Time.tv\_usec / 1000.0));

//convert ADC result to Celsius and then Celsius to Fahrenheit

Celsius = Temp->temp\_read;

Celsius = Celsius \* 5000 / 4096;

Celsius = (Celsius - 500) / 10;

Fahrenheit = ((Celsius\*1.8) + 32);

**if**(info->ask\_notify != 0)//If user entered a temperature to be notified of

{

**if**(Fahrenheit >= info->ask\_notify)

{

info->fan\_set = 1;

**printf**("Fan will turn on at %d degrees Fahrenheit!\n", (**int**)info->ask\_notify);

}

**else**

{

info->fan\_set = 0;

}

}

**if**(info->choose\_File == 1)//write Temperatures collected to info.csv

{

**if**(info->set\_Save == 1)

{

**fprintf**(fd1, ", Saved...");//if set\_Save selected, save measurement

}

**if**(info->fan\_set == 1)

{

**fprintf**(fd1, ", Fan turned on...");//If temperature reached for cool down

}

**fprintf**(fd1, "\n%.3f, %.1f, %.lf", time\_MS, Celsius, Fahrenheit);

}

**if**(info->choose\_File == 2)//write Temperatures collected to info.txt

{

**if**(info->set\_Save == 1)

{

**fprintf**(fd2, "Saved...\n"); //if set\_Save selected, save measurement

}

**if**(info->fan\_set == 1)

{

**fprintf**(fd2, "\nFan turned on...\n");//If temperature reached for cool down

}

**fprintf**(fd2, "\nTime(microseconds): %.4f\n", time\_MS);

**fprintf**(fd2, "Temp (Celsius): %.1f\n", Celsius);

**fprintf**(fd2, "Temp (Fahrenheit): %.lf\n", Fahrenheit);

}

**if**(info->choose\_Print == 1)//if choose\_Print set, print to the terminal

{

**printf**("\nTime: %.3f\n", time\_MS);

**printf**("Temp (Celsius): %.1f\n", Celsius);

**printf**("Temp (Fahrenheit): %.lf\n", Fahrenheit);

}

//deselect flags

info->set\_Save = 0;

info->fan\_set = 0;

**sem\_post**(&sem1);

}

**usleep**(5000);

**free**(Temp);

**pthread\_exit**(0);

}

**void** **UI**(**void** \*ptr)

{

shared\_Info \*info = (shared\_Info \*)ptr;

**int** user\_Choice;

**printf**("System test started...\n"); //system start

**while**(1)

{ //display options to user during data collection

**printf**("\nEnter selection by pressing corresponding keyboard number.\n");

**printf**("1 - Start/Pause Collection Process\n");

**printf**("2 - View Temperatures being collected\n");

**printf**("3 - Save Previous Temperature\n");

**printf**("4 - Stop Collecting Temperatures\n");

**scanf**("%d", &user\_Choice);

//if option 1 selected, start/pause data collection

**if**(user\_Choice == 1)

{

info->set\_Pause = info->set\_Pause ^ (1 << 0);

**if**(info->set\_Pause == 0)

{

**printf**("\nCollecting Data...\n");

}

**else** **if**(info->set\_Pause == 1)

{

**printf**("\nPaused...\n");

}

}

//if option 2 selected, flag choose\_Print which will enable printing to the terminal

**else** **if**(user\_Choice == 2)

{

info->choose\_Print = 1;

**while**(!kbhit())

{ //wait for keyboard press and then end viewing

**printf**("");

}

user\_Choice = 0;

info->choose\_Print = 0;

}

//if option 3 selected, save temperature reading

**else** **if**(user\_Choice == 3)

{

info->set\_Save = 1;

**printf**("\nRecorded!\n");

}

//if option 4 selected, stop system testing

**else** **if**(user\_Choice == 4)

{

info->set\_Stop = 1;

**printf**("\nSystem stop!\n");

**pthread\_exit**(0);

}

}

}

//function used to monitor for a keyboard press

**int** **kbhit**(**void**)

{

**struct** timeval tv;

fd\_set rdfs;

tv.tv\_sec = 0;

tv.tv\_usec = 0;

FD\_ZERO(&rdfs);

FD\_SET(STDIN\_FILENO, &rdfs);

**select**(STDIN\_FILENO+1, &rdfs, NULL, NULL, &tv);

**return** FD\_ISSET(STDIN\_FILENO, &rdfs);

}