EE 472 Lab 3
Learning the Development Environment - The Next Step

Jonathan Ellington
Patrick Ma
Jarrett Gaddy

Contents

List of Tables

List of Figures

10/10

1 ABSTRACT

Don't leave out old content. Still need to say which measurements are taken etc.

In this lab the students are to take on the role of an embedded system design team. They will design modifications to the medical instrument previously designed. When the device finds metrics are out of the acceptable range, the user is notified, thus saving them from potential health risks. The students first laid out the design for their system using various design tools, then they implemented the system in software. Finally the students tested their system and verified that it is ready to start saving lives.

10/10

2 INTRODUCTION

The students are to design an embedded system on the Texas Instruments Stellaris EKI-LM3S8962 and EE 472 embedded design testboard. The design must implement a medical monitoring device. This device must monitor a patient's temperature, heart rate, and blood pressure, as well as its own battery state. The design must indicate when a monitored value is outside of a specified range by flashing an LED on the test board. When a value deviates even further from the valid range an alarm will sound. This alarm will sound until the values return to the valid range or the user acknowledges the alarm with a button. The values of each measurement will also be printed to the OLED screen. This implementation will build upon the previous implementation of the device by adding functionality. Added functions include heart rate sensor, keypad input, menu display, data logging, and UART serial communication.

The design will be tested to verify proper behavior on alarm and warning notifications. In addition the implementation will be tested by measuring the amount of time that each of the 8 program tasks running the instrument take to execute. These tasks are mini programs that each handle a part of the instruments purpose.

3 DISCUSSION OF THE LAB

Side Effects? -5
For each task, need to provide full description, not just what has changed since last lab.

use case: 7/8
Func decomp: 5/8
Front Panel? 0/5
Block? 0/5
Description: 20/30

32/56

3.1 Design Specification

3.1.1 Specification Overview

The entire system must satisfy several lofty objectives. The final product must be portable, lightweight, and Internet-enabled. The system must also make measurements of vital bodily functions, perform simple computations, provide data logging functionality, and indicate when measured vitals exceed given ranges, or the user fails to comply with a prescribed logging regimen.

The initial Phase 1 functional requirements for the system are:

- Provide continuous sensor monitoring capability
- Produce a visual display of the sensor values
- Accept variety of input data types
- Provide visual indication of warning states
- Provide an audible indicator of alarm states

In addition, the following requirements have been added:

- Utilize a hardware-based time reference
- Support dynamic task creation and deletion
- Support a user input device
- Support data logging capabilities
- Support remote communication capability
- Improve overall system performance
- Improve overall system safety

3.1.2 Identified Use Cases

Taking the functional requirements listed above, several use cases were developed. A Use case diagram of these scenarios is given in Figure 1. Each use case is expanded and explained below.

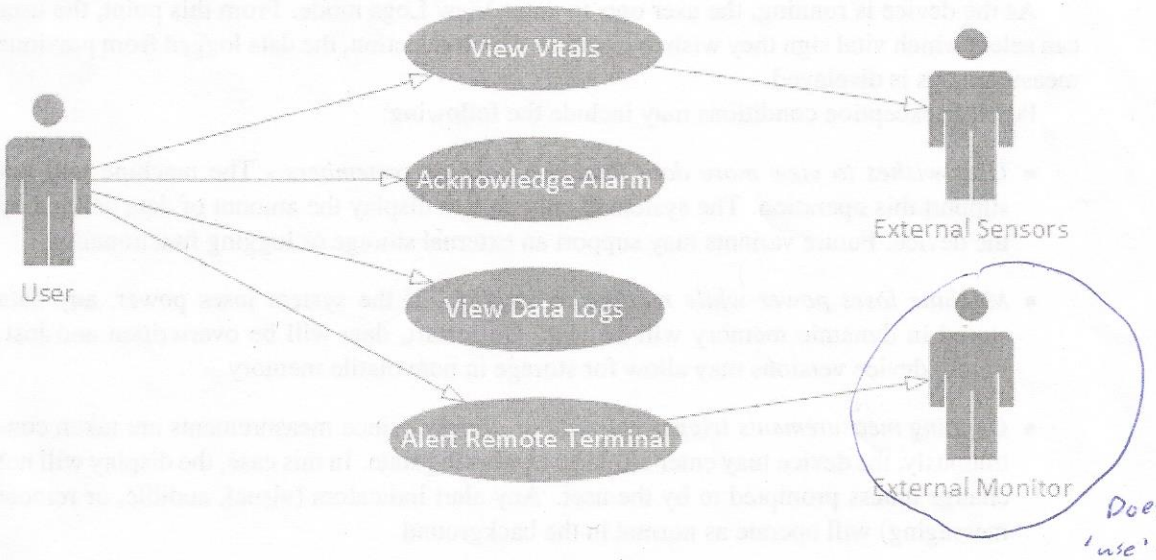


Figure 1: Use case diagram

Use Case #1: View Vital Measurements

In the first use case, the user views the basic measurements picked up by the sensors connected to the device.

During normal operation, once the device is turned on by the user, the system records the value output by each sensor. This raw value is linearized and converted into a human-readable form. The user can select toggle between a summary of current vitals as measured by the system or view measurements of each sensor individually.

Three exceptional conditions were identified for this use case:

- *One or more of the expected sensors is not connected* - If this occurs, the measurements taken by the device may be erratic. At the present moment, no action will be taken in such events. Later revisions may address the issue
- *A measured value is outside 5% of the specified normal range* - In this case, a warning signal will flash as an indication of the warning condition

- *A measured value falls outside 10% of a specified "normal" range* - In this case, an audible alarm will sound to indicate the alarm condition

Use Case #2: Acknowledge Alarm

In the second case, the system is in an alarm state. The user acknowledges the alarm condition by pressing a button.

Upon pressing the button, the system silences the audible alarm. Any visual warnings continue to flash during the silenced period. If significant amounts of time pass and the sensor reading(s) continue to maintain an alarmed state, the audible alarm will recommence.

No exceptional conditions were identified for this use case.

Need exceptions for every case -1

Use Case #3: View Measurement Logs

In the third use case, the user wishes to view previously recorded measurements for a given vital sign.

As the device is running, the user opts to enter View Logs mode. From this point, the user can select which vital sign they wish to examine. Upon selection, the data logged from previous measurements is displayed.

Possible exception conditions may include the following:

- *User wishes to view more data than the machine remembers* - The machine will not support this operation. The system is only able to display the amount of data defined by the device. Future variants may support an external storage or logging functionality

- *Machine loses power while reading or writing* - if the system loses power, any data stored in dynamic memory will be lost. On restart, data will be overwritten and lost. Future device versions may allow for storage in nonvolatile memory

- *Ongoing measurements trigger warning or alarms* - since measurements are taken continuously, the device may enter an alarm or warning state. In this case, the display will not change unless prompted to by the user. Any alert indicators (visual, audible, or remote messaging) will operate as normal in the background

Use Case #4: System Alert to Remote Terminal

In the event that the system enters an alert state (e.g. alarm or warning), the system can send a message to a remote terminal connected to the device. The messages sent can inform a second actor of the cause of alert and provide any additional useful information.

Possible exception conditions may include:

- *Improper configuration of the Remote Terminal* - If the remote terminal connection is improperly configured or initialized, data received may be corrupted and not display properly. The device cannot ensure a proper connection and it is the responsibility of the remote user to ensure the correct configuration is used

- *Remote Connection Lost* – If the user terminates the connection or the connection is lost, messages sent by the device may not arrive at the remote terminal or data may be corrupted. The device will not necessarily monitor the status of any remote connection; this responsibility is the remote terminals. In the event a connection is broken, the device system must continue to perform the other system functions without ill effects.

3.1.3 *Detailed Specifications (update)*

For this project, the requirements have been further specified as follows:

The system must have the following inputs:

- Alarm acknowledgment capability using a pushbutton
- Buttons or switches to allow user to access system modes and menu items
- Sensor measurement input capability consisting of:
  - Body temperature measurement
  - Pulse rate measurement signal
  - Systolic blood pressure measurement
  - Diastolic blood pressure measurement

The system must have the following outputs:

- Visual display of the following data in human-readable formats:
  - Body temperature
  - Pulse rate
  - Systolic blood pressure
  - Diastolic blood pressure
  - Battery status
- Visually indicate warning state with a flashing LED
- Visually indicate a low battery state with an LED
- Audibly indicate an alarm state using a speaker
- External data connection to a remote terminal

The initialization values, normal measurement ranges, displayed units, and warning and alarm behaviors for each vital measurement are given in Table 1. The sensors must be sampled every five seconds and the system cannot block and cease operation for five seconds.

Measurement	Units	Initial Value	Min. Value	Max. Value	Warning Flash Period
Body Temperature	C	75	36.1C	37.8C	1 sec
Systolic BP	mm Hg	80	-	120 mmHg	0.5 sec
Diastolic BP	mm Hg	80	-	80mmHg	0.5 sec
Pulse Rate	BPM	50	60 BPM	100 BPM	2 sec
Remaining Battery	%	200	40 %	-	Constant

Table 1: Specifications for measurement data

A measurement enters a warning state when its value falls outside the stated normal range by 5%.

requirement modification: An alarm state occurs when the systolic blood pressure falls outside its stated normal range by 20%.

Additionally, the system must be implemented using the Stellaris EKI-LM3S8962 ARM Cortex-M3 microcomputer board, The software for the system must be written in C using the IAR Systems Embedded Workbench/Assembler IDE.

3.1.4 *Detailed Task Specifications (updates)*

- New task: KeypadTask
  - The keypad task will scan the keypad and decode any keypresses
  - The task will have support the following user inputs:
  - Mode selection between 2 modes (1 button)
  - Menu selection between 3 options (1 button)
  - Alarm acknowledgement (1 button)
  - Up and down scroll functionality (1-2 buttons)
  - A new set of global variable will be created to store the state of the keypad and key presses

What are these modes/options? -3

- New Task: Initialize (Startup):
  - Changes to Warn/AlarmTask:
  - The warnings will be activated and indicated as before in project 1
  - The alarm state is changed to activate only when the systolic pressure is 20% above the normal range.
  - The alarm will sound in 1 second tones (1 second on, 1 second off)
  - When an alarm or warning state occurs, the serial communication task will be added to the task queue
  - The deactivation period of the alarm sound is defined as 5 measurement periods

None of this has to do w/ startup task -2

- New task: Serial Communication:
  - The task is enabled by the warn/alarm task
  - When run, the task will open an RS-232 connection at 115,200 baud, no flow control, no parity, and 1 stop bit
  - The present corrected measurement will be displayed on the terminal in the same fashion as the display task annunciation mode
  - After sending data to the terminal, the serial communication task will remove itself from the task queue
- Changes to the MeasureTask:
  - Once a complete set of measurements has been taken, the compute task is added to the task queue
  - Pointers to the variables used in the measure task will be relocated to accommodate the new data architecture
  - The pulse measurement will monitor and count the frequency of a pulse rate event interrupt
  - A new value will be stored to memory if the present reading is greater than $\pm 15\%$ of the previous measurement

How do the measurements change?

  - The measurement limits will correspond to 200bpm and 10bpm, determined empirically.

- Changes to ComputeTask: What calculations are performed?
  - All measurements will be recomputed
  - After computing the corrected values for all measurements, the ComputeTask will remove itself from the task queue

- Changes to DisplayTask:
  - Display will now support multiple display options
  - Menu mode will allow selection of each of the individual measurements. Upon selection of a measurement, the current value of the measurement will be displayed onscreen
  - Annunciation mode will display the current status of each measurement as in project 1, and provide the same functionality as the display in project 1.

- Changes to Warn/AlarmTask:
  - The warnings will be activated and indicated as before in project 1
  - The alarm state is changed to activate only when the systolic pressure is 20% above the normal range.
  - The alarm will sound in 1 second tones (1 second on, 1 second off)
  - When an alarm or warning state occurs, the serial communication task will be added to the task queue
  - The deactivation period of the alarm sound is defined as 5 measurement periods

- Changes to Schedule:
  - System will maintain a list of activated and deactivated tasks. The list must be updatable during runtime based on the system state
  - The hardware timer will provide a system interrupt every 250ms or equal to the minor cycle, whichever is shorter
  - At runtime, upon a timer interrupt event, all tasks will be added or removed according to the task activation list. The tasks will then be run
  - Task Control Blocks will have forward and backward pointers to allow references to the next task
  - The scheduler cannot block for five seconds
  - The scheduler will toggle a GPIO pin at least once during execution of the task queue

- Changes to StatusTask:
  - There are no changes to the status task But what does it do?

Activity: 6/8
Class/Task: 7/8
Control/Data Flow: 2/8
State Diagrams?: 0/8

Description: 20/40

Some parts were well detailed but others needed more elaboration or were missing major parts.

35/72

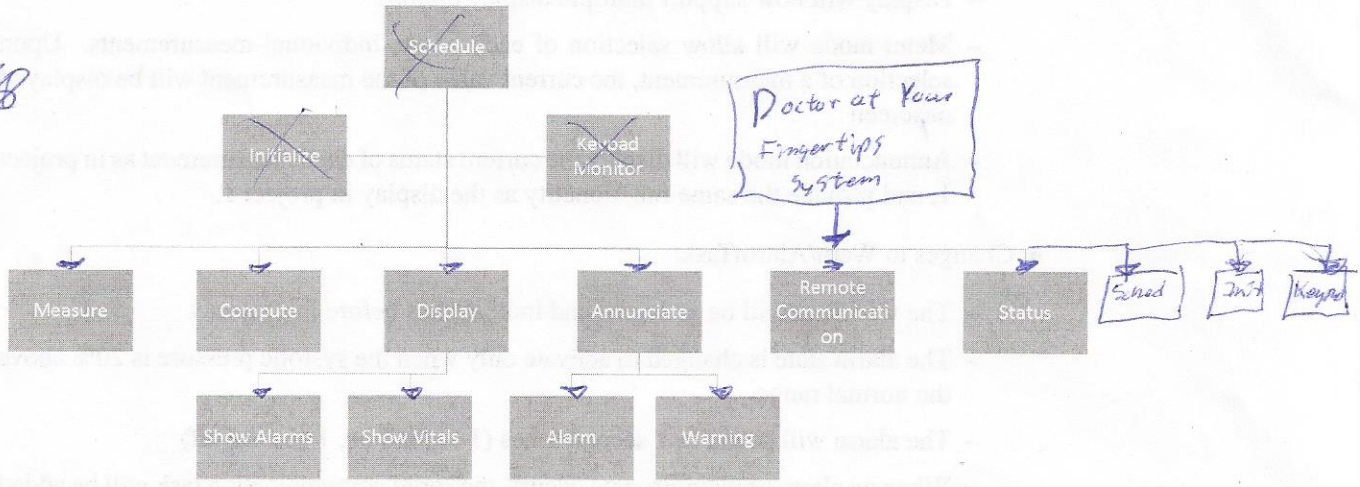
3.2 Software Implementation

A top-down design approach was used to develop the system. First, a functional decomposition of the problem was carried out based on the identified use cases. Next, the system architecture was developed. After understanding the system architecture, the high-level project file structure in C was defined, followed by the low-level implementation of the tasks.

3.2.1 Functional Decomposition

After understanding how the user would interact with the device, the high level functional blocks were developed. These blocks are shown in Figure 2.

5/8



You should be able to break down this task further.

Figure 2: Functional Decomposition

3.2.2 System Architecture

Next, the system architecture was developed (Figure 3). At a high level the system works on two main concepts, the scheduler and tasks. Tasks embody some sort of work being done, and the scheduler is in charge of determining the speed and order in which the tasks execute. The system has several tasks, each with their own specific job. For modularity reasons, each task should have the same public interface and the scheduler should be able to run each task regardless of that specific tasks job or implementation. Thus the task concept is abstracted into a Task Control Block (TCB), and the scheduler maintains a queue of TCBs to run. The TCB abstraction is shown in Figure 3 using inheritance, and the fact that the scheduler has a queue of TCBs is shown with composition. The core functionality of the system was divided into the following eight main tasks:

Initialization Task This task Initializes data structures and does system startup-related jobs. This task is not actually scheduled to run, it only executes a single time at system startup.

Measure Task The measurement task is in charge of interacting with the blood pressure, temperature, and pulse sensors. Each of these is simulated. The blood pressure and temperature are simulated in the CPU. The task will measure the pulse rate by parsing an externally generated square wave of varying frequency; the pulse rate being proportional to the frequency.

Compute Task Compute takes the simulated raw data and converts it to the correct units of measurement. Raw temperature data to Celsius, blood pressure to mm Hg, and pulse rate to BPM.

Keypad Monitor Keypad will check the keypad for user input. It should provide the user with four keys: two for scrolling, one for selection, and one to go back.

Display Task The display task will show a user interface on the Stellaris OLED. The user will interact with the display using a keypad. Under normal operation, a menu will be displayed asking users which measurement they would like to see. If the user presses back while in this menu, they enter annunciate mode which displays all the measurements currently in warning or alarm state.

Warning/Alarm Task Under normal operation, this task will light a green LED signifying that everything is OK. If one of the measurements enters a warning state, the task will flash a red LED at a rate specific to the warning for that measurement. If there is an alarm state, it sounds the alarm by driving the speaker. At any time if the battery goes too low, the yellow LED will illuminate.

Remote Communication Task This task is in charge of sending data to a remote terminal. If any of the states are in a warning or alarm condition, this task will transmit the (corrected) data to the remote terminal.

Status Task Status receives information about the battery on the system and updates it's current data accordingly.

Each of these tasks interact using the shared data shown in Figure 3.

3.2.3 High-level Implementation in C

After developing the system architecture, the design needed to be translated into the C programming language. The design manifested in a multi-file program consisting of the following source files:

- **globals.c/globals.h** - Used to define the Shared Data used among the tasks
- **schedule.c/schedule.h** - Defines the scheduler interface and it's implementation, as well as the TCB structure
- **timebase.h** - Defines the timebase used for the scheduler and tasks
- **task.h** - Defines the TCB interface for a task

Each task also has it's corresponding ".c" and ".h" file (for example, measure.c and measure.h).

The TCB structure that the scheduler uses must work for all tasks, and must not contain any task-specific information. Instead, the TCB consists of only a void pointer to the tasks data, and a pointer to a function that returns void and takes a void pointer, as shown in Listing 1.

7/8

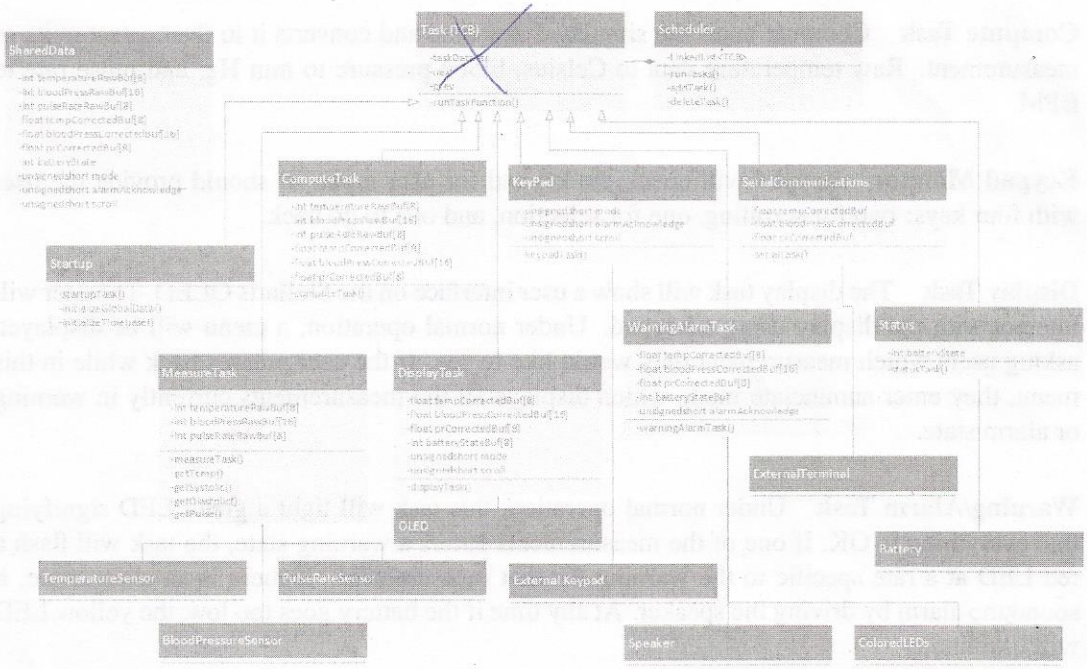


Figure 3: System Architecture Diagram

Don't put code in software Imp.

```
1 struct TCB {
2     void *taskDataPtr;
3     void (*taskRunFn)(void *);
4 }
```

Listing 1: TCB Construct

Leaving out the type information allows the scheduler to pass the task's data (*taskDataPtr) into the task's run function completely unaware of the kind of data the task uses or how the task works.

For increased modularity, the data structure used by each task was not put in the task's header file. Instead, the structure was declared within the task implementation file, and instantiated using a task initialization function. In the header file, a void pointer pointing to the initialized structure is exposed with global scope, as well as the task's run and initialization functions.

Need to update, TCB also has next and prev fields.

3.2.4 Task Implementation

The primary task of this project is to implement C code for a medical device on the Stellaris EKI-LM3S8962 and its ARM Coretex A3 processor. The project was started by creating a main file that initializes the variables used in each task and starts the hardware timer then runs into an infinite while loop. Inside the while loop a run method is called. The run method is part of the scheduler. The run method has a runTask flag which determines whether or not anything should actually be run this call. The flag is set to true by the hardware timer's interrupt handler. Once the runTask flag is true the run method will keep track of whether the device is on a minor cycle or a major cycle and run the preform task method of each task. The runTask flag is then set to false so that the tasks will not be executed again until the hardware interrupt has again

How does it know whether it's a major or minor cycle?

the compiled assembly code, the pointer is never overwritten, yet the pointer memory address appears to be consistently set to a null value.

The ease of change in the code is the result of a large amount of time spent on design. The design makes it easy to configure flash times, add new tasks, and to reason about tasks independently of the whole system. The solid high-level architectural design led to ease of implementation and change.

In terms of performance, the run times of each task appear to correspond with the number of instructions required for each task. Given the speed of the CPU, 8 MHz, we can calculate an estimated number of instructions for each task. This is given in Table 3. The majority

Task	Instructions
Measure	187
Compute	443
Display	12240
Warning	219
Status	45
KeyPad	374
Serial	5917

Table 3: Estimated instructions per task, rounded to the nearest instruction

of the cycles are likely spent waiting for memory. For example, the status task only has two comparisons and an arithmetic operation, but has to reference the data in global memory. The exception here is the display task, which was about three orders of magnitude more instructions than the other tasks. This was due to the sprintf() library call, included in the standard C library. While this could have been optimized, it was found that with a minor cycle delay of 250 ms, the display delay of 22.9 ms was not significant.

20/20

4.3 Analysis of Any Errors

There were two errors found in the final project. First, the pulse rate range is not exactly as specified. Second, it was found that the serial communications task did not produce the correct values.

The specified corrected pulse rate was to be between 10 BPM and 200 BPM. For simplicity, the raw pulse rate was implemented as a 1-1 mapping from frequency to raw pulse rate. For example, a 1 Hz frequency would produce a value of 1 for raw pulse rate, and a 15 Hz signal would produce a value of 15 for raw pulse rate. This caused two issues. When the Input frequency is 1, the measured BPM (using the specified raw to correct conversion) is $8+3\cdot(1)=11BPM$. This is larger than the specified 10 BPM minimum. Also, when the input frequency is 64 Hz, a corrected value of $8+3\cdot(64)=200BPM$ is expected. However, as the frequency increased, the overhead of the other running tasks became significant. As a result, more rising edges could fit in our measurement interval than we expected. This resulted in a maximum BPM of roughly 206 BPM.

There was also an error in the Serial communication protocol in which the usnprintf command used to format the data was causing a runtime error when printing formatted data to a buffer. Rather than printing the actual corrected values to the buffer (to be sent to the remote terminal), it seemed the command was printing addresses. It was believed this was caused by an error with the CircularBuffer implementation, though no other tasks had issues with it (and extensive tests validated the CircularBuffer).

4.4 Analysis of Implementation Issues and Workarounds

The medical instrument design in this project was completed and tested successfully to meet almost all the requirements, the designers did face a few difficulties in designing the device, however, because this design was additional functionality added to a previous projects design, many of the errors previously encountered were easily avoided due to experience of the students, and already completed coding work.

Many of the challenges the designers of this project faced were in the keypad input and the data output. The keypad input posed a difficult hardware challenge as there are 16 input keys on the keypad but only 8 connections for the microprocessor to connect to the keypad. This means that to identify a single key press 4 connections must be set as outputs and 4 as inputs. The inputs can then be scanned as the outputs are set 1 at a time to find which key is pressed. Instead of implementing this design, the students instead opted to use only 4 buttons on the keypad. This allowed the strobe design to be ignored. Instead 1 row of keys was activated all the time and that row was scanned for button presses.

In addition to keypad input, there was also difficulty in implementing data formatting functions. After adding a hardware delay, IAR workbench no longer allows the use of sprintf which had previously been used to print and format data. The usnprintf command was instead used to format and print data to a buffer, however, the students found that usnprintf does not have the ability to print floating point data. This issue was resolved by changing measurements that were previously printed as floats to be printed as integers. usnprintf also caused issues when printing certain data for the serial task. In this case the usnprintf was causing a runtime error and freezing the operation of our device. This issue remained unresolved.

All problems but one were solved before demonstrating the product to the interested parties. The final project still contained the serial error previously mentioned.

5 TEST PLAN

To ensure that this project meets the specifications listed in section 3.1, the following parts of the system must be tested:

- Vitals are measured and updated
- System properly displays corrected measurements and units properly
- System enters, indicates, and exits the proper warning state for blood pressure, temperature, pulse, and battery
- System enters and exits the alarm state correctly
- Alarm is silenced upon button push
- Alarm recommences sound after silencing if system remains in alarm state longer than silence period

Additional tests to determine the runtime of each specific task are also required.

The inclusion of additional specifications for Phase II of the project requires additional tests to ensure the system meets the customer requirements.

Phase II Tests:

- Scheduler loops through linked list properly

- Scheduler adds and removes from the linked list the following tasks correctly:
  - Compute task added by measure task
  - Serial communication task added by annunciation task
  - Compute task removed by itself
  - Serial communication task removed by itself
- Warning task alarm meets the following two requirements
  1. Has one (1) second tones; a total period of 2 seconds
  2. Activates only when systolic pressure is greater than 20% above normal
  3. Has an auditory deactivation or "sleep" period of 5 measurement cycles
- Serial task displays the temperature, blood pressure, pulse rate, and battery status as listed in Section 3.1
- Keypad task captures user inputs, sets the appropriate inputs, and causes the associated changes in system state
- Hardware timer updates the system's minor cycle counter

More detailed explanation of the tests performed is provided in the following sections.

5.1 Test Specification

5.1.1 Scheduler

The scheduler needs to be shown to correctly schedule and dispatch tasks. This means that tasks should execute in the right order, and at the right time. Given a minor cycle of 50 ms, every task should run roughly once every 50 ms. Also, the scheduler needs to successfully add and remove tasks from the queue dynamically. Specifically, the Measure Task should be able to tell the scheduler to add the Compute Task and the Warning/Alarm Task should be able to schedule the Serial Task. Both Compute and Serial should be able to be removed from the schedule.

5.1.2 Measure Task

For this design, the temperature and blood pressure values were simulated on the CPU. The pulse rate was simulated using an externally generated square wave of varying frequency.

- **Temperature** The temperature should increase by two every even major cycle (5 seconds) and decrease by one ever odd major cycle until it exceeds 50, at which point the process should reverse (decrease by two every even major cycle and increase by one every odd major cycle), until it dips below 15, and the whole process should be started over again.
- **Pulse** The pulse rate should match one-to-one with the frequency of the input signal. For example, a 15 Hz signal should produce a raw pulse rate of 15.
- **Systolic Pressure** The systolic pressure should increase by three every even major cycle and decreases by one every odd major cycle. If it exceeds 100, it should reset to an initial value.

- **Diastolic Pressure** The diastolic pressure should decrease by two on even major cycles and decrease by one on odd major cycles, until it drops below 40, when it should restart the process.

The Measurement Task should also successfully add the Compute Task to the schedule queue.

5.1.3 Compute Task

The compute task should be verified to convert raw simulated sensor data according to the following formulas.

- $CorrectedTemperature = 5 + 0.75 * RawTemperature$
- $CorrectedSystolicPressure = 9 + 2 * RawSystolicPressure$
- $CorrectedDiastolicPressure = 6 + 1.5 * RawTemperature$
- $CorrectedPulseRate = 8 + 3 * RawTemperature$

The compute task should also successfully remove itself from the schedule queue.

5.1.4 Keypad Task

The keypad should be tested to successfully capture user input. When the select button is pressed, the measurement selection value should reflect the selected measurement. When the up scroll button is pressed, the scroll value should be incremented, and when the down scroll button is pressed, the scroll value should be decremented. If the alarm acknowledge button is pressed, this should be reflected in the alarmAcknowledge global value.

5.1.5 Display Task

On load, the display task should present the user with an option to select the desired measurement. If the back button is pressed, the annunciation screen should be displayed, showing the measurements in warning or alarm state.

5.1.6 Warning/Alarm Task

The warning/alarm system needs to be tested to do several things. When in a warning state, it should flash the red LED at the rate appropriate for the warning. When the battery is low, it should illuminate the yellow LED. If the system is in an alarm state, it should sound the speaker alarm. The following ranges in Table 4 are calculated from the specified minimum and maximums found in Table 1 on page 4.

This task should also add the Serial task if any of the measurements are in a warning or alarm condition.

5.1.7 Serial Task

If any of the measurements are in warning or alarm state, this task should send this data serially to a remote terminal. The task should send all the data (not just the data in warning or alarm state). It should be printed as shown in Listing 2.

Data	Warning Range	Alarm Range
Temperature	34.3 - 39.7 C	32.5 - 41.6 C
Systolic Pressure	> 84 mmHg	> 88 mmHg
Diastolic Pressure	> 126 mmHg	> 132 mmHg
Pulse	57 - 63 BPM	54 - 110 BPM

Table 4: Initial values and warning/alarm states

```
1. Temperature                  0 C
2. Systolic Blood Pressure      0 mm Hg
3. Diastolic Blood Pressure     0 mm Hg
4. Pulse Rate                   0 BPM
5. Battery                      0 %
```

Listing 2: Remote Terminal Output

5.1.8 Status Task

Since the initial design does not use a battery, the status task simulates the battery state using the CPU. For now, it simply decrements the state of the battery. The test should show that the battery state is decremented by one every major cycle.

5.2 Test Cases

The students begin testing by examining if the alarm sounds at the proper time. This is initially tested by disabling the functions that simulate measurements being made on each of the data measurements, and setting their initial values to be either within the alarm range or outside of the alarm range. The warning states were also initially tested this way. The initial values for raw data given in Table 1 on page 4 were used to test the normal state of the machine because each falls within the acceptable range of measurements for corrected data (also given in Table 1) that does not require a warning. Using these initial values, the code was programmed onto the Stellaris board. Correct operation was verified by the alarm not sounding, and the red led being off, indicating that no warning state was in effect. In addition the green led was on indicating a normal state. Next the students varied one parameter at a time to be outside of the acceptable range by more than 10%. Starting with the temperature being set to an initial raw value of 50, the alarm was verified by hearing the aural annunciation coming from the system. In addition, the temperature warning stat was also in effect. This means that the green led was off and the red led was blinking. To verify correct operation we needed to make sure the led was blinking with a period of 1 second. The correct flashing pattern was verified by counting the number of times the led flashed in 6 seconds. In this case, for temperature, the led flashed 6 times in 6 seconds indicating a 1 second period, and correct operation. After this test, the temperature value was returned to 42 and the Pulse was instead set to 45. The same methods were used to verify that the alarm and warning states for pulse rate were working correctly, but this time the warning led turned on 3 times in 6 seconds indicating a 2 second period which is the intended period of flashing. The pulse rate was then returned to 25 and each pressure reading was checked for correct operation individually by being set to an initial raw value of 100. Once again, the green led started off because the system was not in a normal state. The alarm was sounding due to

the extremely high blood pressure measurements, and the red warning led flashed 12 times in 6 seconds indicating the correct period of .5 seconds for a blood pressure warning. In addition to testing the validity of each warning state and alarm state, the acknowledgment of the alarm was also tested during each of these tests. This was tested by hitting the acknowledge button once during each measurements test. During each test, hitting the acknowledge button turned the alarm sound off for a short time, as intended.

Next the measurement simulation functions were tested. This was done by re-enabling each one that had been disabled from the previous test one at a time. The initial raw values were again set to the values in Figure 5. When each measurement was re-enabled, the students could watch the temperature change at each major cycle using the OLED display. Since the OLED display indicated that the corrected temperature went up .75 degrees on a major cycle then down 1.5 degrees on the next, the temperature measurement was working as intended. This situation also gave the students an opportunity to verify that the warning and alarm states initiated as the temperature fell out of the acceptable range. The Led began flashing with a 1 second period after a few major cycles, then the alarm began sounding, indicating correct operation. Since temperature was working correctly, the temperature measurement function was once again disabled and each blood pressure measurement was re-enabled individually for testing. The Systolic pressure began by rising 4 mm Hg on a major cycle then falling 2 mm Hg on the next, and the Diastolic pressure by rising 3 on a Major cycle and falling 1.5 on the next, this was consistent with the design. The warning and alarm states were activated as each passed its threshold and the red led was blinking with a period of .5 seconds. The warning led was also tested in the case that all warning states were active. To do this all initial values were set to 100. In this case, as designed by the students, the red warning led indicated the fastest blinking warning with a .5 second period.

In this device, a new pulse rate monitor device was added. To test the pulse rate monitor the monitor input was connected to a function generator generating square waves. As the square wave frequency was increased the pulse rate value was expected to increase. This was verified to be working correctly. As the pulse rate passed through the warning zones the design for pulse rate warnings was also verified to be working correctly.

Additionally, the improved medical device now has a menu that is displayed on the OLED display and navigated using the testbench keypad. The design operating these functions was tested by navigating to each part of the menu using the keys on the keypad and visually verifying that each part of the menu displayed the correct data. Each menu, annunciation, main menu, and each measurements selection mode, was visually verified to be working as intended. The keypad functionality was verified as each button was used to navigate through the menu.

The newly added serial communication was then tested using the hyperterm program on the lab station PC. The serial connection was established over a virtual COM port on the USB connection from the PC to the Stellaris board. The program had the intended functionality of displaying each of the measurements, and its current data on the serial port whenever the alarm state was entered. The functionality could be verified by watching the hyperterm screen to see if the data displayed when a warning state occurred. The data on hyperterm could then be compared to the OLED display data to verify its accuracy.

The final bit of testing preformed on the system was timing each task within the system. This was done by adding a general purpose output pin in the scheduler code. This output was set high right before the execution of a task, and set low immediately after the execution of the task. An oscilloscope was then attached to this output pin and set to trigger on a positive edge. The cursors were then used to measure the amount of time the signal was high in each cycle.

6 SUMMARY AND CONCLUSION

6.1 Final Summary

A medical monitoring system with a user interface, alarm notification, and remote terminal display was designed, implemented, and tested. The design simulated temperature and blood pressure, and obtained pulse rate data from an external function generator. These results were converted to a human readable form, and tested to see if there was a warning or alarm state. In the event of warning or alarm, the user was notified visually with LEDs and aurally with an alarm sound. The warning and alarm data was transferred to a remote terminal.

Some implementation errors were encountered. The pulse rate range could not fit the specification exactly, and the serial communications task was not printing the correct values to the remote terminal. Aside from these two implementation errors, the device worked as specified.

6.2 Project Conclusions

This project contained 3 major phases, the design, implementation, and testing steps. The students were immediately introduced to using the unified modeling language(UML) to design embedded systems. This is the first time many students will have used UML for system design which caused some confusion and difficulty. In the end through the use of the UML guidelines for design, the students were able to implement their system in code for the Texas Instruments Stellaris EKI-LM3S8962 much more quickly and with far fewer errors than if they had spent less time in the design phase of this project.

Effective design tools allowed the students to quickly implement their embedded system in C code for an ARM Cortex A3 processor, and move onto the testing phase of the project quickly. Unfortunately, while testing the students encountered a number of problems in using the PWM and general purpose input and output signals. After consulting the documentation for the Stellaris kit and solving their input/output problems, they began testing their design using visual and audio queues, the IAR embedded workbench debugger, and a few specifically programmed debug features. After the results of the testing verified the design to be working correctly, the students proceeded to present their medical instrument to their instructor.

A BREAKDOWN OF LAB PERSON-HOURS (ESTIMATED)

Person	Design Hrs	Code Hrs	Test/Debug Hrs	Documentation Hrs
Patrick	20	10	10	12+
Jonathan	15	5	4	8

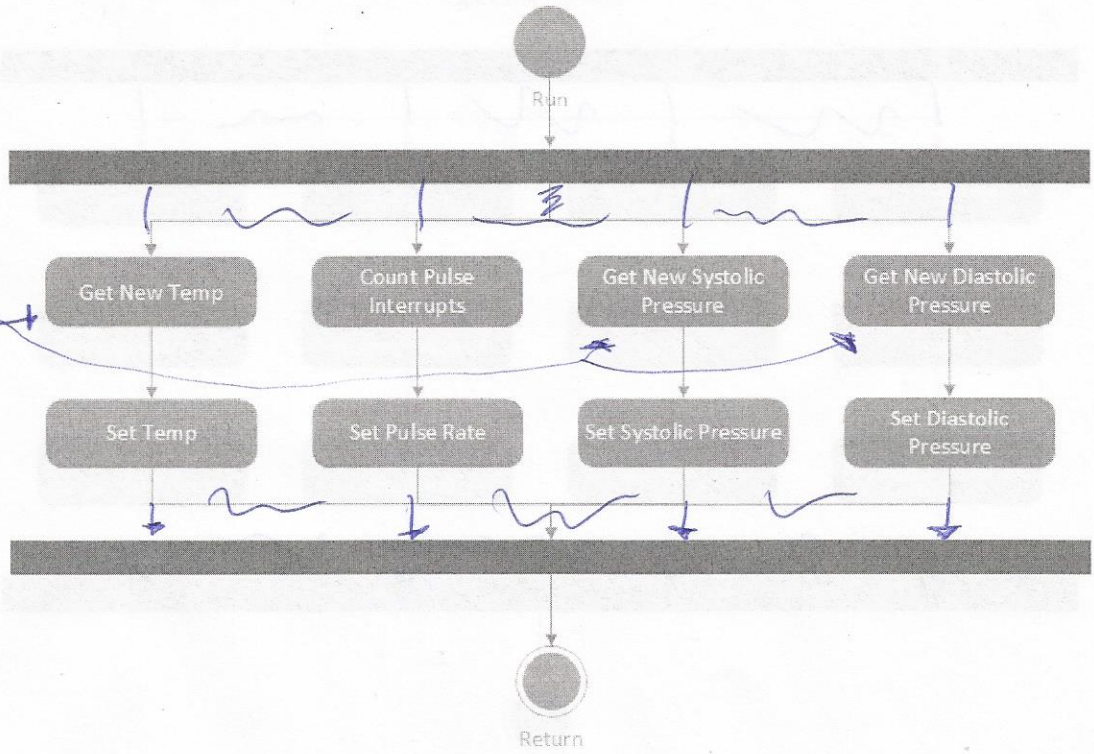
By initializing/signing above, I attest that I did in fact work the estimated number of hours stated. I also attest, under penalty of shame, that the work produced during the lab and contained herein is actually my own (as far as I know to be true). If special considerations or dispensations are due others or myself, I have indicated them below.

Signatures? -3
Missing entry for Jarrett -1

22/26

B ACTIVITY DIAGRAMS

6/8



How is this done? Incorporate logic into diagram. -1

Figure 5: Measure Activity Diagram

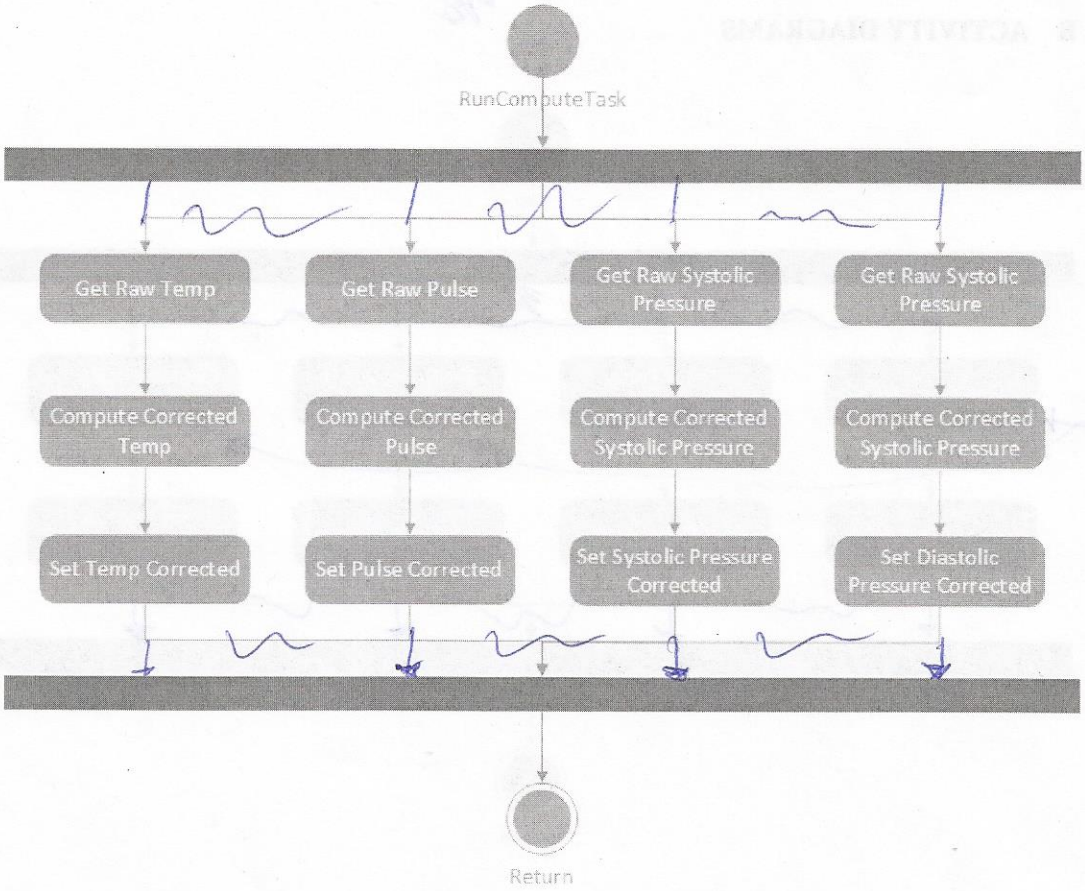


Figure 6: Compute Activity Diagram

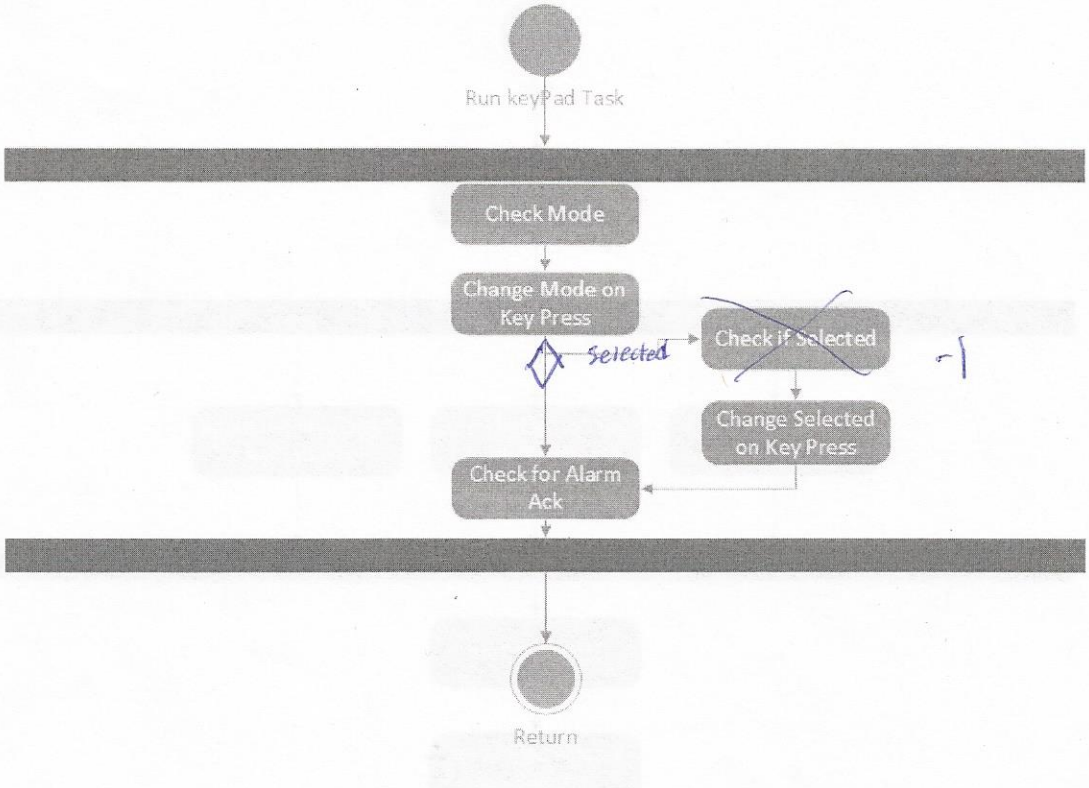


Figure 7: Keypad Activity Diagram

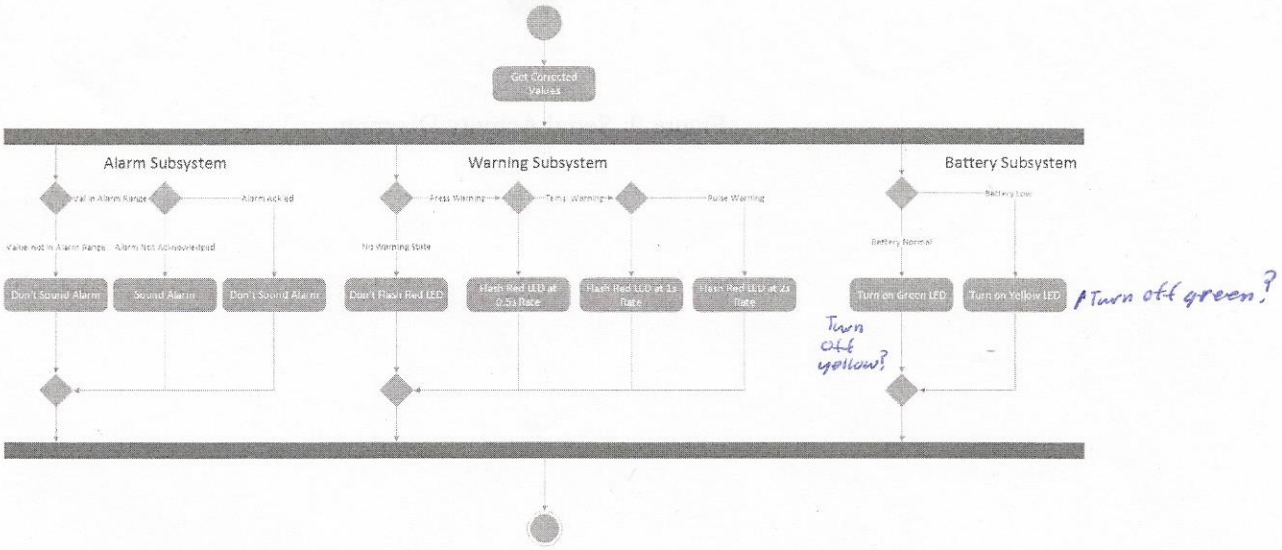


Figure 8: Warning Activity Diagram

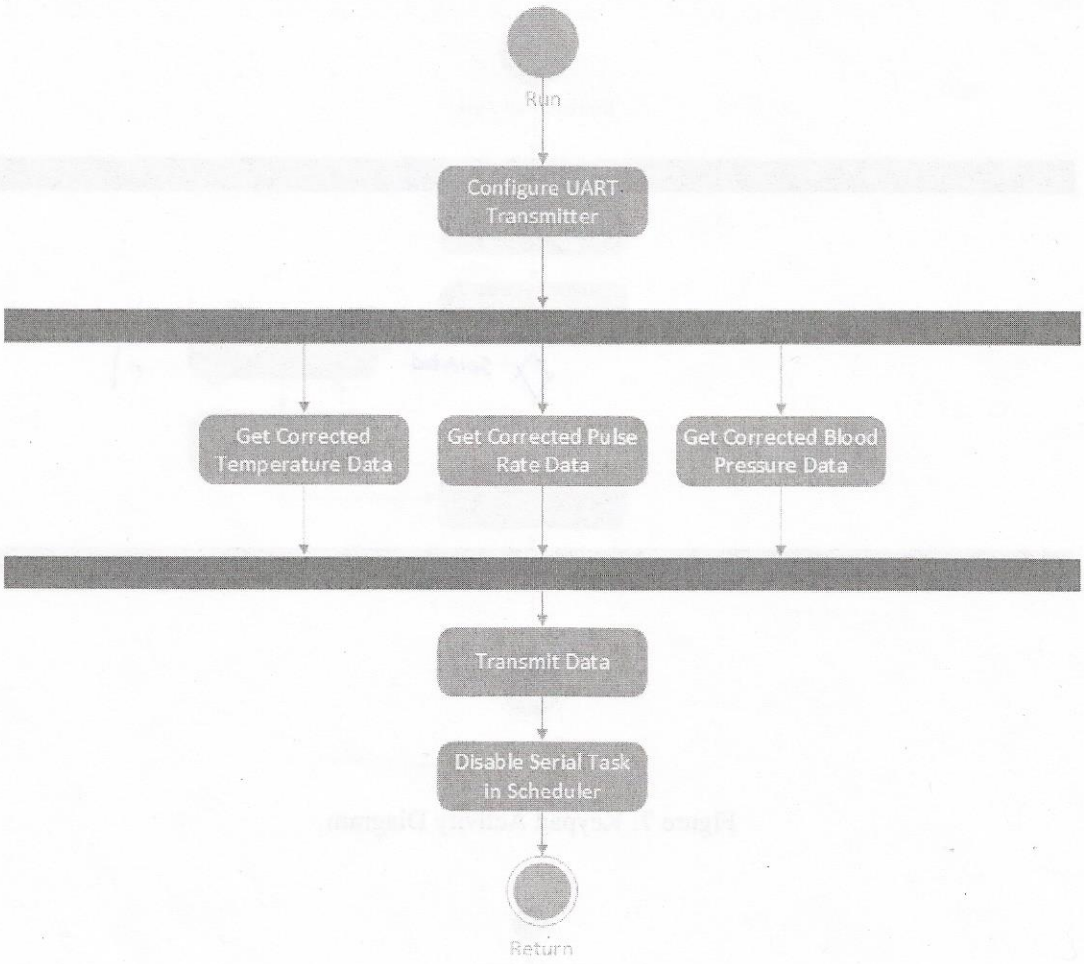


Figure 9: Serial Activity Diagram

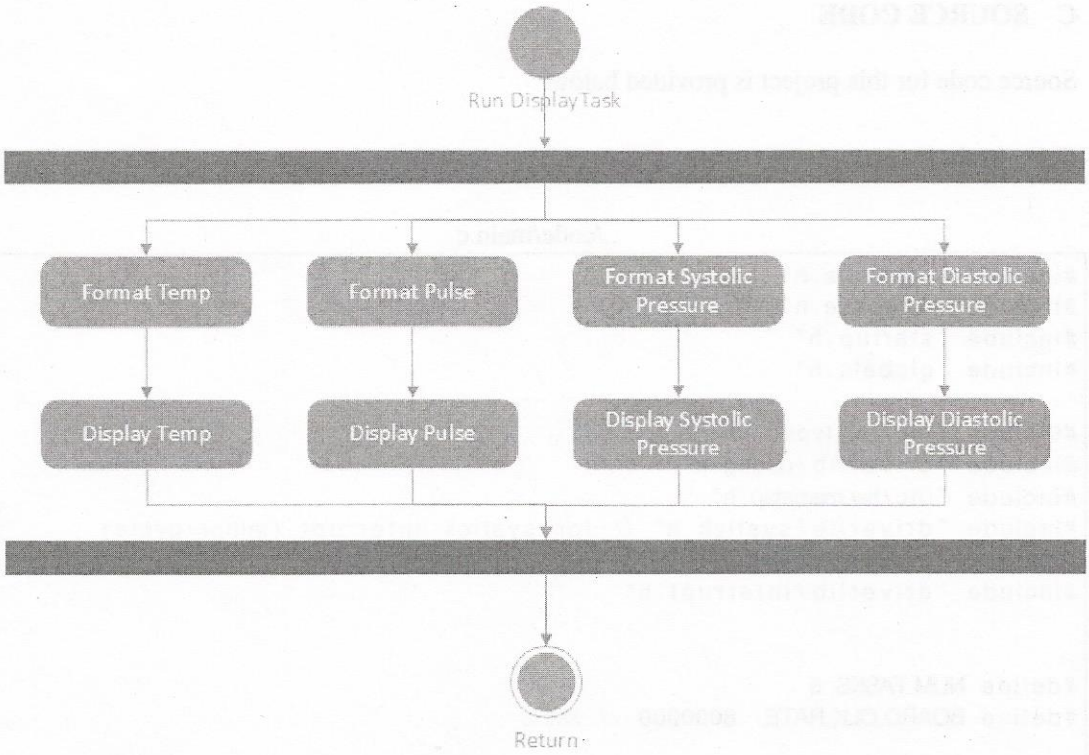


Figure 10: Display Activity Diagram

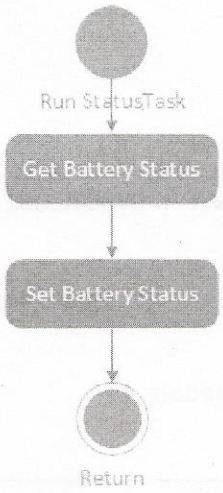


Figure 11: Status Activity Diagram

C SOURCE CODE

Source code for this project is provided below.

C.1 Main Function

../code/main.c

```
#include "schedule.h"
#include "timebase.h"
#include "startup.h"
#include "globals.h"

#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "inc/hw_memmap.h"
#include "driverlib/systick.h" // for systick interrupt (minor cycle)
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"


#define NUM_TASKS 5
#define BOARD_CLK_RATE  8000000 // 8MHz

#ifdef DEBUG
  void
__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif


int main(void) {
  // Set the clocking to run directly from the crystal.
  SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
      SYSCTL_XTAL_8MHZ);

  startup();     // initialize system state and variables

  while (1) {
      runTasks();  // from schedule.h
  }
}
```

C.2 Global Data

../code/globals.h

```
/*
 * globals.h
 * Author(s): Jonathan Ellington, Patrick Ma
 * 1/28/2014
```

	EE 472 Winter 2014	Team:	
	Lab 3 Report	Jonathan Ellington Patrick Ma	
	Grader: Patrick / Jocelyn	Jarrett Gaddy	
	Sections	**Points Possible**	**Points Received**
1	Organization	5	5
2	Formatting	5	5
3	Abstract	10	10
4	Introduction	10	10
5	Design Spec	56	32
6	Software Implementation	72	35
7	Presentation, Discussion and Analysis of Results	33	20
8	Analysis of Errors	20	20
9	Analysis of why project didn't work	5	5
10	Test Plan	10	10
11	Test Specification	15	15
12	Test Cases	15	15
13	Summary	10	10
14	Conclusion	10	10
15	Appendices	26	22
	$\lfloor 10$ grammar/spelling etc errors $/4 \rfloor = 2$	302	-2
	Total:	**302**	222