Lab 5

Kaiser Sun, Xinyu Gu, Kanika Aggarwal

June 11, 2019

Abstract

This lab is the adaption of previous lab. New features are added at this time's update, including external control of blood pressure, temperature, respiration rate, and EKG data; inter-system remote control and data exchange; and a remote user interface for doctors. We also added three additional features: the traffic management system, calling doctor service, and games for patients with Alzheimer. All the newly added functions passed through the test and satisfied the requirements.

Contents

1 Introduction

This project is the final phase in the development of a low cost, portable, medical monitoring system. In the this version, we built a simple kernel and utilized a non-preemptive schedule to manage the selection and execution of the set of tasks comprising our system. We took the first steps towards implementing and incorporating the Peripheral Subsystem by replacing several of the modeled / simulated measurement capabilities with the initial design of the drivers to support the specified tasks and developed the initial support for the Intrasystem Communication Channel.

2 Design Specification

2.1 Data Structure

2.1.1 Task Queue

Task Queue is the execution flow of the tasks. Compared to last time, task Queue is implemented as a double linked list.

2.1.2 Task Control Block

Task Control Block(TCB) is the node of the task queue. Each TCB contains a pointer to the task function, a pointer to the task data, a previous TCB node, and a next TCB node.

2.2 Scheduler

This function will schedule all the tasks, including deleting task node, inserting task node, and arrange the tasks.
Each two seconds, the scheduler will implement an interrupt to call the function that is in charge of reading the pressed point on TFT display. When ever it was called, the series of select function will be executed.
Each 5 seconds, the remote communication function will be called and the doctor's UI will be updated.
The system will keep recording the number of times that the patient reach a warning level every 8 hours. Whenever the warning data in each category appears equals to or more than 100 times, the system will automatically call both the doctor and a emergency website.
Whenever the blood pressure or temperature is too high, or the pulse rate is too low, the interruption of communications function will be called to send warning message to the serial window of Mega.

2.3 Measure

At this phase, real-time measurement will be done on the patient based on external voltage signals and computer generated data.

2.3.1 Temperature

Temperature is measured from an analogue signal then scaled to human temperatures. It should be stored in a circular reading buffer that doesn't update unless temperature changed by 15

2.3.2 Blood Pressure

Every time a button is pressed, the blood pressure cuff should "inflate" or "deflate" causing the blood pressure to increase/decrease by 10 percent. Whether the cuff inflates or deflates is determined by switch. Systolic pressure should be in the range 110-150 while diastolic pressure should be in the 50-80 mm Hg range. It should be stored in a circular reading buffer that does not update unless BP changed by 15 percent.

2.3.3 Pulse Rate

Pulse rate value should be collected from an external analog signal which ranges from 0-3.3V. The frequency of the analog signal should correspond to pulse rate. The pulse rate should range from 10-200 BPM.I t should be stored in a circular reading buffer that doesn't update unless PR changed by 15 percent.

2.3.4 Respiration Rate

Respiration rate value should be collected from an external analog signal which ranges from 0-3.3V. The frequency of the analog signal should correspond to pulse rate. The pulse rate should range from 10-50 BPM.I t should be stored in a circular reading buffer that doesn't update unless RR changed by 15 percent

2.3.5 EKG

Collect patients EKG data by accepting an analog sinusoidal input with a peak of 3.2 V and returning the frequency of the input. Collect 256 equally spaced samples with a sampling rate from 2.5-3 times the max frequency and store them into a buffer to process. Perform an FFT on the EKG samples to process the frequency. The EKG frequency range is 35Hz to 3.75KHz.

2.4 Compute

This function will take the data acquired by the Measure task and perform necessary transformations or corrections.
Temperature is corrected to be in Celsius.
Systolic pressure and Diastolic pressure are corrected to be in mm Hg.
Pulse rate is corrected to be in beats per minute.
The computed data will be stored in the first element of the array, and all the other values will be shifted to next index. The oldest, which is also the last, value will be dropped.
The EKG data was processed in Compute function;

2.5 Display

Display function will be called whenever the TFT goes into annunciation mode. It accepts the returned values from warning and compute function and displays the selected values on the TFT display
Blood pressure(systolic and diastolic) should be presented on the top row.
Temperature, pulse rate, and battery status will be presented on the second row of the display.
The default color will be green. The alarm/warning value will be red.
The display panel will appears with an EXIT button. By hitting the EXIT button, users can get back to the main panel.

2.6 Warning-Alarm

This function accepts the returned value from the compute function.
This function keeps track that if temperature, blood pressure, pulse rate are in the normal range.

2.7 Status

This function updates the battery state. The battery state will be decremented by 1 each time the function is called.

2.8 Select

When first enter select function, goes into the main panel with two buttons: Menu and Annunciations. When Menu is pressed, goes into the menu function, while Annunciation is pressed, goes into the annunciation mode.

2.8.1 Menu

In the menu function, it reads user's selection of which value to display. There are blood pressure, pulse rate, and temperature to select. When selected to display, the button will turns green; when undo selected, the button will turns red.

2.8.2 Anno

In the anno function, it will call display function and reads user's selection. The only button on Anno's layout is EXIT button, which leads user back to the main page.

2.9 Remote communication

User can control the display and measurement through local network on a laptop. The message it sends including a start of message, command name, and end of message; The message it receives including the data that system control obtains and holds, warning information of the patient, or the message send actively by patient.

2.10 Game

It is designed for patient with Alzheimer diseases to practice ability of thinking. It can also help other users with recognition impairment to train their thinking method.
It basically provide users games to play. Only when the answer is correct the user will be able to exit the page– this can enforce their motivation.

2.11 Call Doctor

It is like remote communication, but was in emergency case. It should be initiated by the user and the doctor should receive the message immediately. It has the highest priority in the system.

2.12 Traffic Management System

When this button is pressed, a animation of phasor fire will be played to show the clear of traffic.

3 Software Implementation

3.1 Scheduler

In the scheduler, we utilized the properties of linked list nodes of TCBs. There will be a system clock on Mega, on which will not have delay function(we will talk about the interrupt on UNO later). Whenever the value of time interval reaches two seconds, we insert an linked list node of select function, which is to read

the input of user's touch. After execution, we delete it from the linked list.
Whenever a warning occurs, a communication function will be insert into the linked list if it is enabled. Since in the requirement it is not required, we can still enable it by uncommenting the code.
When the warning has occurred 100 times in 8 hours, an emergency task will be inserted into task queue. Then, the doctor(remote control) will receive an emergency announcement and the emergency website will be called.

3.2 Measure

3.2.1 Temperature

An analog signal is collected from an external source (see circuit diagram). The value of the analog signal range from 0-1024 in arduino. The value is divided by 37 and + 20 to scale to a wide variety of human temperatures. Temperatures of ¡10 are rounded to 10 and temperatures ¿50 are rounded to 50.

3.2.2 Blood Pressure

A digital 1 or 0 is collected from the arduino for the switch and button. If the value of the button changes from 0 to 1, the value of the blood pressure change. If the value of the switch if 1, the blood pressure is multiplied by 1.1, if the value of the switch is 0, the blood pressure is multiplied by .9. Systolic pressures less than 50 are rounded to 50. Systolic pressures greater than 70 are rounded to 70. Diastolic pressures less than 29 are rounded to 29. Diastolic pressures greater than 50 are rounded to 50. (These limits are on the raw values of the pressure) .

3.2.3 Pulse Rate

For pulse rate we use an attach interrupt function that measures the frequency of an external source. Every time the analog value read by the pulse rate goes from 0 to 1, the attach interrupt calls an ISR function that increments the frequency count by 1. Every time the measure frequency function is called the function checks the frequency count, waits half a second, checks the frequency again, and subtracts the two values to get the frequency over half a second. This is multiplied by 2. When pulse rate is called it measures the frequency over 1 second and returns this number as the BPM value of pulse rate. Raw values greater than 64 are rounded to 14 and raw values less than 1 are rounded to 1.

3.2.4 Respiration Rate

Respiration rate uses the same measure frequency function used in pulse rate. When respiration rate is called it measures the frequency over 1 second, divides this by 4 (to scale) returns this number as the number of breaths per minute. Raw values greater than 14 are rounded to 64 and values less than 1 are rounded to 1.

3.2.5 EKG

We simulated the EKG Data with a frequency of 100Hz and a sampling frequency of two and a half times the frequency. We then stored the first 256 values along a sine wave of amplitude 3.2, frequency 100, and at a sampling rate of 1/sampling frequency into a buffer of raw values. We also created an imaginary buffer of size 256 with 0 stored in every value.
Below is a sample of circuit diagram.

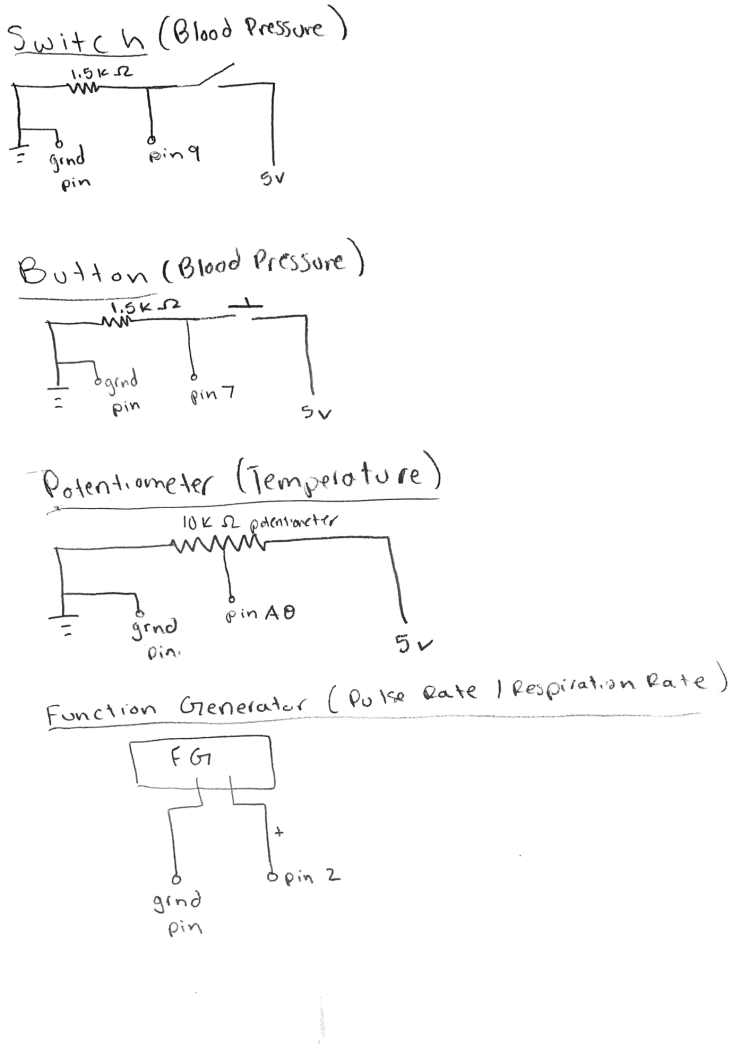

Figure 1: Circuit diagram

3.3 Compute

When the schedule in the mega executes the compute element in the task queue the compute method corrects the raw values of each of the patient's vitals based on the values in the spec. The method takes the raw value of temperature stored in the TCB, multiplies by .75, adds 5, and updates the value of the corrected value of temperature. This repeats with the systolic (multiply by 2 and add 9) diastolic (multiply by 1.5 and add 6) and pulse rate (multiple by 3 and add 8).

3.4 Display

When the schedule in the mega executes the display element in the task queue, the function displays the real time values of the patients values. The TFT display is set to a color and text is printed to display the values of the patients values. If the values are out of range the text is set to red if they're in range the text is set to green. Warnings and alarms are printed at the bottom of the screen.
Normal range for the measurements:
- Temperature: 36.1-37.8
- Systolic pressure: 120-130 mmHg

- Diastolic pressure: 70 to 80 mmHg
- Pulse rate: 60 to 100 beats/minute
- Battery: greater than 20%

3.5 Warning-Alarm

When the schedule in the mega executes the warning/alarm element in the task queue, the function identifies the patient's vitals that are out of range. If the raw value of the temperature systolic pressure , diastolic pressure, pulse rate, are lower than the spec defined lower limit or greater than the spec defined upper limit, booleans for the respective values are set to 1. (ie if temperature raw value is greater than 37.8 or lower than 36.1 the boolean tempOutOfRange is set to 1). It also identifies is the temperature is too high or the pulse rate is too low. I the temperature is greater than the spec defined upper limit, a boolean to identify high temperatures is set to 1. If the pulse rate is lower than the spec defined lower limit, a boolean defining low pulse is set to low.

3.6 Status

When the schedule in the mega executes the status element in the task queue, the function takes the current value of the battery, and decrements the value inside the pointer by 1 if the battery status is greater than 0 (this prevents negative values of battery).

3.7 Remote Communication Call Doctor

The remote communication was implemented through Putty. We used the serial port to simulate the local network. The command will be accepted by the terminal of Putty and will be sent to Mega.

3.8 Game

The game was a simple game for patients with Alzheimer disease. This was designed to test the respond ability of patients, and to keep patients' thinking ability being active.

3.9 Interrupt

In UNO code, we implemented an interrupt service routine by using the function attachInterrupt. We count the appearance of 1 from function generator at the interrupt interval of 60 seconds.

3.10 Traffic Management System

We simply utilized the unused button left on the main page of Mega. Whenever the user pressed the button, a phsaor fire will be initiated.

3.11 Control flow and data flow

3.11.1 Control flow

The main control is the ATMega. Mega will execute the tasks in the linked list task queue. There is a system time on the mega: whenever the time interval between current time and last time reaches two seconds, an node of select function will be inserted. After execution, the node will be deleted.
When Mega is running the main program, UNO is also running the program for measuring the pulse rate using the interrupt. Whenever the UNO receive a command from Mega, it write the values in serial.
While Mega and Uno both running its own work, the data and warning result they produced will be sent to the remote control on the laptop.

3.11.2 Data flow

The data mainly stored on Mega and are temporarily processed on UNO. The measured values on UNO are stored in integers or doubles, and them are wrote to mega to stored in the arrays.
The result of data processing will be sent to remote control, which is the laptop in this case. The remote control will also return the doctor's command back to the Mega for further operation.
Below is a brief demonstration of control flow and data flow:

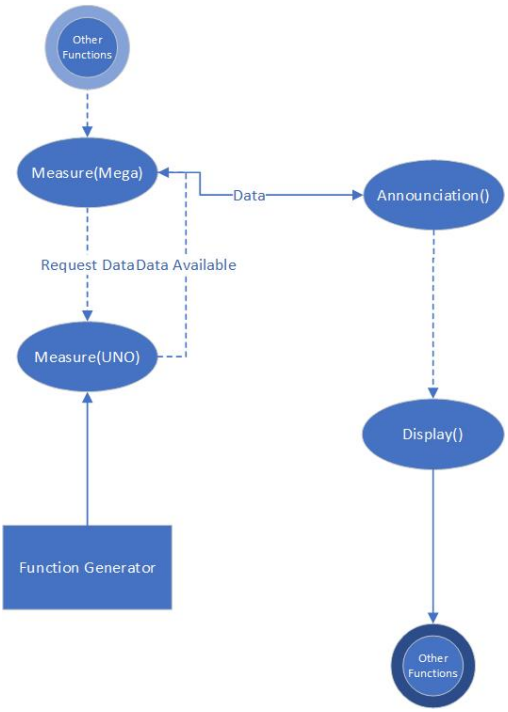

Figure 2: Control Flow

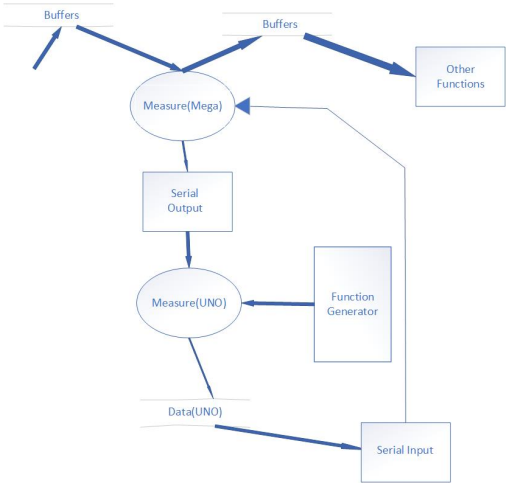

Figure 3: Data Flow

4 Test Plan

In last lab, the basic function of our prototype was already tested. Therefore, in this test plan, we will focus on the testing of newly added functions, which is the touchscreen, interrupt, buffer, and the linked list task queue.

5 Test Specification

In order to make sure the device is properly working we must check the following:

- The touchscreen is checking for pressure every two seconds and each button is working well with a clear text of function.
- The values of the temperature blood pressure, and pulse rate disappear as they are selected not to display.
- The color of buttons on menu page will change from red to green or from green to red when an item is selected or unselected.
- The buffer need to work as well as single values at version 1 did. It should properly record the changing values and be delivered correctly.

- We must make sure that the old values are stored in the buffer correctly.
- The remote communication is working correctly.
- Sending commands and receiving data accurately and display data on the terminal correctly.
- The interrupt should work correctly. The measurement of pulse rate should corresponding to the setting of function generator.

6 Test Cases

We observed the TFT and observed the values of temperature, and blood pressure were changing in the expected intervals as defined in the spec. We noted that each of the values remained within their upper and lower limits. We altered the hertz on the function generator and noted that the pulse rate was changing relative to the frequency. (Double the frequency should correlate to about double the pulse rate). We noted that the battery value started at 200, and decremented by 1 everytime the screen refreshed until the battery value reached 0. One by one we de-selected each value in the menu and noted that those values were no longer being displayed by the TFT.

We also observed the remote terminal to see 1. if the values are displayed 2. if the values are the correct values to be displayed 3. if the commands are correctly sent to Mega and the operations are correctly handled.

6.1 Buffer

Buffer is tested both individually and on the device, because the whole program is too large to debug on.
We create an individual program to run the shiftchar method of buffer. With initial value set of 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', new value of 'i' to see whether the values are shifted correctly, with 'i' at the index of 0.
We also tested buffer on the device. We test whether it stores values correctly.
Test values:

name	initial value
temperature	42.5
Systolic pressure	169
Diastolic pressure	129
pulse rate	158
battery	200

Expected values of buffer(take after one loop as example):

buffer name	value(after 1 loop)
temperature	44.5 ,42.5, 0, 0, 0, 0, 0, 0
pressure	172, 127,169, 129, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
pulse rate	60 ,67, 0, 0, 0, 0, 0, 0, 0

buffer name	value(after 2 loop)
temperature	43.5 ,44.5 ,42.5, 0, 0, 0, 0, 0
pressure	171, 126, 172, 127, 169, 129, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
pulse rate	53, 60 ,67, 0, 0, 0, 0, 0, 0, 0

6.2 Task Queue

To test the task queue, we tested the linked list file and the general task queue separately. We create a file to run the test of linked list { 1, 2, 3, 4, 5 }, { 7, 6, 4, 3, 2, 1, 10, 99}, and tried to insert value into specific index and delete them when the loop is finished. We inserted the values of 10, 3, 293 at index 2, 3, 4, 5; When we have a index out of range, the linked list insert will automatically add it at the end.

In the device, we use Serial.print at each task function in mega to make sure that the task queue is executed correctly. The predicted output order should be: select $->$ measure $->$ compute $->$ status; Within the CD of 2 seconds of select functions, it should be measure $->$ compute $->$ status.

6.3 Interrupt

To test whether the interrupt is working correctly, we both output the count of heart beat during a period of time and we measure the time outside the device.
We also try different input value of function generator to see the change of value of pulse rate.
The value we tried varied from .1 Hz to 6 Hz. We expected the heart beat should have great change over the change of values.

6.4 EKG Data

To test whether the EKG data is working correctly we check that the EKG value is the same as the frequency we input.

6.5 Blood Pressure

To test whether the blood pressure is working correctly we make sure the initial value is 109 and 149. Every time we click to increment/decrements the value should only change every other click and it should have changed by at least 15 percent. It should also not go below 109/149 or above 150/70.

6.6 Pulse Rate

To check is respiration rate is working correctly the value of resp rate should be the frequency of the pulse rate multiplied by 3 and add 8. It should not go above 200 or below 10.

6.7 Respiration Rate

To check is respiration rate is working correctly the value of resp rate should be the frequency of the function generator multiplied by 3 divided by 4 and add 7. It should not go aove 50 or below 10.

6.8 Temperature

To check if temperature is working correctly the value should be between 20-40. The value should change as the potentiometer changes, and should only change if the new temperature value is atleast 15 percent different than the previous value.

6.9 Game

Game does not have any value input. We directly tested it by hand.
First test to input the wrong choice, then see what the screen reflects. The screen was supposed to keep still if user's input is wrong.
Then test to input the right choice, then see what the screen reflects. The screen was suppose to exit game mode and back to menu.

6.10 Remote control

First test static data. Set a set of data on Mega of temperature: 24; Blood pressure 9, 18; Respiration rate: 45; pulse rate: 25; EKG: 100; Battery: 200. Observe if the terminal display as we designed.
Then test the command. First input "sbe", which means control the display of blood pressure. Observe if there is an output on the terminal, then observe if there is a display/undisplay of blood pressure. Then input "ste", "sre", "ske"... 's' means the start of message, 'e' mens end of message. Then try input invalid

commands, which means message without start of message and end of message: 't', 'bte', 'se', 'sr'... Observe that if the remote system respond with "invalid statement".

7 Result Analysis

The device is working properly. All cases were tested and passed. Below are the photos of working devices. The menu screen allows you to de-select whether it displays temperature, blood pressure or by making it red. Annunciations displays the selected (green on menu) values. As you can see when blood pressure is deselcted it does not appear on the Annunciation screen.

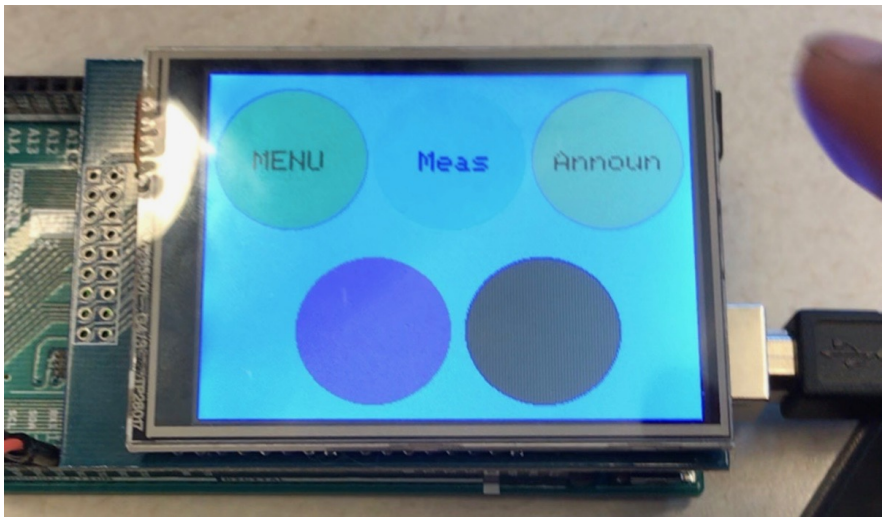

Figure 4: Main Page

Figure 5: Annunciation Page(Blood pressure unselected)

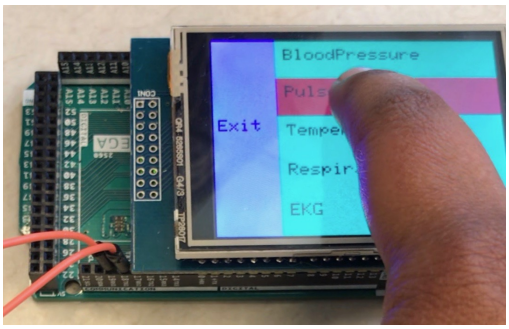

Figure 6: Menu Page

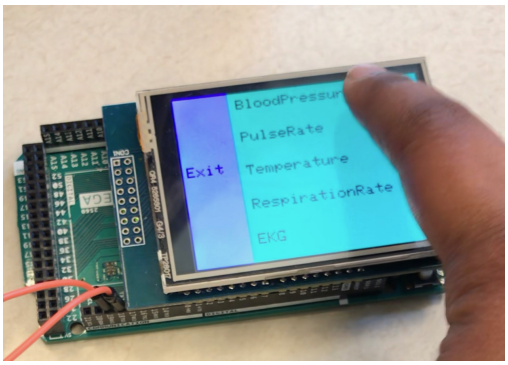

Figure 7: Unselected Blood Pressure

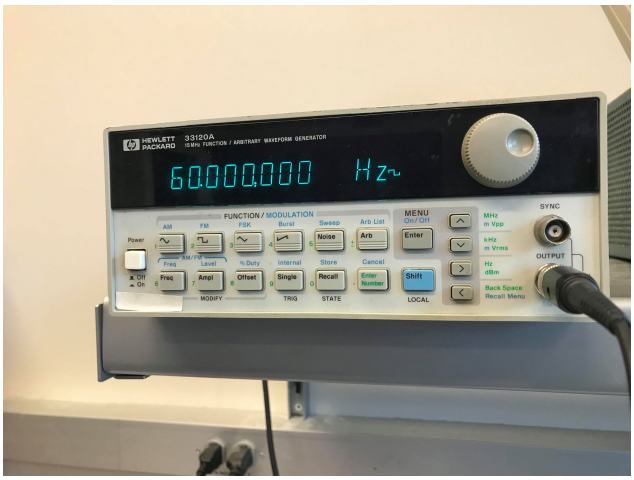

Figure 8: Function Generator

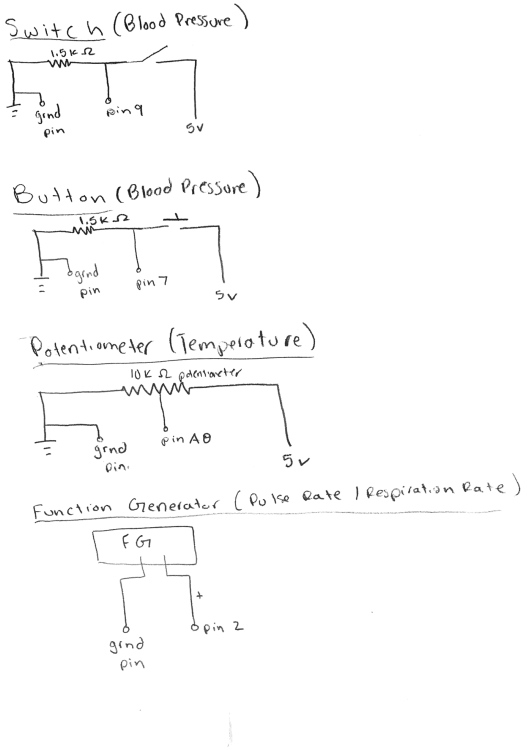

Figure 9: Circuit Diagrams

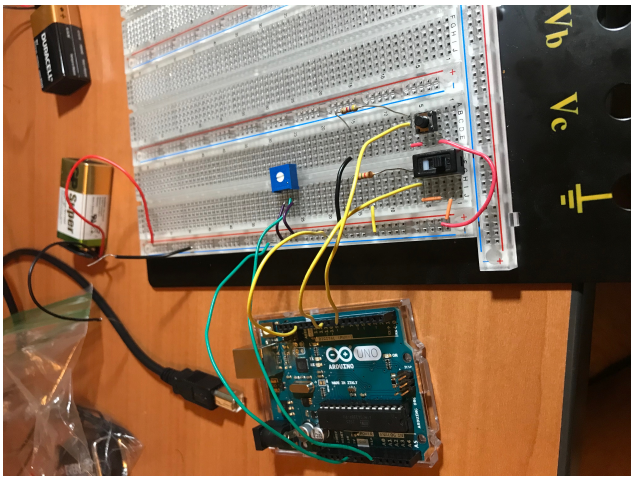

Figure 10: Button and Swith (for Blood Pressure) and Potentiometer (For Temperature)

If a stealth submersible sinks it can be found using echolocation. This can be done through the arduino attaching 2 ultrasonic sensors and using heron's formula to complete the triangle and locate the object. Alternatively you can train beluga whales that have built in echolocation skills to locate the object. A helium filled balloon is going to move away perpendicularly to the surface from the surface of the earth. If a balloon is south of the equator, moving away from the surface of the earth perpendicularly will cause it to "fall south of the equator" . This is demonstrated in the graphic below:

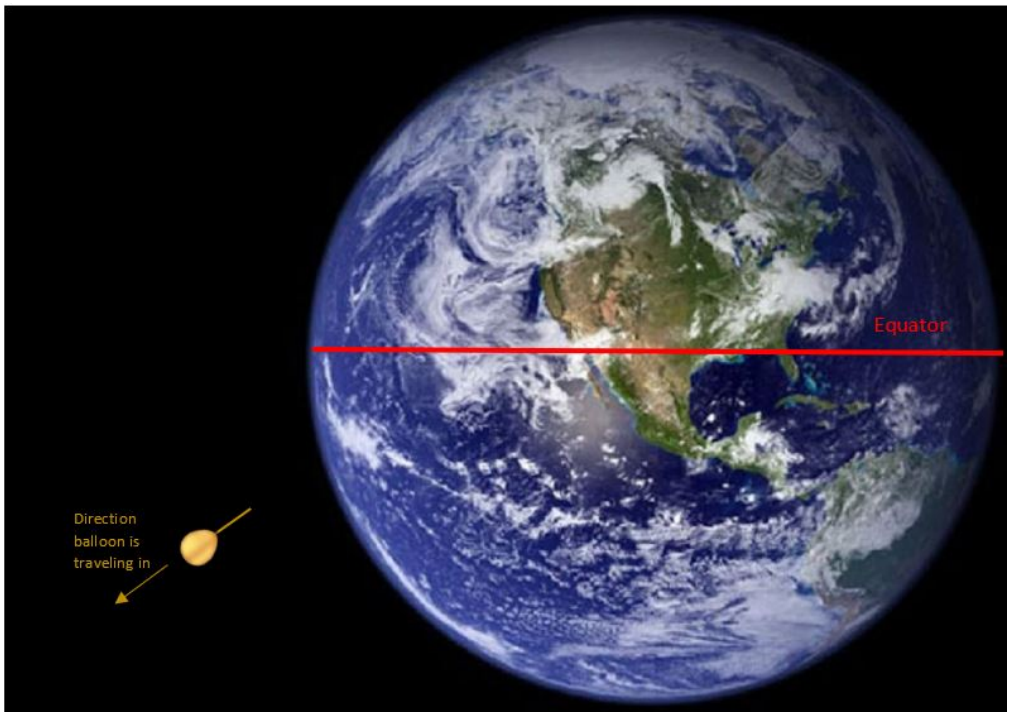

Figure 11: Demonstration

7.1 Analysis of Resolved Errors

At first did not have the buttons working using the button API provided. We figured out that and then wrote a button API by ourselves. However, later on we found that we only need to add a pin refresh to correctly implementation of API.
Then we find that our values were almost fixed when displaying on the TFT. Later we found that it was because we have the screen to refresh every 2 seconds to make it more user-readable, but during this time the change of values are mostly completed. Therefore, we choose to make values update as the refresh of the screen.

7.2 Analysis of Unresolved Errors

We had an unresolved error in the function generator portion of the lab. Although we were successfully able to make the pulse rate read values from the transducer and the pulse rate (although a bit erratic for the first few seconds) managed to stabilize to a single, steady, pulse rate value at higher frequency values. However since our time interval over which we measured pulse rate was short, at low frequencies our pulse rate reading became a little erratic. There was a bit of a trade off here because if we increased the time interval, it would take longer for the pulse rate to stabilize to a value and the pulse rate readings would not be real time (ie waiting a full minute before displaying pulse rate readings).

8 Summary

This project further familiarized us with many important embedded systems concepts. We created a linked list node TCB that held values and pointers to data used throughout the lab. First we created high level diagrams, activity diagrams, data and control diagrams, and class diagrams that detailed the intricacies of this project, and a high level organization structure of the functions and classes we need to create. We then created the buffer to store the values of pressure, pulse rate, and temperature. We also created an interrupt to measure the pulse rate, blood pressure, and respiration rate, and EKG data. We realized the control

through a remote protocol and deliver the data finely. Finally we debugged and tested. In the code we wrote, we implemented array pointers, interrupts, linked list, serial communication, TFT touch screen, function genderator, and interrupt. Finally, we put together a cohesive lab report detailing how we executed the project, the results we received, and a reflection on our experience. Through this lab we created a functional e-doctor device that can obtain values from function generator, and user can select the values to display.

9 Conclusion

Throughout this project we learned to use TFT touch screen, serial communications, function generator, interrupts, doubly linked list, and debugging arduino code. We also strengthen our ability for a complex engineering project, and the ability of design as long as making diagrams. We also enhance our ability of troubleshooting and test design. This lab was a fantastic application of the significant concepts we learned in the lecture. It applied the knowledge of UML diagrams, function generator with interrupt, and serial communication. It also forced us to apply fundamental principles of engineering and problem solving skills that we learn through the course of the lab.

Through out this quarte we learned a lot of embedded system development. We learned about C language, pointers, interrupts, and real time systems. We gained the ability of design and debugging. (**?**)

10 Appendices + Code

10.1 Task Execution Time

Measure: $\sim$ 30ms
Compute: $\sim$ 2ms
Warning/Alarm: $\sim$ 4ms
Status: $\sim$ 7 ms
Display: $\sim$ 40ms

10.2 Pseudo Code

10.2.1 Mega Main

Mega Code Main

// include core graphics library, Hardware-specific library and tcb header

// initialize unsigned int arrays for raw values of the temperature, systolic pressure, diastolic pressure, pulse rate,

// initialize unsigned int arrays for corrected values of the temperature, systolic pressure, diastolic pressure, pulse rate,
//initialize pointers to point to their respective arrays //initialize short for initial battery state (set to 200)

//initialize chars for blood pressure, temperature, and pulse out of range (set to 0)

//initialize booleans for high blood pressure, high temperature, and low pulse (set to false)

//initialize booleans for display

// initialize taskQueue and TCBs

- Initialize an array of 5 elements that stores TCBs;
- Initialize TCB's: meas, comp, disp, alar, and stat;
- Initalize Data Struct MeasureData with name meaD;
- Initalize Data Struct ComputeData with name cD;
- Initalize Data Struct DisplayData with name dDa;
- Initalize Data Struct WarningAlarmData with name wAD;
- Initalize Data Struct StatusData with name sD;

```
void setup()
{
  // Initialize the serial connections to 9600 bits per second
 // Initialize TFT Display
// Setup the data structs
```

- meaD data struct: pass in the raw buffer pointers for temperature, systolic pressure, diastolic pressure, and pulse rate
- cD data struct: pass in the raw buffer pointers for temperature, systolic pressure, diastolic pressure, and pulse rate
- dDa data struct: pass in the corrected buffer pointers for temperature, systolic pressure, diastolic pressure, pulse rate, and the pointer for battery state
- wAD data struct: pass in the raw buffer pointers for temperature, systolic pressure, diastolic pressure, pulse rate, and the pointer for battery state
- sD data struct: pass in the buffer pointer for battery state

```
 // Setup the TCBs
```

- meas TCB: pass in the address of the Measure function and the address of meaD data Struct
- comp TCB: pass in the address of the Compute function and the address of cD data Struct
- disp TCB: pass in the address of the Display function and the address of dDa data Struct
- alar TCB: pass in the address of the WarningAlarm function and the address of wAD data Struct
- stat TCB: pass in the address of the Status function and the address of sD data Struct

```
 // Setup task queue
```

- Store the meas TCB in the 0th element of taskQueue linked list
- Store the comp TCB in the 1st element of taskQueue linked list
- Store the stat TCB in the 2nd element of taskQueue linked list
- Store the alar TCB in the 3rd element of taskQueue linked list
- Store the disp TCB in the 4th element of taskQueue linked list

```
}

void loop()
{
  //Call the scheduler Test function and pass the Task Queue into the parameter
}
```

10.2.2 UNO Main

initialize pins
    set switch pin
    set function generator pin
    set button pin
    set temp pin
initialize raw value
initialize global variables (frequency, time delay, etc)
set up
    set transmission rate to 4800
    open pin mode, input button pin
    open pin mode, input switch pin
    open pin mode, input attach interrupt and the pin connected to function generator
    call arduino attachinterrupt function with input pin, an isr of incrementPulse() function, and for it to change every time it goes from high to low (RISING)
loop
    Call Measure(UNO)
    Call Communications (UNO)

10.2.3 communications (UNO Main)

check if byte is available
if the byte is s, start reading values
while the byte is not e, proceed
if the incoming byte is t, read from serial
if the next byte is p, write the temperature
if the incoming byte is b, read from serial
if the next byte is u, write the blood pressure
if the incoming byte is p, read from serial
if the next byte is l, write the pulse rate
if the incoming byte is r, read from serial
if the next byte is l, write the respiration rate

10.2.4 measureFreq (UNO Main)

wait half a second
subtract the total freq count from the previously stored count
update the previous count variable to the new frequency count
return the result of the subtration

10.2.5 incrementFreq (UNO Main)

add 1 to total frequency count

10.2.6 getRawPulseRate (UNO Main)

call measureFreq and save as PR
set PR to 64 if ¿ 66
set PR to 1 if ¡ 1
return PR

10.2.7 getRespirationRate (UNO Main)

call measureFreq and save as RR
set RR to 14 if ¿ 14
set RR to 1 if ¡ 1
return PR

10.2.8 getTemp(UNO Main)

read the analog temperature value from the analog pin
divide that value by 37 and add 20 to scale to human temp values
return that scaled value

10.2.9 getBP(UNO Main)

read the pin for the switch and button to get the new button val and increment/decrement value
if the previous button value was 0 and the new button value is 1
    update the previous button value to the new value
    if the switch value is 1 (increment position) multiply systolic and diastolic pressure by 1.1
    if the switch value is 0 (decrement position) multiply systolic and diastolic pressure by 0.9
if the previous button value was 1 and the new button value is 0
    update the previous button value to the new value if the systolic pressure is less than 50 set it to 50
if the systolic pressure is greater than 70 set it to 70
if the diastolic pressure is less than 29 set it to 29
if the diastolic pressure is greater than 50 set it to 50
if sys = 1 (requestion systolic Pressure variable) return the sysP value else return the DiaP value

10.2.10 Measure (UNO Main)

call getTemp and store under global temperature variable
call getBP(1) and store under global systolic pressure variable
call getBP(0) and store under global diastolic pressure variable
call getPulseRate and store under global pulse rate variable
call getRespRate and store under global resp rate variable

10.2.11 Schedule (Mega Functions)

// create TCB pointer taskQueue of front
    // check if taskQueue is null
    // if taskQueue is not null, call run(taskQueue) and update taskQueue to next
    // else do anything

10.2.12 Measure (Mega Functions)

creates a pointer to a MeasureData struct
sends code to retrieve value of new temperature from uno

reads the new temperature value and stores it in a variable
sends code to retrieve value of new pulse rate from uno
reads the new pulse rate and stores it in a variable
sends code to retrieve value of new pulse rate from uno
reads the new systolic pressure value and stores it in a variable
sends code to retrieve value of new diastolic pressure from uno
reads the new diastolic pressure value and stores it in a variable
push each of the new values into their respective buffers using the shift function

10.2.13 Compute (Mega Functions)

creates pointer to computeData struct
retrieves and updates the following values from the struct
    TC = Temperature * 0.75 + 5
    sysC = systolic pressure * 2 + 9
    diasC = diastolic pressure * 1.5 + 6
    prC = pulse rate * 3 + 8
push each of the new values into their respective buffers using the shift function

10.2.14 Warning (Mega Functions)

if (TC < 36.1 or TC > 37.8)
    temp out of range warning
if (sysC > 120 or diasC < 80)
    bp out of range warning
if (pulse rate > 100 or pulse rate < 60)
    pulse rate out of range warning
if (battery state ¡ 20)
    battery inefficient warning

10.2.15 Status (Mega Functions)

subtract Battery State by 1 unless it's at 0.

10.2.16 Display (Mega Function)

if the boolean to display temperature is true
    if (TC out of range)
        display TC in red
    else
        display TC in green
if the boolean to display BP is true
    if (sysC out of range)
        display sysC in red
    else
        display sysC in green
    if (diasC out of range)
        display diasC in red
    else
        display diasC in green
if the boolean to display pulse rate is true
    if (PR out of range)
        display PR in red

    else
        display PR in green
if (battery inefficient)
    display battery state in red
else
    display battery state in green
display warnings as needed

10.2.17 Shift (Mega Function)

accepts the new value to be stored in the buffer, the buffer size, and a pointer to the buffer
for (each element except the first (starting at the end))
    the value inside the last element = the value inside the value before it
the first element = the new value

10.2.18 Menu (Mega Function)

create a pointer to the Keypad data struct
print enter menu mode
print measure and Annunciation
print an Exit button
set tft requirements
    set cursor, draw subsm fill rectangulars, set text size, etc
open pin mode for XM and YM output
check if pressure is within a certain range
print next menu
if the xy window for temperature is true (temp button is selected)
    set the display temperature boolean to true
if the xy window for blood pressure is true (blood pressure is selected)
    set the blood pressure boolean to true
if the xy window for pulse rate is true (pulse rate button is selected)
    set the display pulse rate boolean to true
if the xy window for quit is true (exit button is selected)
    set the selection pointer to false

10.2.19 anno (Mega Function)

// create pointer to the date pointer
    // setup tft, set cursor, draw subsm fill rectangulars, set text size, etc
    // update point parameters
    // update alarm acknowledge pointer
    To be implemented

10.2.20 Select (Mega Function)

// create a pointer to the keypad data struct
    // check measurement selection point and alarm acknowledge pointer
    // setup tft, set cursor, draw subsm fill rectangulars, set text size, etc
    // update measurement selection pointer ans alarm acknowledge pointer
    // call on menu() id measurement selection pointer is 1
    // call on anno() if alarm acknowledge pointer is 1

10.2.21 doubly linked list insert

// if head of the list is null, point both head and tail pointers to the newly inserted TCB
// else, add to the back of the list

10.2.22 doubly linked list delete

// if head is null, no deletion
// if head equals tail and equals the TCB given, set head and back to null
// if head is not null and head does not equal to back, create current pointer to keep track of current TCB while iterating through the list to find the target TCB
// update the previous and next pointers if find given TCB

10.3 Code Source

10.3.1 Mega Main

```
// #include <Elegoo_GFX.h> // Core graphics library
// #include <Elegoo_TFTLCD.h> // Hardware-specific library
#include "tcb.h"
#include "task.h"
// #include "helpers.h"



bool startF = FALSE;

unsigned int BPindex = 0;
unsigned int PRindex = 0;
unsigned int Tindex = 0;


// initialization started!
unsigned int temperatureRawBuf[8]= {30, 0, 0, 0, 0, 0, 0, 0};
unsigned int bloodPressRawBuf[16]= {80, 80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
unsigned int pulseRateRawBuf[8] = {70, 0, 0, 0, 0, 0, 0, 0};
unsigned int respirationRateRawBuf[8] = {0, 0, 0, 0, 0, 0, 0, 0};
signed int EKGRawBuf[256];
signed int EKGImgBuf[256] = {0};

unsigned char EKGFreqBuf[16] = {0};

unsigned char tempCorrectedBuf[8]= {28, 0, 0, 0, 0, 0, 0, 0};
unsigned char bloodPressCorrectedBuf[16] = {126, 169, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
unsigned char pulseRateCorrectedBuf[8]= {218, 0, 0, 0, 0, 0, 0, 0};
unsigned char respirationRateCorBuf[8] = {0, 0, 0, 0, 0, 0, 0, 0};

unsigned short Mode = 0;
unsigned short batteryState = 200;
unsigned long time1;
unsigned long time2;
unsigned long time3;
unsigned long timec;
unsigned long timeb;
unsigned long timeRD1;
unsigned long timeRD2;
// Emergency data
unsigned long timeEmer1;
```

```
unsigned long timeEmer2;

unsigned int counterBP = 0;
unsigned int counterT = 0;
unsigned int counterPR = 0;

// initialize raw value pointers
unsigned int* temperatureRawPtrr = temperatureRawBuf;
unsigned int* bloodPressRawPtrr = bloodPressRawBuf;
unsigned int* pulseRateRawPtrr = pulseRateRawBuf;
unsigned int* respirationRateRawPtr = respirationRateRawBuf;
unsigned int* EKGRawBufPtr = EKGRawBuf;
unsigned int* EKGImgBufPtr = EKGImgBuf;
unsigned char* EKGFreqBufPtr = EKGFreqBuf;
//initialize corrected pointers
unsigned short* ModePtrr = &Mode;
unsigned char* tempCorrectedPtrr = tempCorrectedBuf;
unsigned char* bloodPressCorrectedPtrr = bloodPressCorrectedBuf;
unsigned char* pulseRateCorrectedPtrr = pulseRateCorrectedBuf;
unsigned short* batteryStatePtrr = &batteryState;
unsigned char* respirationRateCorPtr = respirationRateCorBuf;

//initialize keypad values and pointers
unsigned short localFunctionSelect = 0;
unsigned short measurementSelection = 0;
unsigned short alarmAcknowledge = 0;
unsigned short command = 0;
unsigned short remoteFunctionSelect = 0;
unsigned short measurementResultSelection = 0;
unsigned short displaySelection = 0;
unsigned short displayTM = 0;
unsigned short displayEmergency = 0;
unsigned short displayGame = 0;

unsigned short* displayTMPtr = &displayTM;
unsigned short* displayEmergencyPtr = &displayEmergency;
unsigned short* displayGamePtr = &displayGame;
unsigned short* displaySelectionPtr = &displaySelection;
unsigned short* localFunctionSelectPtr = &localFunctionSelect;
unsigned short* measurementSelectionPtr = &measurementSelection;
unsigned short* alarmAcknowledgePtr = &alarmAcknowledge;
unsigned short* commandPtr = &command;
unsigned short* remoteFunctionSelectPtr = &remoteFunctionSelect;
unsigned short* measurementResultSelectionPtr = &measurementResultSelection;

// Count the number of emergencies
int tWCounter = 0, bpWCounter = 0, rrWCounter = 0, ekgWCounter = 0, prWCounter = 0;

// initialize taskQueue and TCBs
//TCB taskQueue;
TCB meas, comp, disp, alar, stat, keyp, com, remDisp;
MeasureData meaD;
//EKGData EKGD;
ComputeData cD;
DisplayData dDa;
WarningAlarmData wAD;
StatusData sD;
KeypadData kD;
CommunicationsData comD;
```

```
void setup() {
    // Initialize the serial connection of 9600 bits per second
    Serial.begin(9600);
    Serial1.begin(4800);
    // Print
    Serial.println(F("TFT LCD test"));


  #ifdef USE_Elegoo_SHIELD_PINOUT
  Serial.println(F("Using Elegoo 2.4\" TFT Arduino Shield Pinout"));
  #else
  Serial.println(F("Using Elegoo 2.4\" TFT Breakout Board Pinout"));
  #endif

  Serial.print("TFT size is "); Serial.print(tft.width()); Serial.print("x");
      Serial.println(tft.height());

  tft.reset();

   //Use identifier to make sure tft works
   uint16_t identifier = tft.readID();
   if(identifier == 0x9325) {
    Serial.println(F("Found ILI9325 LCD driver"));
    } else if(identifier == 0x9328) {
    Serial.println(F("Found ILI9328 LCD driver"));
    } else if(identifier == 0x4535) {
    Serial.println(F("Found LGDP4535 LCD driver"));
    }else if(identifier == 0x7575) {
    Serial.println(F("Found HX8347G LCD driver"));
    } else if(identifier == 0x9341) {
    Serial.println(F("Found ILI9341 LCD driver"));
    } else if(identifier == 0x8357) {
    Serial.println(F("Found HX8357D LCD driver"));
    } else if(identifier==0x0101)
    {
      identifier=0x9341;
       Serial.println(F("Found 0x9341 LCD driver"));
  }
  else if(identifier==0x1111)
    {
      identifier=0x9328;
       Serial.println(F("Found 0x9328 LCD driver"));
    }
    else {
    Serial.print(F("Unknown LCD driver chip: "));
    Serial.println(identifier, HEX);
    Serial.println(F("If using the Elegoo 2.8\" TFT Arduino shield, the line:"));
    Serial.println(F(" #define USE_Elegoo_SHIELD_PINOUT"));
    Serial.println(F("should appear in the library header (Elegoo_TFT.h)."));
    Serial.println(F("If using the breakout board, it should NOT be #defined!"));
    Serial.println(F("Also if using the breakout, double-check that all wiring"));
    Serial.println(F("matches the tutorial."));
    identifier=0x9328;

    }
    tft.begin(identifier);
    tft.setRotation(1);
```

```
    startUp();
}


int n = 0;

void loop()
{
    time2 = millis();
    timec = millis();
    timeRD2 = millis();
    timeEmer2 = millis();
    // Check if there is an emergency
    if(timeEmer2 - timeEmer1 > 28800000) {
      if(bpWCounter > 100 || tWCounter > 100 || rrWCounter > 100|| ekgWCounter > 100 || prWCounter
          > 0) {
         emergency();
      }
      bpWCounter = 0;
      tWCounter = 0;
      rrWCounter = 0;
      ekgWCounter = 0;
      timeEmer1 = timeEmer2;
    }
    // Call remote display every 5 seconds
    if(timeRD2 - timeRD1 > 5000) {
      insertLast(&remDisp);
      schedulerTest();
      timeRD1 = timeRD2;
      deleteNode(&remDisp);
    }

    if (timec - timeb > 500) {
      counterBP++;
      counterT++;
      counterPR++;
      timeb = timec;
    }
    if (counterT == 5) {
      counterT = 1;
    }
    if (counterPR == 9) {
      counterPR = 1;
    }
    if (counterBP == 3) {
      counterBP = 1;
    }

    if (time2 - time1 > 1) {
      if(*(displayGamePtr) == 0 && *(displayEmergencyPtr) == 0 && *(displayTMPtr) == 0 &&
          *(measurementSelectionPtr) == 0 && *(alarmAcknowledgePtr) == 0 && *(displaySelectionPtr)
          == 0) {
        TCB temp = {&Select, &kD};
        insertLast(&temp);
        run(&temp);
        deleteNode(&temp);
      }
      if(*(measurementSelectionPtr) == 1) {
        TCB temp2 = {&menu, &kD};
```

```
      insertLast(&temp2);
      run(&temp2);
      deleteNode(&temp2);
      // menu(&kD);

    } else if(*(alarmAcknowledgePtr) == 1) {
        *ModePtrr = 0;
        TCB temp3 = {&anno, &kD};
        insertLast(&temp3);
        run(&temp3);
        deleteNode(&temp3);
        // anno(&kD);
        if (time2 - time1 > 2000) {
          (*disp.myTask)(disp.taskDataPtr);
          startF = true;
        }
        if (startF == true) {
          flash();
        }


    } else if (*(displaySelectionPtr) == 1) {
       *ModePtrr = 1;
        Measurement(&kD);
        if (time2 - time1 > 2000) {
            (*disp.myTask)(disp.taskDataPtr);
            startF = true;
        }
        if (startF == true) {
            flash();
        }
    }

    Serial.println(*(displayTMPtr));

    if (*(displayTMPtr) == 1) {
        Serial.print("blank");
        blank();
        //*(displayTMPtr) = 0;
    } else if (*(displayEmergencyPtr) == 1) {
        callDoctor();
    } else if (*(displayGamePtr) == 1) {
        game();
        //*(displayGamePtr) = 0;
    }

  }

  if(time2 - time1 > 2000) {
    schedulerTest();
    time1 = time2;

  } else {

  }

//   if (n % 2 == 0) {
//     generateEKG();
//     Serial.println("?");
```

```
//    }
//    Serial.println(EKGFreqBuf[0]);
}
```

10.3.2 UNO Main

```
//initialize values to measure frequency
const byte interruptPin = 2;
#define delayTimeSec .5
// Pulse Rate& respRate
unsigned int freqCount = 0;
unsigned int freqPrevious = 0;

//initalize GPIO values to measure temperature
int analogPin = A0;
int analogVal = 0;

//initialize GPIO values to measure blood pressure
int buttonPin = 7;
int switchPin = 9;
int buttonVal = 0; //if buttonVal = 0 blood pressure doesn't change
int increment = 1; //if increment=1 buttonval=1 the cuff inflates
                   //if increment=0 and buttonval=1 the cuff deflates
char incoming;
char fun;

int temperatureRaw = 30;
int systolicPressRaw = 80;
int diastolicPressRaw = 80;
int pulseRateRaw = 70;
int respRate = 40;

void setup() {
  Serial.begin(4800);
  pinMode(interruptPin, INPUT);
  pinMode(buttonPin, INPUT);
  pinMode(switchPin, INPUT);
  attachInterrupt(digitalPinToInterrupt(interruptPin), incrementFrequency, RISING);
}

void loop() {
  Measure();
  communications(); //Kaiser's Code
}
//main function that gets looped
void Measure() {
 temperatureRaw = getTemp();
 systolicPressRaw = getBP(1);
 diastolicPressRaw = getBP(0);
 pulseRateRaw = getRawPulseRate();
 respRate = getRespirationRate();
}

void communications() {
    if(Serial.available()) {
        incoming = Serial.read();
    }
```

```
    if(incoming == 's') {
    if(Serial.available()) {
      incoming = Serial.read();
    }
      while(incoming != 'e') {

        if(incoming == 't') {
          fun = Serial.read();
          // Read in function name
          if(fun == 'p'){
          Serial.write(temperatureRaw);
          }
        }
        if(incoming == 'b') {
          fun = Serial.read();
          // Read in function name
          if(fun == 'u'){
          Serial.write(systolicPressRaw);
          Serial.write(diastolicPressRaw);
          }
        }
        if(incoming == 'p') {
          fun = Serial.read();
          // Read in function name
          if(fun == 'l'){
          Serial.write(pulseRateRaw);
          }
          // if(mePR) { mePR = false; } else { mePR = true; }
        }
        if(incoming == 'r') {
          fun = Serial.read();
          // Read in function name
          if(fun == 'l'){
          Serial.write(respRate);
          // if(meR) { meR = false; } else { meR = true; }
          }
        }
        // if(Serial.available()) {
            incoming = Serial.read();
        // } else {
        //   break;
        // }
      }
    }
}



//returns the frequency value of the function generator
int measureFreq() {
    delay(1000 * delayTimeSec);
    unsigned int freq = freqCount - freqPrevious;
    freqPrevious = freqCount;
    //Serial.println(freq);
    return freq*2;
}


//attach interrupt; increases the frequency count by 1
```

```
void incrementFrequency() {
  freqCount += 1;
}

////gets raw pulse rate by getting frequency values (in Hz) and scaling
unsigned int rawPulseRate = 0;
int getRawPulseRate() {
  int frequencyPulse = measureFreq();
  rawPulseRate = frequencyPulse;
  if (rawPulseRate > 64 ) { //upper limit corrected = 200 so upper limit raw = 64
    rawPulseRate = 64;
  }
  if (rawPulseRate < 1) { //lower limit corrected = 10 so lower limit raw = 1
    rawPulseRate = 1;
  }
  return rawPulseRate;
  //Serial.print(rawPulseRate);
}

//gets raw respiration rate by getting frequency values (in Hz) and scaling
unsigned int rawRespirationRate = 0;
int getRespirationRate() {
  int frequencyPulse = measureFreq();
  rawRespirationRate = frequencyPulse/4;
  if (rawRespirationRate > 14 ) { //upper limit corrected = 50 so upper limit raw = 14
    rawRespirationRate = 14;
  }
  if (rawRespirationRate < 1) { //lower limit corrected = 10 so lower limit raw = 1
    rawRespirationRate = 1;
  }
  return rawRespirationRate;
}


//gets Temp by reading value on analog pin A0 which reads a value between 0 and 1024
//and scaling the result
int getTemp() {
  analogVal = analogRead(analogPin);
  int temp = 20 + (analogVal/37); //scaled it so min raw = 20 max raw = 47
  return temp;

}

unsigned int SysP = 50;
unsigned int DiaP = 29;
unsigned int previousbuttonVal = 0;
int getBP (boolean sys) {
  // delay(5);
  buttonVal = digitalRead(buttonPin);
  increment = digitalRead(switchPin);
  if (previousbuttonVal == 0 & buttonVal == 1) {
    previousbuttonVal = 1;
    if (increment == 1 ) {
      SysP = SysP * 1.1;
      DiaP = DiaP * 1.1;
    } else {
      SysP = SysP * .9;
      DiaP = DiaP * .9;
    }
```

```
  }
  if (previousbuttonVal == 1 & buttonVal == 0) {
    previousbuttonVal = 0;
  }
  if (SysP < 50 ) { //Corrected Systolic lower limit = 110, so systolic raw lower limit = 50
    SysP = 50;
  } else if (SysP > 70) { //Corrected Systolic upper limit = 150, so systolic raw upper limit = 70
    SysP = 70;
  }
  if (DiaP < 29 ) { //Corrected diastolic lower limit = 50, so diastolic raw lower limit = 29
    DiaP = 29;
  } else if (DiaP > 50) { //Corrected diastolic upper limit = 80, so diastolic raw upper limit = 50
    DiaP = 50;
  }
  if (sys == 1) {
    return SysP;
  }
  else return DiaP;
}
```

10.3.3 Mega Funtions

```
#include "tcb.h"
#include "helpers.h"

bool dispBP = TRUE;
bool dispT = TRUE;
bool dispPR = TRUE;
bool dispRR = TRUE;
bool dispEKG = TRUE;

bool flashBP = FALSE;
bool flashT = FALSE;
bool flashPR = FALSE;

//bool flashBP = FALSE;
//bool flashT = TRUE;
//bool flashPR = TRUE;

bool refSelect = TRUE;
bool refMenu = TRUE;
bool refAnnu = TRUE;
bool refMeas = TRUE;
bool refDisp = TRUE;
bool ref = TRUE;
bool refCD = TRUE;
bool refG = TRUE;

bool genEKG = TRUE;

bool TSelected = FALSE;
bool BPSelected = FALSE;
bool PRSelected = FALSE;
bool RRSelected = FALSE;
bool EKGSelected = FALSE;
bool Disp = TRUE;
```

```
bool Disp2 = TRUE;

// By high we mean 15% out of range
bool bpHigh = FALSE;
bool tempHigh = FALSE;
bool prHigh = FALSE;
bool rrLow = FALSE;
bool rrHigh = FALSE;
bool EKGHigh = FALSE;

unsigned char bpOutOfRange = 0;
unsigned char tempOutOfRange = 0;
unsigned char pulseOutOfRange = 0;
unsigned char rrOutOfRange = 0;
unsigned char EKGOutOfRange = 0;



bool trIsReverse = FALSE, prIsReverse = FALSE, isEven = TRUE;



/*
*    @para: void* dataPtr, we assume it's MeasureData pointer; integer isEven, check if it's even;
*    isEven range: 0 or 1;
*    Increase temperatureRaw by 2 even number time, decrease by 1 odd times called before reach 50;
     0 is even;
*    Then reverse the process until temperatureRaw falls below 15
*    systolicPressRaw - even: Increase by 3; - odd: decrease by 1; Range: no larger than 100
*    diastolicPressRaw -even: decrease by 2; - odd: increase by 1; Range: no less than 40
*    pulseRateRaw - even: decrease by 1; - odd: increase by 3; Range: 15-40
*    April, 22, 2019 by Kaiser
*/
void Measure(void* dataPtr)
{
    MeasureData md = *((MeasureData*) dataPtr);
    Serial1.write('s');
    if(dispT) {
            Serial1.write('t');
            Serial1.write('p');
            if(Serial1.available()) {
                int newTemp = Serial1.read();
                if(moreThan15(newTemp, md.temperatureRawBuf)) {
                    shift(newTemp, 8, (md.temperatureRawBuf));
                }
            }
    }

    if(dispBP) {
        Serial1.write('b');
        Serial1.write('u');
        if(Serial1.available()) {
            int newSys = Serial1.read();
            int newDia = Serial1.read();
            if(moreThan15(newSys, (md.bloodPressRawBuf) + 1) && moreThan15(newDia,
                md.bloodPressRawBuf)){
                shift(newSys, 16, (md.bloodPressRawBuf));
                shift(newDia, 16, (md.bloodPressRawBuf));
            }
```

```
        }
    }
    if(dispPR) {
        Serial1.write('p');
        Serial1.write('l');
        if(Serial1.available()) {
            int newPr = Serial1.read();
            if(moreThan15(newPr, md.pulseRateRawBuf)) {
                shift(newPr, 8, (md.pulseRateRawBuf));
            }
        }
    }
    if(dispRR) {
        Serial1.write('r');
        Serial1.write('l');
        if(Serial1.available()) {
            int newRR = Serial1.read();
            if(moreThan15(newRR, md.pulseRateRawBuf)) {
                shift(newRR, 8, (md.respirationRateRawBuf));
            }
        }
    }
    Serial1.write('e');
    delay(200);
    //generateEKG();
    return;
}


double Fs = 0;

int i;
int fft = 0;
double f0 = 0;
int m_index = 0;
void generateEKG() {
    f0 = 200;
    Fs = 2.2 * f0;

    for (i = 0; i < 256; i++) {
        EKGRawBuf[i] = 32*sin(2*3.14*f0*i/Fs);
        //Serial.println(EKGRawBuf[i]);
        EKGImgBuf[i] = 0;
    }
    delay(1000);
    //Serial.println(EKGRawBuf[0]);
    m_index = optfft(EKGRawBuf, EKGImgBuf);
    fft = Fs *m_index /256 + 1;
    Serial.println(fft);
    shiftChar(fft, 16, EKGFreqBuf);
    Serial.println(EKGFreqBuf[0]);
    return;

}



/*
*    @para: generic pointer dataPtr;
```

```
*    Assume the data pointer is of type ComputeData
*    Compute the corrected values of data
*    April 23, 2019 by Kaiser Sun
*    Change the type of corrected to char[]
*    April 24, 2019 by Kaiser Sun
*    May 24,2019 modified by Xinyu
*/
void Compute(void* dataPtr) {
    ComputeData comd = *((ComputeData*) dataPtr);
    //EKGData EKGD = *((EKGData*) data);
    if(dispT) {
        int correctedTemp = (*(comd.temperatureRawBuf)) * 0.75 + 5;
        shiftChar(correctedTemp, 8, (comd.tempCorrectedBuf));
    }
    if(dispBP) {
        int correctedDia = (*(comd.bloodPressRawBuf)) * 1.5 + 6;
        int correctedSys = (*(comd.bloodPressRawBuf + 1)) * 2 + 9;
        shiftChar(correctedSys, 16, (comd.bloodPressCorrectedBuf));
        shiftChar(correctedDia, 16, (comd.bloodPressCorrectedBuf));
    }
    if(dispPR) {
        int correctedPr = (*(comd.pulseRateRawBuf)) * 3 + 8;
        shiftChar(correctedPr, 8, (comd.prCorrectedBuf));
    }
    if(dispRR) {
        shiftChar((*(comd.respirationRateRawBuf))*3 + 7, 8, (comd.respirationRateCorBufPtr));
    }
    if (dispEKG && genEKG) {
        generateEKG();
        genEKG = false;
    }
    return;
}

int countt = 0;

/*
*    @para: generic pointer dataPtr;
*    Assume the data pointer is of type DisplayData
*    Display the data on the TFT display
*    April 23, 2019 by Kaiser Sun
*    May 24, 2019 modified by Xinyu
*/
int index = 15;
void Display(void* dataPtr) {
    // TODO: change the color of display!
    // Serial.println("run display");
    index = 15;
    DisplayData dd = *((DisplayData*) dataPtr);
    // Setup of tft display
    if (refDisp) {
      tft.setCursor(0, 0);
      tft.setTextColor(CYAN);
      tft.setTextSize(2);
      tft.println("      Mobile Doctor");
      tft.setTextColor(WHITE);
      //tft.setTextSize(1.9);
      if (dispBP) {
        tft.println("Systolic Pressure: ");
```

```
    tft.println("Diastolic Pressure: ");
  }
  if (dispT) {
    tft.println("Temperature: ");
  }
  if (dispPR) {
    tft.println("Pulse Rate: ");
  }
  if (dispRR) {
    tft.println("Respiration Rate: ");
  }
  if (dispEKG) {
    tft.println("EKG: ");
  }
  if (*(dd.Mode) == 0) {
    tft.print("Battery: ");
  }
  refDisp = false;
}

tft.fillRect(225,15,400,118,BLACK);
tft.setTextSize(2);

// Display Pressure
if(dispBP) {
    if(bpOutOfRange == 0) {
        tft.setTextColor(GREEN);
        flashBP = false;
    } else if (bpHigh) {
      tft.setTextColor(RED);
      flashBP = false;
    } else {
       tft.setTextColor(YELLOW);
       flashBP = true;
    }

    tft.setCursor(225, index);
    BPindex = index;
    tft.print(*(dd.bloodPressCorrectedBuf + 1));
    tft.print(" mmHg");
    index += 16;
    tft.setCursor(225, index);
    tft.print(*(dd.bloodPressCorrectedBuf));
    tft.print(" mmHg");

}

// print temperature
if(dispT) {
    if(tempOutOfRange == 0) {
        tft.setTextColor(GREEN);
        flashT = false;
    } else {
        if(tempHigh) {
          tft.setTextColor(RED);
          flashT = false;
        } else {
          tft.setTextColor(YELLOW);
          flashT = true;
```

```
        }
    }

    index += 16;
    Tindex = index;
    tft.setCursor(225, index);
    tft.print(*(dd.tempCorrectedBuf));
    tft.print(" C");
}

// Display pulse
if(dispPR) {
    if(pulseOutOfRange == 0) {
        tft.setTextColor(GREEN);
        flashPR = false;
    } else {
        if(prHigh) {
          tft.setTextColor(RED);
          flashPR = false;
        } else {
          tft.setTextColor(YELLOW);
          flashPR = true;
        }
    }

    index += 16;
    PRindex = index;
    tft.setCursor(225, index);
    tft.print(*(dd.prCorrectedBuf));
    tft.print("BPM");
}

// Display respiration rate
if(dispRR) {
    if(rrOutOfRange == 0) {
        tft.setTextColor(GREEN);
    } else {
        if(rrHigh) {
          tft.setTextColor(RED);
        } else {
          tft.setTextColor(YELLOW);
        }
    }
    index += 16;
    tft.setCursor(225, index);
    tft.print(*(dd.respirationRateCorrectedBuf));
    tft.print(" /s");
}

if (dispEKG) {
    if (EKGOutOfRange == 0) {
        tft.setTextColor(GREEN);
    } else {
        if (EKGHigh) {
            tft.setTextColor(RED);
        } else {
            tft.setTextColor(YELLOW);
        }
    }
```

```
        index += 16;
        tft.setCursor(225, index);
        tft.print(*(dd.EKGFreqBuf));
        tft.print("Hz");

    }


    // Display battery status
    if(*(dd.batteryStatePtr) > 20) {
        tft.setTextColor(GREEN);
    } else {
        tft.setTextColor(RED);
    }

    if (*(dd.Mode) == 0) {
//      tft.print("Battery: ");
      index += 16;
      tft.setCursor(225, index);
      tft.println(*(dd.batteryStatePtr));
      tft.setTextSize(1.5);
      tft.setTextColor(RED);
      tft.fillRect(0,130,400,30,BLACK);
      tft.setCursor(0, 135);
      if(tempHigh) {
          tft.println("Your body temperature is too high! Calm down QWQ");
      }

      if (bpHigh) {
          tft.println("Your blood pressure is too high! Calm down QWQ");
      }

      if(prHigh) {
          tft.println("Your heart beat is too slow. Do something exciting 0w0");
      }
    }

    // End
    tft.setTextSize(1.4);
    tft.setTextColor(CYAN);
    tft.setCursor(0, 163);
    tft.println("                CSE 474 Inc.");
    return;
}


/*
*    @param: generic pointer dataPtr;
*    assume the dataPtr is of type dataPtr;
*    if the data are out of range, diplay with red;
*    April 23, 2019 by Kaiser Sun
*/
void WarningAlarm(void* dataPtr) {
    WarningAlarmData wad = *((WarningAlarmData*) dataPtr);
    if (*(wad.temperatureRawBuf) > 43.7*1.05 || *(wad.temperatureRawBuf) < 41.5*0.95) {
        tempOutOfRange = 1;
        tempHigh = isTHight(float(*(wad.temperatureRawBuf)));
        if(tempHigh) { tWCounter++; }
    } else {
```

```
        tempOutOfRange = 0;
    }
    if(*(wad.bloodPressRawBuf + 1) > 60.5*1.05 || *(wad.bloodPressRawBuf) > 49.3*1.05 ||
        *(wad.bloodPressRawBuf + 1) < 55.5*0.95 || *(wad.bloodPressRawBuf) < 42.7*0.95) {
        bpOutOfRange = 1;
        //sys: 1, Dia: 0
        bpHigh = isBPHigh(*(wad.bloodPressRawBuf + 1), *(wad.bloodPressRawBuf));
        if(bpHigh) { bpWCounter++; }
    } else {
        bpOutOfRange = 0;
    }
    if(*(wad.pulseRateRawBuf) < 17.3*1.05 || *(wad.pulseRateRawBuf) > 30.7*0.95) {
        pulseOutOfRange = 1;
        prHigh = isPRHigh(float(*(wad.pulseRateRawBuf)));
        if(prHigh) { prWCounter++; }
    } else {
        pulseOutOfRange = 0;
    }
    if(*(wad.rrRawBuf) < 1.67*0.95 || *(wad.rrRawBuf) > 6*1.05) {
        rrOutOfRange = 1;
        rrHigh = isRRHigh(float(*(wad.rrRawBuf)));
        if(rrHigh) {rrWCounter++;}
    } else {
        rrOutOfRange = 0;
    }
    if(*(wad.EKGFreqBuf) < 35*0.95 || *(wad.EKGFreqBuf) > 3750*1.05) {
        EKGOutOfRange = 1;
        EKGHigh = isEKGHigh(float(*(wad.EKGFreqBuf)));
        if(EKGHigh) {ekgWCounter++;}
    } else {
      EKGOutOfRange = 0;
    }
    return;
}
/*
*    @param: generic pointer dataPtr;
*    Assume the dataPtr is of type StatusData;
*    BatteryState shall decrease by 1 each it is called;
*    April 23, 2019 by Kaiser Sun
*/
void Status(void* dataPtr) {
    StatusData sd = *((StatusData*) dataPtr);
    if(*(sd.batteryStatePtr) > 0) {
        *(sd.batteryStatePtr) -= 1;
    }
    return;
}



/*
*   This function is called when the TFT is in menu mode
*   May 9, 2019 by Kaiser
*   May 24, 2019 modified by Xinyu
*/
void menu(KeypadData* dataPtr) {
    //refMenu = false;
    KeypadData d = *dataPtr;
```

```
if (refMenu) {
  tft.setCursor(0, 0);
  tft.fillScreen(BLACK);
  drawSub(70, 0, dispBP);
  drawSub(70, 48, dispPR);
  drawSub(70, 96, dispT);
  drawSub(70, 144, dispRR);
  drawSub(70, 192, dispEKG);
  tft.fillRect(0, 0, 70, 240, MAGENTA);
  tft.setTextSize(2);
  tft.setTextColor(BLACK);
  tft.setCursor(5, 100);
  tft.print("Exit");
  refMenu = false;
}
if (TSelected) {
  drawSub(70, 96, dispT);
  TSelected = false;
}
if (BPSelected) {
  drawSub(70, 0, dispBP);
  BPSelected = false;
}
if (PRSelected) {
  drawSub(70, 48, dispPR);
  PRSelected = false;
}
if (RRSelected) {
  drawSub(70, 144, dispRR);
  RRSelected = false;
}
if (EKGSelected) {
  drawSub(70, 192, dispEKG);
  EKGSelected = false;
}

// Get point
digitalWrite(13, HIGH);
TSPoint p = ts.getPoint();
digitalWrite(13, LOW);
pinMode(XM, OUTPUT);
pinMode(YP, OUTPUT);
// If we have point selected
if (p.z > ts.pressureThreshhold) {
// scale from 0->1023 to tft.width
    p.x = (tft.width() - map(p.x, TS_MINX, TS_MAXX, tft.width(), 0));
    p.y = (tft.height()-map(p.y, TS_MINY, TS_MAXY, tft.height(), 0));

    if(T(p.x, p.y)) {
        TSelected = true;
        if(dispT) {
            dispT = FALSE;
        } else {
            dispT = TRUE;
        }
    }
    if(BP(p.x, p.y)) {
        BPSelected = true;
        if(dispBP) {
```

```
                dispBP = FALSE;
            } else {
                dispBP = TRUE;
            }
        }
        if(PR(p.x, p.y)) {
            PRSelected = true;
            if(dispPR) {
                dispPR = FALSE;
            } else {
                dispPR = TRUE;
            }
        }
        if(RR(p.x, p.y)) {
            RRSelected = true;
            if(dispRR) {
                dispRR = FALSE;
            } else {
                dispRR = TRUE;
            }
        }
        if (EKG(p.x, p.y)) {
            EKGSelected = true;
            if (dispEKG) {
              dispEKG = false;
            } else {
              dispEKG = true;
            }
        }
        if(QUIT1(p.x, p.y)) {
            *(d.measurementSelectionPtr) = 0;
            refSelect = true;
            refDisp = true;
            //Select((void*) &d);
        }
    }
    //refMenu = true;
    return;
}

/*
*    @param: KeypadData pointer, dataPtr
*    Submethod of select; when called, goes into announciation mode
*    May 10th by Kaiser
*    May 24, 2019 modified by Xinyu
*/
void anno(KeypadData* dataPtr) {
    //refAnnu = false;
    KeypadData d = *dataPtr;
    // Draw exit button

    if (refAnnu) {
      tft.fillScreen(BLACK);
      tft.fillRect(0, 180, 330, 60, RED);
      tft.setTextSize(2);
      tft.setTextColor(BLUE);
      tft.setCursor(150, 200);
      tft.print("Exit2");
```

```
      refAnnu = false;
    }
    //(*disp.myTask)(disp.taskDataPtr);

    // getPoint
    digitalWrite(13, HIGH);
    TSPoint p = ts.getPoint();
    digitalWrite(13, LOW);
    pinMode(XM, OUTPUT);
    pinMode(YP, OUTPUT);
    // If we have point selected
    if (p.z > ts.pressureThreshhold) {
        // scale from 0->1023 to tft.width
        p.x = (tft.width() - map(p.x, TS_MINX, TS_MAXX, tft.width(), 0));
        p.y = (tft.height()-map(p.y, TS_MINY, TS_MAXY, tft.height(), 0));
        if(QUIT2(p.x, p.y)) {
            *(d.alarmAcknowledgePtr) = 0;
            refSelect = true;
            refDisp = true;

        }
    }


    return;
}

/*
*   @param: generic pointer dataPtr
*   Assume the dataPtr is of type keyData
*   Goes into mode select; from mode select, we can select or
*   get announciation data
*   May 8, 2019 by Kaiser Sun
*   May 9, 2019 rewrote by Kaiser
*   May 24, 2019 modified by Xinyu
*/
void Select(void* dataPtr) {
    //refSelect = false;
    KeypadData kd = *((KeypadData*) dataPtr);
    // When it's select mode

       if (refSelect) {
          tft.fillScreen(WHITE);
          tft.fillCircle(55, 60, 50, GREEN);
          // Not used button
          tft.fillCircle(160, 60, 50, CYAN);
          // Upper: not used button
          tft.setTextSize(2);
          tft.setTextColor(WHITE);
          tft.setCursor(30, 55);
          tft.print("Menu");
          tft.fillCircle(265, 60, 50, YELLOW);
          // Not used button
          tft.setCursor(140, 55);
          tft.print("Meas");
          //tft.setTextColor(WHITE);
          tft.fillCircle(55, 180, 50, PINK);
          tft.setCursor(40, 175);
          tft.print("TM");
```

```
      tft.fillCircle(160, 180, 50, PURPLE);
      tft.setCursor(120, 175);
      tft.print("Call Dr");
      tft.fillCircle(265, 180, 50, BLUE);
      tft.setCursor(245, 175);
      tft.print("Game");
      // Upper: not used button
      tft.setCursor(230, 55);
      tft.print("Announ");
      refSelect = false;
    }
    digitalWrite(13, HIGH);
    TSPoint p = ts.getPoint();
    digitalWrite(13, LOW);
    pinMode(XM, OUTPUT);
    pinMode(YP, OUTPUT);
    if (p.z > ts.pressureThreshhold) {
    // scale from 0->1023 to tft.width
        p.x = (tft.width() - map(p.x, TS_MINX, TS_MAXX, tft.width(), 0));
        p.y = (tft.height()-map(p.y, TS_MINY, TS_MAXY, tft.height(), 0));
        //Serial.println(MENU(p.x, p.y));
        if(ANN(p.x, p.y)) {
            // If it's announciation, turn to display
            *(kd.alarmAcknowledgePtr) = 1;
            refAnnu = true;
            refDisp = true;
        }
        if(MENU(p.x, p.y)) {
            *(kd.measurementSelectionPtr) = 1;
            refMenu = true;
            refDisp = true;
        }
        if (MEAS(p.x, p.y)) {
           *(kd.displaySelectionPtr) = 1;
           refDisp = true;
           refMeas = true;
        }
        if (TM (p.x, p.y)) {
           *(kd.displayTMPtr) = 1;
           refDisp = true;
           ref = true;
           //Serial.println(*(kd.displayTMPtr));
           //blank();
        }
        if (EM (p.x, p.y)) {
          *(kd.displayEmergencyPtr) = 1;
          refDisp = true;
          refCD = true;
          //blank();
        }
        if (Game (p.x, p.y)) {
          *(kd.displayGamePtr) = 1;
          refDisp = true;
          refG = true;
          //blank();
        }
    }

return;
```

```
}

/*
*   Extra credit feature; It will call doctor when the button is pressed
*   June 8, Kaiser
*/
void callDoctor() {
    if (refCD) {
      //Serial.println("The patient is calling you!");

      tft.setTextSize(3);

      tft.fillScreen(BLACK);
      tft.fillRect(0, 180, 330, 60, RED);
      tft.setTextSize(2);
      tft.setTextColor(BLUE);
      tft.setCursor(150, 200);
      tft.print("Exit");
      tft.setTextColor(WHITE);
      tft.setCursor(10, 80);
      tft.print("Calling the Doctor......");
      delay(1000);
      tft.fillRect(0, 0, 330, 150, BLACK);
      tft.setCursor(10, 80);
      tft.print("Please wait for your doctor.");

      //Serial.println("The patient is calling you!");

      refCD = false;
    }

    digitalWrite(13, HIGH);
    TSPoint p = ts.getPoint();
    digitalWrite(13, LOW);
    pinMode(XM, OUTPUT);
    pinMode(YP, OUTPUT);

    // If we have point selected
    if (p.z > ts.pressureThreshhold) {
        // scale from 0->1023 to tft.width
        p.x = (tft.width() - map(p.x, TS_MINX, TS_MAXX, tft.width(), 0));
        p.y = (tft.height()-map(p.y, TS_MINY, TS_MAXY, tft.height(), 0));
        if(QUIT3(p.x, p.y)) {
            //*(displayTMPtr) = 0;
            *(displayEmergencyPtr) = 0;
            //*(displayGamePtr) = 0;
            refSelect = true;
            refDisp = true;
        }
    }

    return;
}

void game() {
    if (refG) {
      tft.setCursor(0, 20);
      tft.setTextColor(WHITE);
      tft.setTextSize(2);
```

```
      tft.fillScreen(BLACK);
      tft.print("Select the block that has green words to escape this page.");
      tft.setTextColor(BLUE);
      tft.setCursor(230, 120);
      tft.print("GREEN");
      // Green words
      tft.setTextColor(GREEN);
      tft.setCursor(10, 120);
      tft.print("BLUE");
      refG = false;
    }
        TSPoint p = ts.getPoint();
        digitalWrite(13, LOW);
        pinMode(XM, OUTPUT);
        pinMode(YP, OUTPUT);
        if (p.z > MINPRESSURE && p.z < MAXPRESSURE) {
        // scale from 0->1023 to tft.width
            p.x = (tft.width() - map(p.x, TS_MINX, TS_MAXX, tft.width(), 0));
            p.y = (tft.height()-map(p.y, TS_MINY, TS_MAXY, tft.height(), 0));
            if(G(p.x, p.y)) {
              //*(displayTMPtr) = 0;
              *(displayGamePtr) = 0;
              //*(displayGamePtr) = 0;
              refSelect = true;
              refDisp = true;
            }
        }
    return;
}

void blank() {
    if (ref) {
      tft.fillScreen(WHITE);
      tft.fillRect(0, 180, 330, 60, RED);
      tft.setTextSize(3);
      tft.setCursor(10, 90);
      tft.setTextColor(RED);
      tft.print("Make Way for Ambulance !!!");
      tft.setTextColor(BLUE);
      tft.setCursor(150, 200);
      tft.setTextSize(2);
      tft.print("Exit");
      ref = false;
      delay(500);
    } else {

      tft.fillCircle(150, 90, 60, RED);
      tft.fillRect(0, 0, 330, 180, WHITE);
    }
    digitalWrite(13, HIGH);
    TSPoint p = ts.getPoint();
    digitalWrite(13, LOW);
    pinMode(XM, OUTPUT);
    pinMode(YP, OUTPUT);

    // If we have point selected
    if (p.z > ts.pressureThreshhold) {
        // scale from 0->1023 to tft.width
        p.x = (tft.width() - map(p.x, TS_MINX, TS_MAXX, tft.width(), 0));
```

```
        p.y = (tft.height()-map(p.y, TS_MINY, TS_MAXY, tft.height(), 0));
        if(QUIT3(p.x, p.y)) {
            *(displayTMPtr) = 0;
            *(displayEmergencyPtr) = 0;
            *(displayGamePtr) = 0;
            refSelect = true;
            refDisp = true;
        }
    }

    return;
}


/*
*    @param: KeypadData pointer, dataPtr
*    Submethod of select; when called, goes into
*    May 24th by Xinyu
*/
void Measurement(KeypadData* dataPtr) {

    KeypadData d = *dataPtr;
    // Draw exit button

    if (refMeas) {
      tft.fillScreen(BLACK);
      tft.fillRect(0, 180, 330, 60, RED);
      tft.setTextSize(2);
      tft.setTextColor(BLUE);
      tft.setCursor(150, 200);
      tft.print("Exit3");
      refMeas = false;
    }

    // getPoint
    digitalWrite(13, HIGH);
    TSPoint p = ts.getPoint();
    digitalWrite(13, LOW);
    pinMode(XM, OUTPUT);
    pinMode(YP, OUTPUT);

    // If we have point selected
    if (p.z > ts.pressureThreshhold) {
        // scale from 0->1023 to tft.width
        p.x = (tft.width() - map(p.x, TS_MINX, TS_MAXX, tft.width(), 0));
        p.y = (tft.height()-map(p.y, TS_MINY, TS_MAXY, tft.height(), 0));
        if(QUIT3(p.x, p.y)) {
            *(d.displaySelectionPtr) = 0;
            refSelect = true;
            refDisp = true;
        }
    }

    return;
}


/*
*   @param: data pointer of communications data
```

```
*   Receive the command from remote subsystem
*   change the display status; 's' is start of message
*   'e' is end of message
*   Order: temperature, pressure, pulserate, respiration rate
*   May 29th, 2019 by Kaiser
*/
char t = 'q';
void Communications(void* dataPtr) {
    CommunicationsData cd = *((CommunicationsData*) dataPtr);
    if(Serial.available() > 0) {
        if(Serial.read() == 's') {
            int i = 0;
            t = Serial.read();
            while(t != 'e') {
                // When they did not read end of message, keep loop
                if(t == 't') {
                    Serial.write("Temperature: ");
                    Serial.println(*(cd.tempCorrectedBuf));
                    Serial.println(" ");
                    if(dispT) { dispT = false; } else {dispT = true; }
                }
                if(t == 'b') {
                    Serial.write("Systolic Pressure: ");
                    Serial.println(*(cd.bloodPressCorrectedBuf + 1));
                    Serial.println(" ");
                    Serial.write("Diastolic Pressure: ");
                    Serial.println(*(cd.bloodPressCorrectedBuf));
                    Serial.println(" ");
                    if(dispBP) { dispBP = false; } else { dispBP = true; }
                }
                if(t == 'p') {
                    Serial.write("Pulse Rate: ");
                    Serial.println(*(cd.prCorrectedBuf));
                    Serial.println(" ");
                    if(dispPR) { dispPR = false; } else { dispPR = true; }
                }
                if(t == 'r') {
                    Serial.write("Respiration Rate: ");
                    Serial.println(*(cd.respirationRateCorBufPtr));
                    Serial.println(" ");
                    if(dispRR) {dispRR = false; } else { dispRR = true; }
                }
                t = Serial.read();
            }
        }
        t = 'e';
    }
    return;
}


/*
*    @param: array of TCB taskQ
*    pre: assume length(taskQ) = 5
*    helper method of scheduler, used to measure the time
*    that each task takes to execute; should be commented
*    after development;
*    April 23, 2019 by Kaiser Sun
*    May 12, 2019 modified by Xinyu
```

```
*/
void schedulerTest() {
       TCB* taskQueue = front;

       while (taskQueue != NULL) {
           run(taskQueue);
           taskQueue = taskQueue->next;
       }
}

char* doctorName = "Kaiser";
char* patientName = "Shouta";
/*
*   Display data on remote system
*   Share the data with display
*   TODO: Will be called each 5 seconds
*/
void remoteCommunicationDisplay(void* dataPtr) {
  // Print out product name, doctor name, and patient name
  Serial.println("__________________Mobile__Doctor_____________________");
  Serial.println("____________0w0_Are_You_Healthy_Today?_________________");
  Serial.println("Mobile Doctor 0W0");
  Serial.println(doctorName);
  Serial.println(patientName);
  // Dereference data
  DisplayData dd = *((DisplayData*) dataPtr);
  // Display temperature
  if(dispT) {
    Serial.print("Temperature:     ");
    Serial.print(*(dd.tempCorrectedBuf));
    Serial.println(" C");
  }
  if(dispBP) {
    Serial.print("Systolic Pressure: ");
    Serial.print(*(dd.bloodPressCorrectedBuf + 1));
    Serial.println("mmHg");
    Serial.print("Diastolic Pressure: ");
    Serial.print(*(dd.bloodPressCorrectedBuf));
    Serial.println("mmHg");
  }
  if(dispPR) {
    Serial.print("Pulse rate:      ");
    Serial.print(*(pulseRateRawBuf));
    Serial.println("BPM");
  }
  if(dispEKG) {
    Serial.print("EKG:             ");
    // TODO: makesure if the EKG is right
    Serial.print(*(EKGFreqBuf));
    Serial.println("Hz");
  }
  Serial.print("Battery:          ");
  Serial.println(*(batteryStatePtrr));
  if(tempHigh) {
    Serial.println("Your body temperature is too high! Calm down QWQ");
  }

  if (bpHigh) {
    Serial.println("Your blood pressure is too high! Calm down QWQ");
```

```
  }

  if(prHigh) {
    Serial.println("Your heart beat is too slow. Do something exciting 0w0");
  }
  Serial.println("_____________________________________CSE474_Inc._____");
  Serial.println(" ");
  Serial.println(" ");
  return;
}


/*
*    Execute only at the Setup function once
*    Start the system timer, arrange the taskQueue
*    May 22, 2019 by Kaiser
*/
void startUp() {
    // Setup the data structs
  meaD = MeasureData{temperatureRawPtrr, bloodPressRawPtrr, pulseRateRawPtrr,
      respirationRateRawPtr, measurementSelectionPtr, EKGFreqBufPtr};
  //EKGD = EKGData{EKGRawBuf, EKGImgBuf, EKGFreqBuf};
  cD = ComputeData{temperatureRawPtrr, bloodPressRawPtrr, pulseRateRawPtrr, respirationRateRawPtr,
      EKGRawBufPtr, EKGImgBufPtr, EKGFreqBufPtr, tempCorrectedPtrr, bloodPressCorrectedPtrr,
      pulseRateCorrectedPtrr, respirationRateCorPtr,measurementSelectionPtr};
  dDa = DisplayData{ModePtrr, tempCorrectedPtrr, bloodPressCorrectedPtrr, pulseRateCorrectedPtrr,
      respirationRateCorPtr, EKGFreqBufPtr, batteryStatePtrr};
  wAD = WarningAlarmData{temperatureRawPtrr, bloodPressRawPtrr, pulseRateRawPtrr,
      respirationRateRawPtr, EKGFreqBufPtr, batteryStatePtrr};
  sD = StatusData{batteryStatePtrr};
  kD = KeypadData{localFunctionSelectPtr, measurementSelectionPtr, alarmAcknowledgePtr, commandPtr,
      remoteFunctionSelectPtr, measurementResultSelectionPtr, displaySelectionPtr, displayTMPtr,
      displayEmergencyPtr, displayGamePtr};
  comD = CommunicationsData{tempCorrectedPtrr, bloodPressCorrectedPtrr, pulseRateCorrectedPtrr,
      respirationRateCorPtr, EKGFreqBufPtr};
// Setup the TCBs
  meas = {&Measure, &meaD};
  //gene = {&generateEKG, &EKGD};
  comp = {&Compute, &cD};
  disp = {&Display, &dDa};
  alar = {&WarningAlarm, &wAD};
  stat = {&Status, &sD};
  remDisp = {&remoteCommunicationDisplay, &dDa};
  //keyp = {&Select, &kD};
  com = {&Communications, &comD};

  // Setup task queue
  insertLast(&meas);
  insertLast(&comp);
  //insertLast(&gene);
  insertLast(&stat);
  insertLast(&alar);
  insertLast(&com);
  //insertLast(&keyp);
  time1 = millis();
  timeb = time1;
  timeRD1 = time1;
  timeEmer1 = time1;
  // taskQueue[4] = disp;
```

```
  //run(&keyp);
  //generateEKG();

}


void flash() {
  if (dispBP && flashBP) {
    if (counterBP == 1) {
      tft.fillRect(225, BPindex, 400, 30, BLACK);
    } else {
      tft.setTextColor(YELLOW);
      tft.fillRect(225, BPindex, 400, 30, BLACK);
      tft.setTextSize(2);
      tft.setCursor(225, BPindex);
      tft.print(*(bloodPressCorrectedBuf + 1));
      tft.print(" mmHg");
      tft.setCursor(225, BPindex + 16);
      tft.print(*(bloodPressCorrectedBuf));
      tft.print(" mmHg");
    }
  }

  if (dispPR && flashPR) {
    if (counterPR == 4) {
      tft.fillRect(225, PRindex, 400, 15, BLACK);
    } else if (counterPR == 8){
      tft.setTextColor(YELLOW);
      tft.fillRect(225, PRindex, 400, 15, BLACK);
      tft.setTextSize(2);
      tft.setCursor(225, PRindex);
      tft.print(*(pulseRateCorrectedPtrr));
      tft.print("BPM");
    }
  }
  if (dispT && flashT) {
    if (counterT == 2) {
      tft.fillRect(225, Tindex, 400, 15, BLACK);
    } else if (counterT == 4) {
      tft.setTextColor(YELLOW);
      tft.fillRect(225, Tindex, 400, 15, BLACK);
      tft.setTextSize(2);
      tft.setCursor(225, Tindex);
      // Serial.println(Tindex);

      tft.print(*(tempCorrectedBuf));
      tft.print(" C");
    }
  }

}



/*
*    @param: int index, TCB* taskQ;
*    pre: index < length(taskQ);
*    helper method of sechdule function;
*    April 23, 2019 by Kaiser Sun
```

```
*/
void run(TCB* taskQ) {
    // Call the function in the taskQ;
    (*taskQ->myTask)(taskQ->taskDataPtr);
}

/*
*   Called when user having an emergency situation
*/
void emergency() {
  Serial.println("EMERGENCY!!!");
  // Display website
  Serial.println("https://www.911.gov/");
  tft.fillScreen(RED);
}
```

10.3.4 Data Structure

```
#ifndef STRUCTDEF
#define STRUCTDEF
#include <Elegoo_GFX.h> // Core graphics library
#include <Elegoo_TFTLCD.h> // Hardware-specific library
#include <TouchScreen.h> // TouchScreen library
// #include <Adafruit_TFTLCD.h>
// #include <Adafruit_GFX.h>
// April 23th by Kaiser Sun
// May 8th modified by Kaiser Sun, add touch screen
// May 24, 2019 modified by Xinyu


// pin assignments for TFT
// The control pins for the LCD can be assigned to any digital or
// analog pins...but we'll use the analog pins as this allows us to
// double up the pins with the touch screen (see the TFT paint example).
#define LCD_CS A3 // Chip Select goes to Analog 3
#define LCD_CD A2 // Command/Data goes to Analog 2
#define LCD_WR A1 // LCD Write goes to Analog 1
#define LCD_RD A0 // LCD Read goes to Analog 0

#define LCD_RESET A4 // Can alternately just connect to Arduino's reset pin

#define BLACK  0x0000
#define GREY   0x7BEF
#define BLUE   0x001F
#define RED    0xF800
#define GREEN  0x07E0
#define CYAN   0x07FF
#define MAGENTA 0xF81F
#define YELLOW 0xFFE0
#define WHITE  0xFFFF
#define ORANGE 0xFF8C00
#define NAVY   0x000F
#define MAROON 0x7800
#define OLIVE  0x7BE0
#define VIOLET 0x9199
#define PINK   0xF97F
#define PURPLE 0x780F
```

```
// touch screen settings
#define YP A2 // must be an analog pin, use "An" notation!
#define XM A3 // must be an analog pin, use "An" notation!
#define YM 8  // can be a digital pin
#define XP 9  // can be a digital pin

//Touch For New ILI9341 TP
#define TS_MINX 115
#define TS_MAXX 960

#define TS_MINY 125
#define TS_MAXY 920
// We have a status line for like, is FONA working
#define STATUS_X 65
#define STATUS_Y 10

// Set up bound for pressure
#define MINPRESSURE 5
#define MAXPRESSURE 1000

// Macro for location on screen
#define MENU(x, y) (y > 0) && (y < 110) && (x > 0) && (x < 115)
#define ANN(x, y) (y > 0) && (y < 110) && (x > 250)
#define MEAS(x, y) (y > 0) && (y < 110) && (x > 120) && (x < 245)
#define TM(x, y) (x > 0) && (x < 115) && (y > 130)
#define EM(x, y) (x > 120) && (x < 245) && (y > 130)
#define Game(x, y) (x > 250) && (y > 130)
#define QUIT1(x, y) (y > 0) && (y < 240) && (x > 0) && (x < 70)
#define T(x, y) (x > 70) && (y < 192) && (y > 128)
#define BP(x, y) (x > 70) && (y < 64) && (y > 0)
#define PR(x, y) (x > 70) && (y < 128) && (y > 64)
#define RR(x, y) (x > 70) && (y < 256) && (y > 192)
#define EKG(x,y) (x > 70) && (y > 256)
#define QUIT2(x, y) (x > 0) && (y > 180)
#define QUIT3(x, y) (x > 0) && (y > 180)
#define G(x, y) (y > 120) && (y < 140) && (x > 10) && (x < 50)
Elegoo_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);
// Adafruit_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);
TouchScreen ts = TouchScreen(XP, YP, XM, YM, 300);

#define M_PI acos(-1.0)

// Define bool tyoe
enum myBool {FALSE = 0, TRUE = 1};
typedef enum myBool Bool;

struct MyStruct
{
void (*myTask)(void*);
void* taskDataPtr;
struct MyStruct* next;
struct MyStruct* prev;
};
typedef struct MyStruct TCB;

//this link always point to first Link
TCB* front = NULL;
```

```
//this link always point to last Link
TCB* back = NULL;
void insertLast(TCB* node);
TCB* deleteNode(TCB* node);

// Declare the functions
    void Measure(void* dataPtr);
    void Compute(void* dataPtr);
    void Status(void* dataPtr);
    void WarningAlarm(void* dataPtr);
    void Display(void* dataPtr);
    void Communications(void* dataPtr);
    void Scheduler(TCB* taskQueue);
    void sechdulerTest();
    void startUp();
    //void Select(void* dataPtr);
typedef struct
{
    signed int EKGRawBuf;
    signed int EKGImgBuf;
    unsigned char EKGFreqBuf;
} EKGData;

typedef struct
{
    unsigned int* temperatureRawBuf;
    unsigned int* bloodPressRawBuf;
    unsigned int* pulseRateRawBuf;
    unsigned int* respirationRateRawBuf;
    unsigned short* measurementSelectionPtr;
    unsigned char* EKGFreqBuf;
} MeasureData;

typedef struct
{
    unsigned int* temperatureRawBuf;
    unsigned int* bloodPressRawBuf;
    unsigned int* pulseRateRawBuf;
    unsigned int* respirationRateRawBuf;
    unsigned int* EKGRawBuf;
    unsigned int* EKGImgBuf;
    unsigned char* EKGFreqBuf;
    unsigned char* tempCorrectedBuf;
    unsigned char* bloodPressCorrectedBuf;
    unsigned char* prCorrectedBuf;
    unsigned char* respirationRateCorBufPtr;
    unsigned short* measurementSelectionPtr;
} ComputeData;

typedef struct
{
    unsigned short* batteryStatePtr;
} StatusData;

typedef struct
{
  unsigned short* Mode;
  unsigned char* tempCorrectedBuf;
```

```
  unsigned char* bloodPressCorrectedBuf;
  unsigned char* prCorrectedBuf;
  unsigned char* respirationRateCorrectedBuf;
  unsigned char* EKGFreqBuf;
  unsigned short* batteryStatePtr;
} DisplayData;

typedef struct
{
    unsigned int* temperatureRawBuf;
    unsigned int* bloodPressRawBuf;
    unsigned int* pulseRateRawBuf;
    unsigned int* rrRawBuf;
    unsigned char* EKGFreqBuf;
    unsigned short* batteryStatePtr;
} WarningAlarmData;

typedef struct
{
    unsigned short* localFunctionSelectPtr;
    unsigned short* measurementSelectionPtr;
    unsigned short* alarmAcknowledgePtr;
    unsigned short* commandPtr;
    unsigned short* remoteFunctionSelectPtr;
    unsigned short* measurementResultSelectionPtr;
    unsigned short* displaySelectionPtr;
    unsigned short* displayTMPtr;
    unsigned short* displayEmergencyPtr;
    unsigned short* displayGamePtr;
} KeypadData;

typedef struct
{
    unsigned char* tempCorrectedBuf;
    unsigned char* bloodPressCorrectedBuf;
    unsigned char* prCorrectedBuf;
    unsigned char* respirationRateCorBufPtr;
    unsigned char* EKGFreqBuf;
} CommunicationsData;


# endif
```

10.4 Linked List

```
 * task.h
 *
 *  Created on: May 9, 2019
 *      Author: xinyu
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include "tcb.h"
```

```
// struct node {
//    TCB data;

//    struct node *next;
//    struct node *prev;
// };



bool isEmpty() {
   return front == NULL;
}

void insertLast(TCB* node) {

   //create a link
   //TCB* link = (TCB*) malloc(sizeof(TCB));

   //link->data = data;

   if(isEmpty()) {
      //make it the last link
    front = node;
      back = node;
   } else {
      //make link a new last link
      back->next = node;

      //mark old last node as prev of new link
      node->prev = back;
    back = node;
   }

   return;
}

//delete given item

TCB* deleteNode(TCB* node) {
  //start from the first link
     TCB* current = front;
     TCB* previous = NULL;

     //if list is empty
     if(front == NULL) {
        return NULL;
     }

     //navigate through list
     while(current != node) {
        //if it is last node

        if(current->next == NULL) {
           return NULL;
        } else {
           //store reference to current link
           previous = current;
```

```
         //move to next link
         current = current->next;
      }
   }

   //found a match, update the link
   if(current == front) {
      //change first to point to next link
      front = front->next;
   } else {
      //bypass the current link
      current->prev->next = current->next;
   }

   if(current == back) {
      //change last to point to prev link
      back = current->prev;
   } else {
      current->next->prev = current->prev;
   }

   return current;
}
```

10.4.1 Help Functions

Our Helper Function

```
 #include<stdio.h>
#include "tcb.h"

// the number of recording time; be int, count how many 0.5 sec has passed
int halfsec = 0;


// Time values, to determine whether to blink
unsigned long timef;



void shift(int newVal,int Bufsize,unsigned int* Buf) {
  for (int i = 1; i <= Bufsize-1; i++) {
    //if ( (*Buff)[Bufsize - (i+1)] != null)
   // if ( Buf[Bufsize - (i+1)] != NULL) { //if element 6 is not null
      *(Buf + (Bufsize - i)) = *(Buf + (Bufsize - (i+1))); //element 7 = element 6
    }
    *Buf = newVal;
  //}
}

void shiftChar(int newVal,int Bufsize,unsigned char* Buf) {
  for (int i = 1; i <= Bufsize-1; i++) {
    //if ( (*Buff)[Bufsize - (i+1)] != null)
   // if ( Buf[Bufsize - (i+1)] != NULL) { //if element 6 is not null
      *(Buf + (Bufsize - i)) = *(Buf + (Bufsize - (i+1))); //element 7 = element 6 //}
  }
    *Buf = newVal;
```

```
}


/*
*    @param: take in the systolic and diastolic BloodPressure
*    return true if it's high; false otherwise
*    May23, 2019 Kaiser
*/
bool isBPHigh(int sys, int dia) {
    if(sys > 60.5) {
        if(float(sys - 60.5)/60.5 > 0.2) {

          return true;
        }
        else { return false; }
    } else if(sys < 55.5) {
        if(float(55.5 - sys)/55.5 > 0.2) {

          return true;
        }
        else { return false; }
    }
    if(dia > 49.3) {
        if(float(dia - 49.3)/49.3 > 0.2) {

          return true;
        }
        else { return false; }
    } else {
        if(float(42.7 - dia)/42.7 > 0.2) {

          return true;
        }
        else { return false; }
    }
    return false;
}

/*
*   @param: float Temperature t
*   return true if Temperature high; false otherwise
*/
bool isTHight(float t) {
    return t > 43.7 && (t - 43.7)/43.7 > 0.15
           || t < 41.5 && (41.5 - t)/41.5 > 0.15;
}

/*
*   @param: float pulse rate pr
*    return true if it's too out of range(0.15); false otherwise
*/
bool isPRHigh(float pr) {
    return (pr > 30.7) && (pr - 30.7) / 30.7 > 0.15
           || (pr < 17.3) && (17.3 - pr) / 17.3 > 0.15;
}

/*
*   @param: float respiration rate rr
```

```
*   return true if it's too out of range(0.15); false otherwise
*/
bool isRRHigh(float rr) {
    return (rr > 6) && (rr - 6) / 6 > 0.15
        || (rr < 1.67) && (1.67 - rr) / 1.67 > 0.15;
}

bool isEKGHigh(float ekg) {
    return ekg > 3750 && (ekg - 3750)/3750 > 0.15
            || ekg < 35 && (35 - ekg)/35 > 0.15;
}



/*
*    @para: take in x, y, the center of the rectangle
*           bool d, to see whether it is unselected
*    Draw the buttons
*    (700, 250) -> T
*    (700, 500) -> BP
*    (700, 750) -> PR
*/
void drawSub(int x, int y, bool d) {
    Serial.print("Entered drawSub");
    if(!d) {
        tft.fillRect(x, y, 260, 48, RED);
    } else {
        tft.fillRect(x, y, 260, 48, GREEN);
    }
    // See Line595 of Elegoo_GFX.cpp
    tft.setTextSize(2);
    tft.setTextColor(BLACK);
//    tft.setCursor(x + 10, y + 10);
     tft.setCursor(x + 10, y + 10);
    if(y == 0) {
        tft.print("BloodPressure");
    } else if(y == 48) {
        tft.print("PulseRate");
    } else if(y == 96) {
        tft.print("Temperature");
    } else if (y == 144) {
        tft.print("RespirationRate");
    } else {
        tft.print("EKG");
    }
}

bool moreThan15(int val, unsigned int* Buf) {
    return float(abs(val - *Buf))/(*(Buf))> 0.15;
}
```

FFT Help Function

```
 /************************************************************************/
/* optfft.c                                                             */
/*                                                                      */
/* An optimized version of the fft function using only 16-bit integer math. */
/*                                                                      */
/* Optimized by Brent Plump                                             */
```

```
/* Based heavily on code by Jinhun Joung                                  */
/*                                                                        */
/* - Works only for input arrays of 256 length.                           */
/* - Requires two arrays of 16-bit ints. The first contains the samples, the */
/*   second contains all zeros. The samples range from -31 to 32          */
/* - Returns the index of the peak frequency                              */
/**************************************************************************/

#include "optfft.h"
#include "tables.h"
#define ABS(x) (((x)<0)?(-(x)):(x))
#define CEILING(x) (((x)>511)?511:(x))

signed int optfft(signed int real[256], signed int imag[256]) {
signed int i, i1, j, l, l1, l2, t1, t2, u;

  /* Bit reversal. */
  /*Do the bit reversal */
  l2 = 128;
  i=0;
  for(l=0;l<255;l++) {
    if(l < i) {
      j=real[l];real[l]=real[i];real[i]=j;
    }
    l1 = l2;
    while (l1 <= i){
      i -= l1;
      l1 >>= 1;
    }
    i += l1;
  }

  /* Compute the FFT */
  u = 0;
  l2 = 1;
  for(l=0;l<8;l++){
    l1 = l2;
    l2 <<= 1;
    for(j=0;j<l1;j++){
      for(i=j;i<256;i+=l2){
        i1 = i + l1;
        t1 = (u1[u]*real[i1] - u2[u]*imag[i1])/32;
        t2 = (u1[u]*imag[i1] + u2[u]*real[i1])/32;
        real[i1] = real[i]-t1;
        imag[i1] = imag[i]-t2;
        real[i] += t1;
        imag[i] += t2;
      }
      u++;
    }
  }

  /* Find the highest amplitude value */
  /* start at index 1 because 0 can hold high values */
  j=1;
  l=0;
  for ( i=1; i<(128); i++ ) {
    l1 = square2[CEILING(ABS(real[i]))]+square2[CEILING(ABS(imag[i]))];
    if (l1 > l) {
```

```
      j = i;
      l = l1;
    }
  }
  return (j);
}
```

10.5 Diagrams

10.5.1 Functional Diagrams for System Control

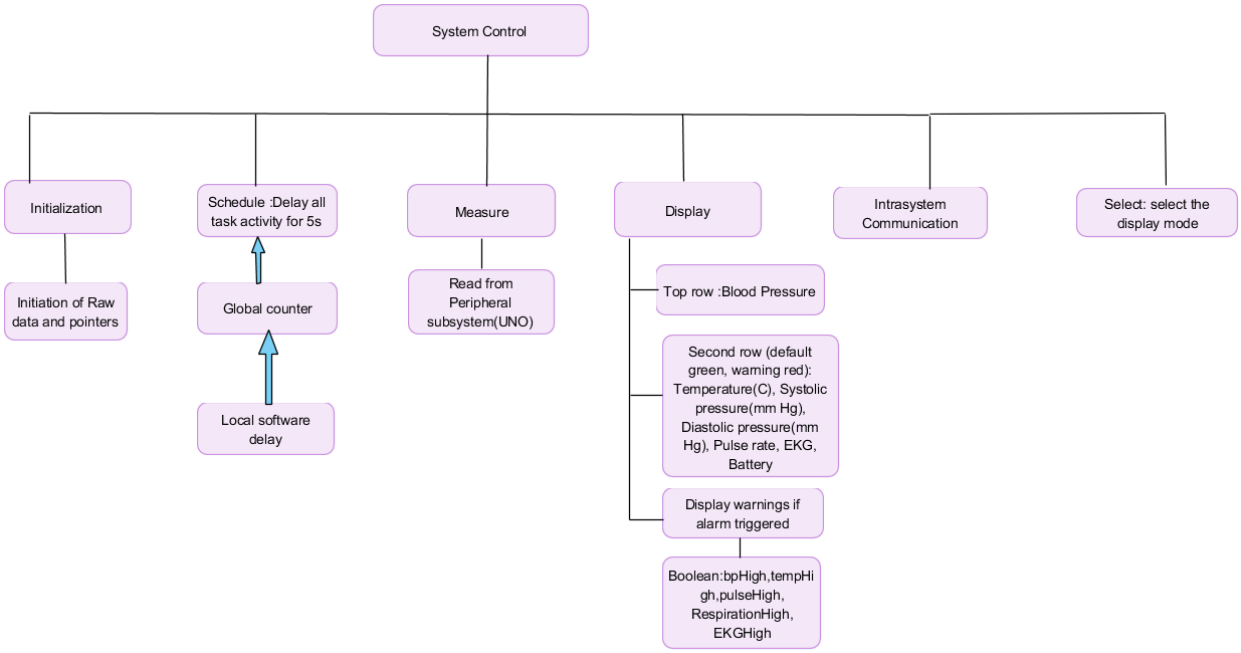

Figure 12: System Control

10.5.2 Functional Diagrams for Peripheral Subsystem

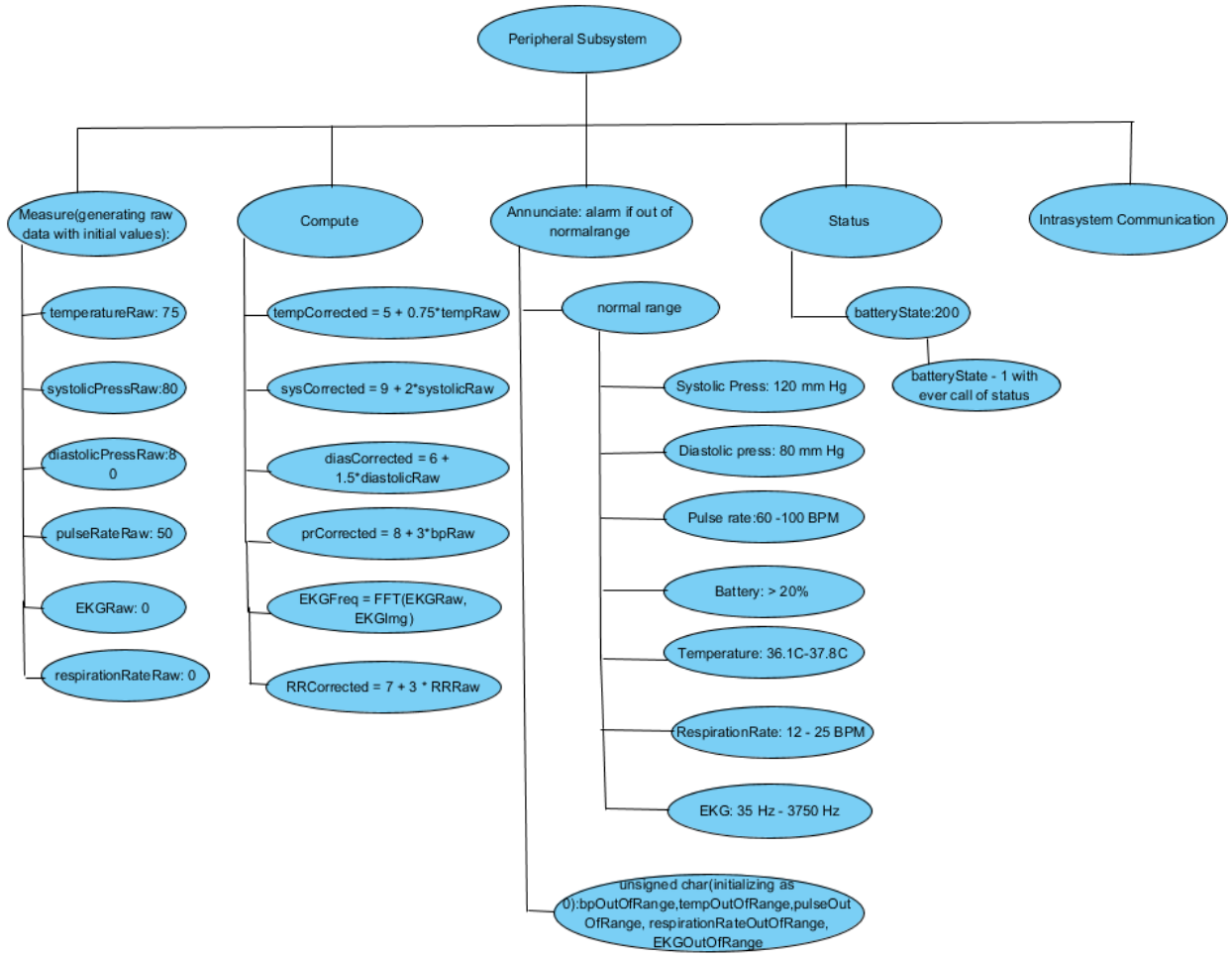

Figure 13: Peripheral Subsystem

10.5.3 Use Case for System Control

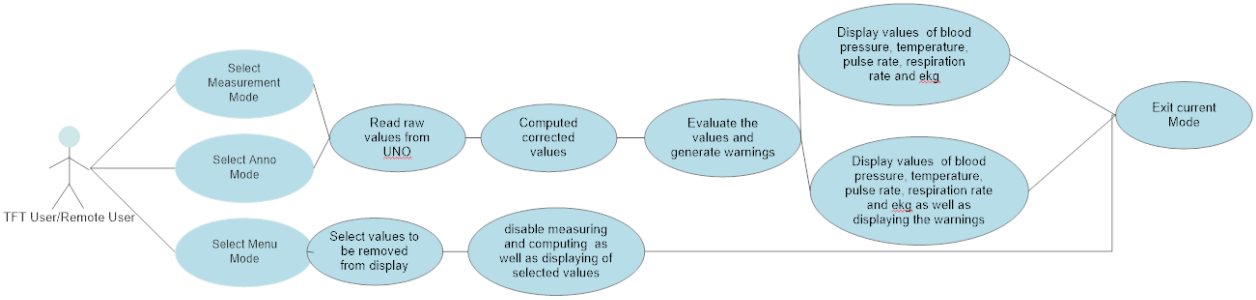

Figure 14: Mega Use Case

10.5.4 Use Case for Peripheral Subsystem

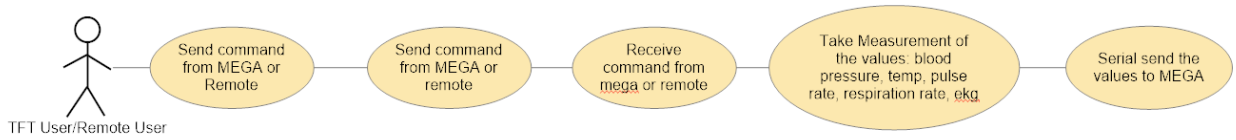

Figure 15: UNO Use case

10.5.5 Class Diagram for System Control

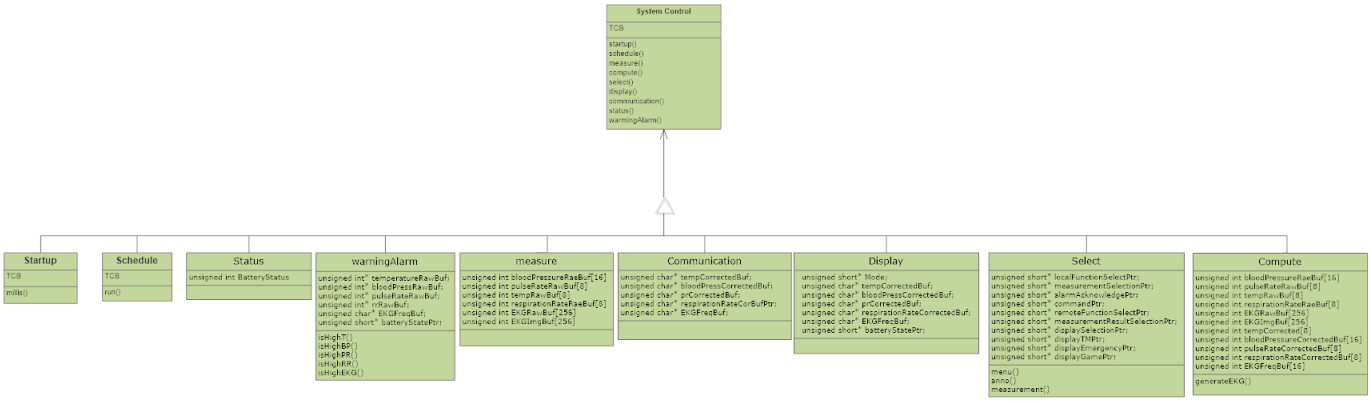

Figure 16: System Control

10.5.6 Class Diagram for Peripheral Subsystem (UNO)

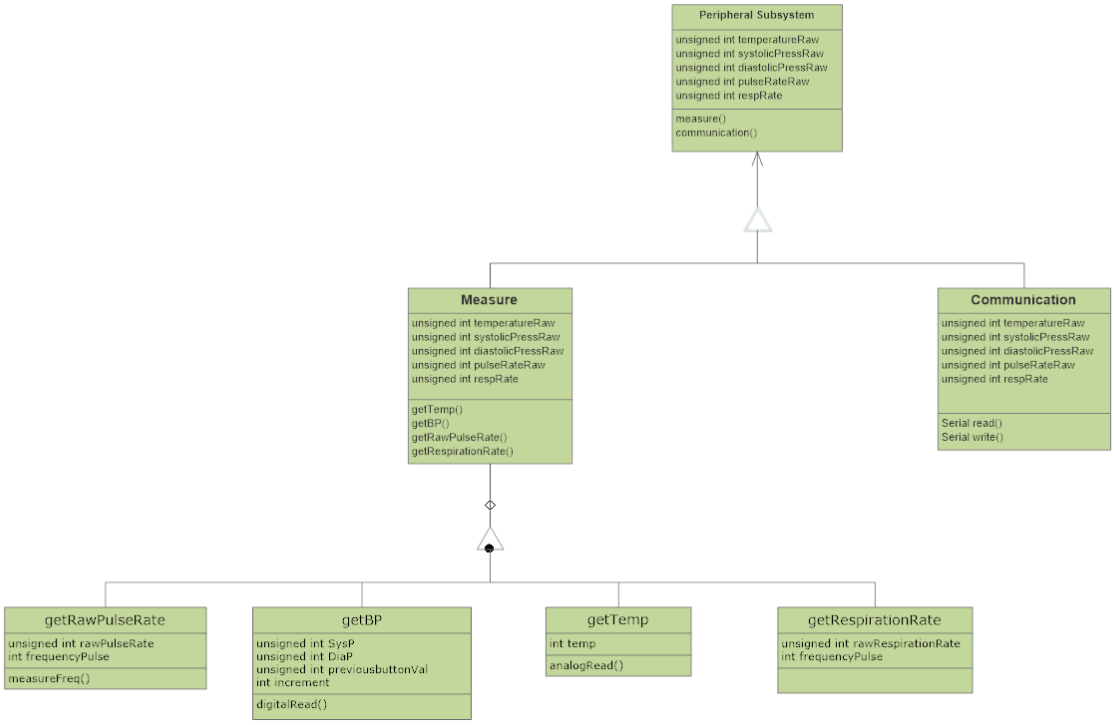

Figure 17: Peripheral Subsystem

10.5.7 Activity Diagram for Peripheral Subsystem

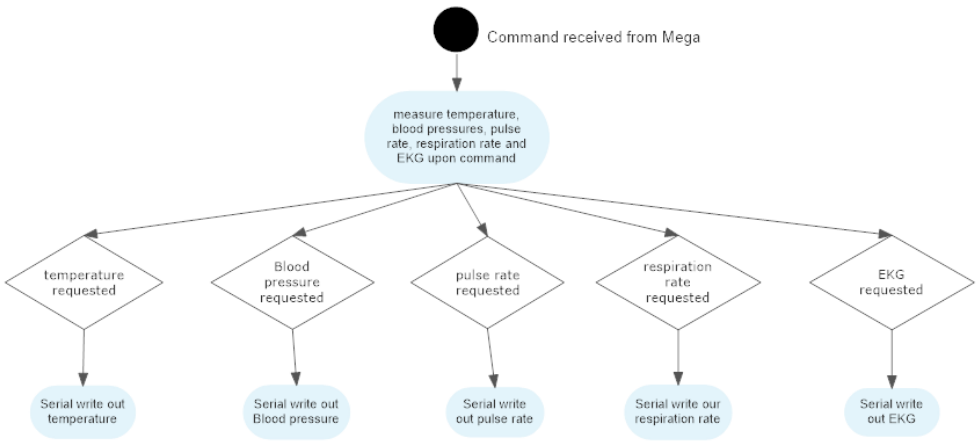

Figure 18: Activity Diagram for Peripheral Subsystem

10.5.8 Activity Diagram for System Control

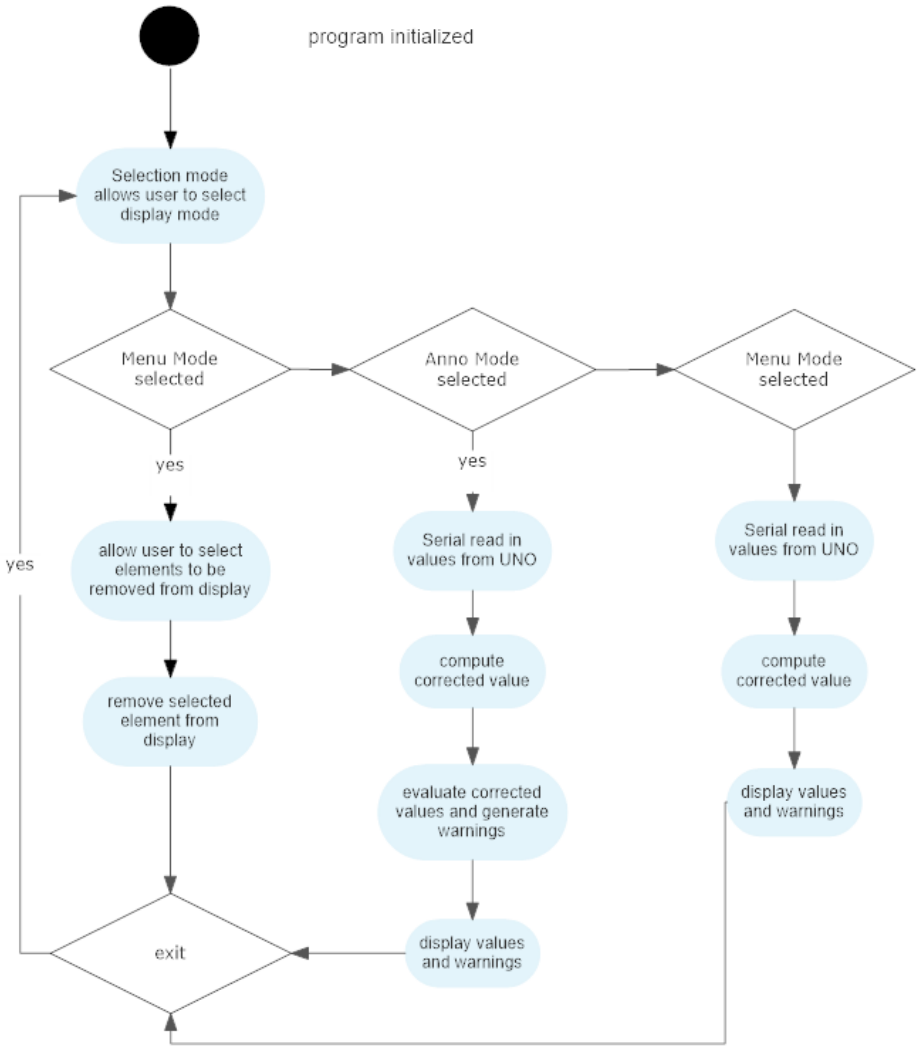

Figure 19: Activity Diagram for System Control

10.5.9 Dynamic Diagram for Measure()

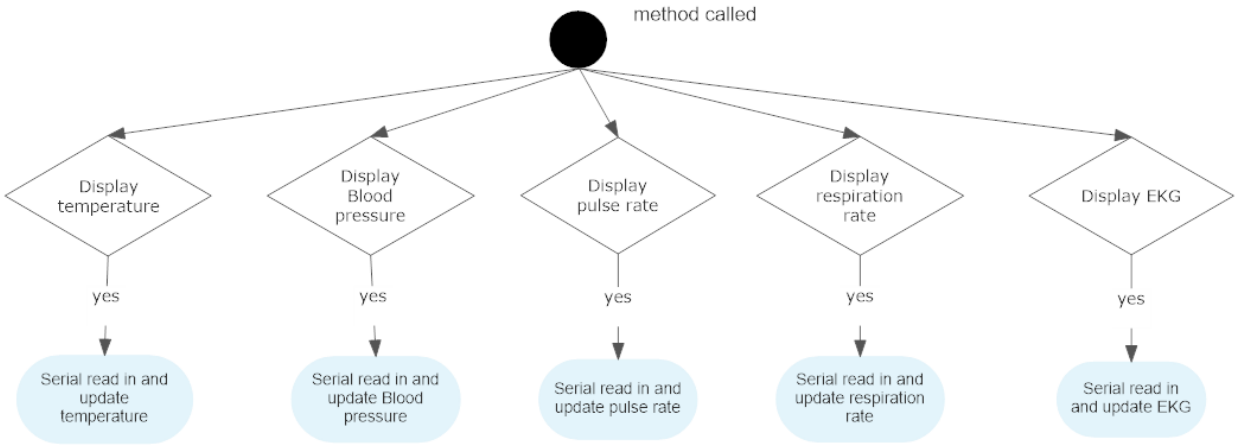

Figure 20: Measure

10.5.10 Dynamic Diagram for Compute()

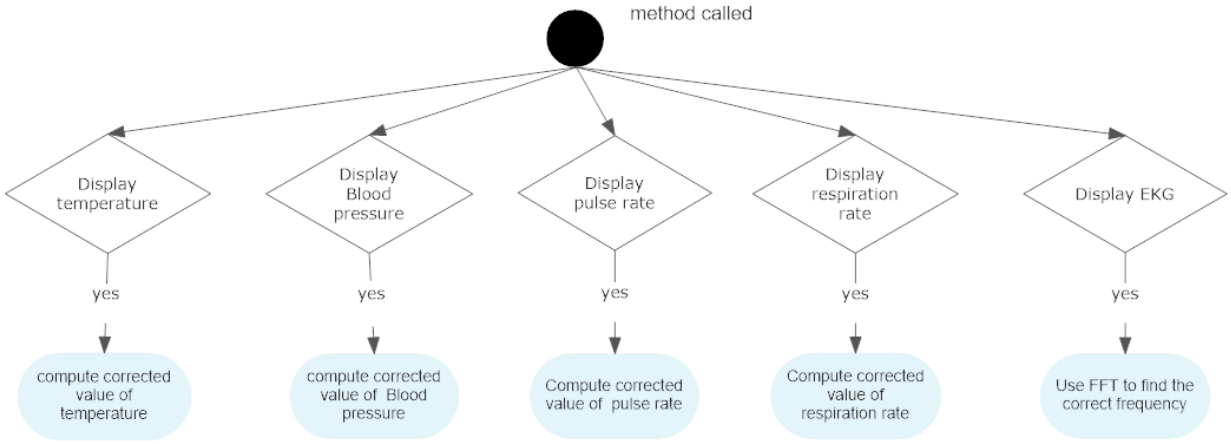

Figure 21: Compute

10.5.11 Dynamic Diagram for Display()

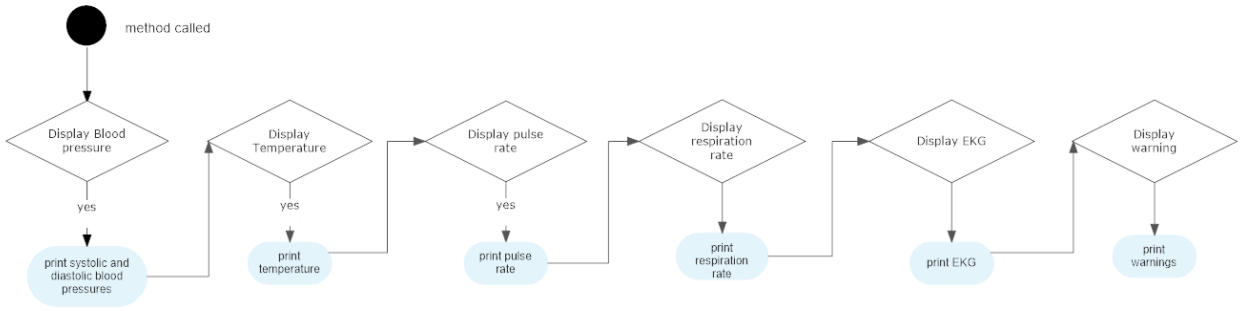

Figure 22: Display

10.5.12 Dynamic Diagram for Alarm()

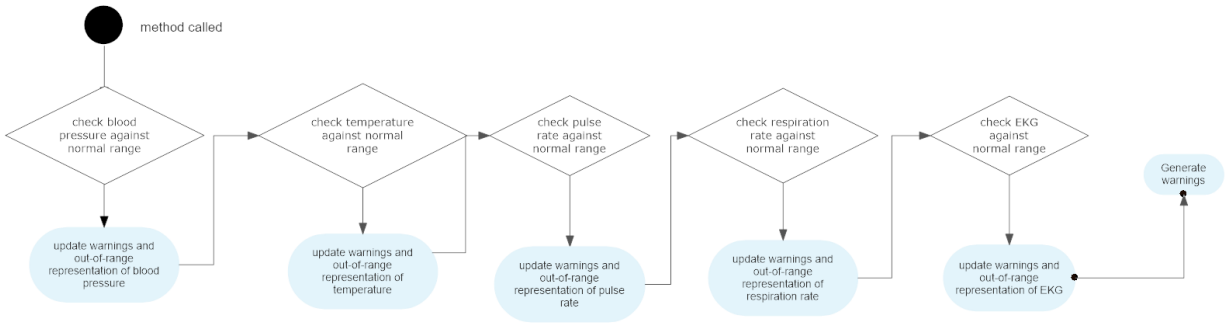

Figure 23: Alarm

10.5.13 Dynamic Diagram for Status()

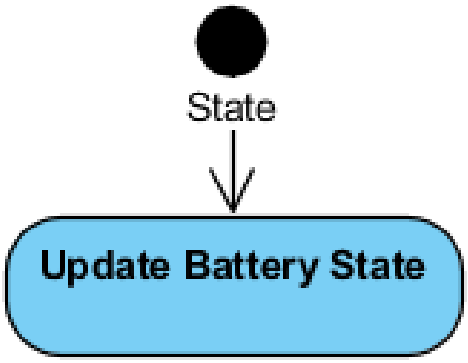

Figure 24: Alarm

10.5.14 Display Panel

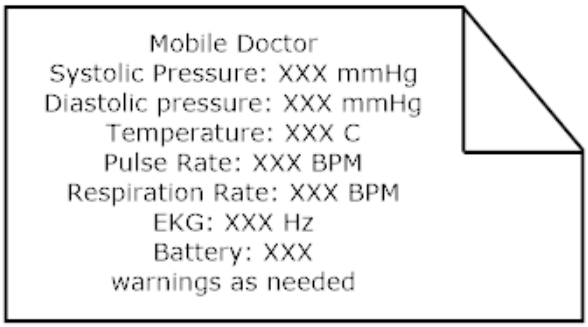

Figure 25: Display Panel

11 Time Division and Contributions

11.1 Time Division

Design → 4h
Coding → 30h
Test/Debug → 30h
Documentation → 10h

11.2 Contributions

Code

- LAN external communication - Kaiser
- Remote communication - Kaiser
- Implementation of internal serial communication- Kaiser
- Implementaion of TMT- Kaiser
- Improvements to Display (Screen Response time)- Xinyu
- Improvements to Display (Attractive User Interface)- Xinyu
- Task Queue Add/Delete Mechanism- Xinyu
- Implementation of Game- Xinyu
- Implementation of ECG- Xinyu and Kanika
- Implementation of Buffer- Kanika
- Implementation of Analog Temperature- Kanika
- Implementation of Digital Blood Pressure- Kanika
- Implementation of Respiration Rate- Kanika

Report

- Abstract- Kaiser

- Intro- Kaiser
- Design Specifications, Software Implementation Kaiser, Kanika, Xinyu
- Test Plan, Test Specs, Test Cases - Kaiser
- Result Analysis- Kanika
- Error Analysis- Kanika
- Summary, Conclusion - Kaiser
- Task Execution Time- Kanika
- Functional Diagrams- Kaiser, Xinyu
- Class Diagrams- Kaiser, Xinyu
- Activity Diagrams- Xinyu
- Pseudo Code- Kanika, Xinyu
- Citations- Kanika

Electronic Signature Kanika Aggarwal _____KA_____
Electronic Signature Kaiser Sun __KaiserTheGreat_
Electronic Signature Xinyu Gu _____XG_____

12 Citations

Aggarwal, K., Sun, K., & Gu, X. (n.d.). *Lab Report 1 EE 474*(Rep.).

Aggarwal, K., Sun, K., & Gu, X. (n.d.). *Lab Report 2 EE 474*(Rep.). Aggarwal, K., Sun, K., & Gu, X. (n.d.). *Lab Report 3 EE 474*(Rep.).

Arduino Forum - Index. (n.d.). Retrieved from https://forum.arduino.cc/

Serial. (n.d.). Retrieved from https://www.arduino.cc/reference/en/language/functions/communication/serial/

Arduino Forum - Index. (n.d.). Retrieved from https://forum.arduino.cc/

Serial. (n.d.). Retrieved from https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

Arduino Forum - Index. (n.d.). Retrieved from https://forum.arduino.cc/

Serial. (n.d.). Retrieved from https://www.arduino.cc/reference/en/language/functions/digital-io/digitalread/