EE 472 Lab 1
Introducing the Lab Environment

Jonathan Ellington
Patrick Ma
Jarrett Gaddy

Contents

List of Tables

List of Figures

1 ABSTRACT

In this lab the students are to take on the role of an embedded system design team. They will design an exciting new medical instrument to monitor various patient metrics. When the device finds metrics are out of the acceptable range, the user will be notified, thus saving them from potential health risks. The students must first layout the design for their system using various design tools, then they must implement the system in software. Finally the students must test their system to make sure that it is ready to start saving lives.

2 INTRODUCTION

Brief introduction and overview of the purpose of the lab and of the methods and tools used.

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.

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 5 program tasks running the instrument take to execute. These tasks are mini prograams that each handle a part of the instruments purpose.

3 DISCUSSION OF THE LAB

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 datalogging functionality, and indicate when measured vitals exceed given ranges, or the user fails to comply with a prescribed logging regimen.

At the present time, only two subsystems must be produced: the display and alarm portions. Additionally, the system must demonstrate the ability to store basic measurements.

The initial 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

3.1.2 Detailed Specifications

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

The system must have the following inputs:

- Alarm acknowledgment capability using a pushbutton
- Sensor measurement input capability consisting of:
  * Body temperature measurement
  * Pulse rate measurement
  * 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

The initization 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.

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%. An alarm state occurs when any measurement falls outside its stated normal range by 10%.

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.
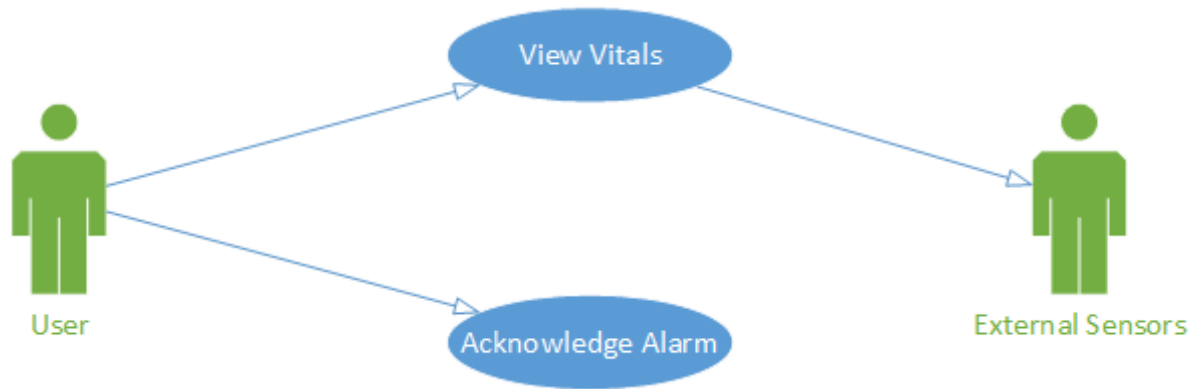
Figure 1: Use case diagram

3.1.3 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.

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. Finally, this value is displayed onscreen.

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 a specified amount time passes 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.

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 system was functionally decomposed into high-level blocks as shown in Figure 2. The main system control is located in the CPU, which controls all data flow into and out of the peripheral devices. The OLED displays the user's current vitals including blood pressure (systolic and diastolic), temperature, and pulse rate. In the future external sensors will be added, but for now the values are simulated using the CPU. The CPU also controls three LEDs colored green, yellow, and red. These LEDs are used to inform the user on the current state of their vitals as well as the state of the device. Under normal circumstances, the green LED will be lit. If the users' vitals fall outside of a specified range, the red LED will flash at a specified rate, dependant on which vital is out of range. If the battery is low, the yellow LED will be illuminated.
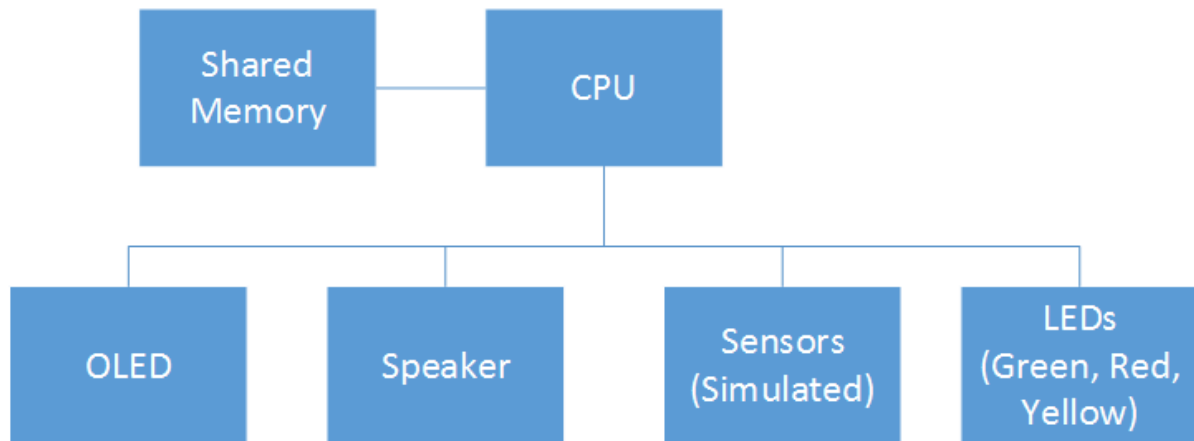
Figure 2: Functional Decomposition

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 five main tasks:

- **Measure Task** - In charge of interacting with the blood pressure, temperature, and pulse sensors (simulated)

- **Compute Task** - Converts sensor data into human readable format

- **Display Task** - Displays the measurements on the Stellaris OLED
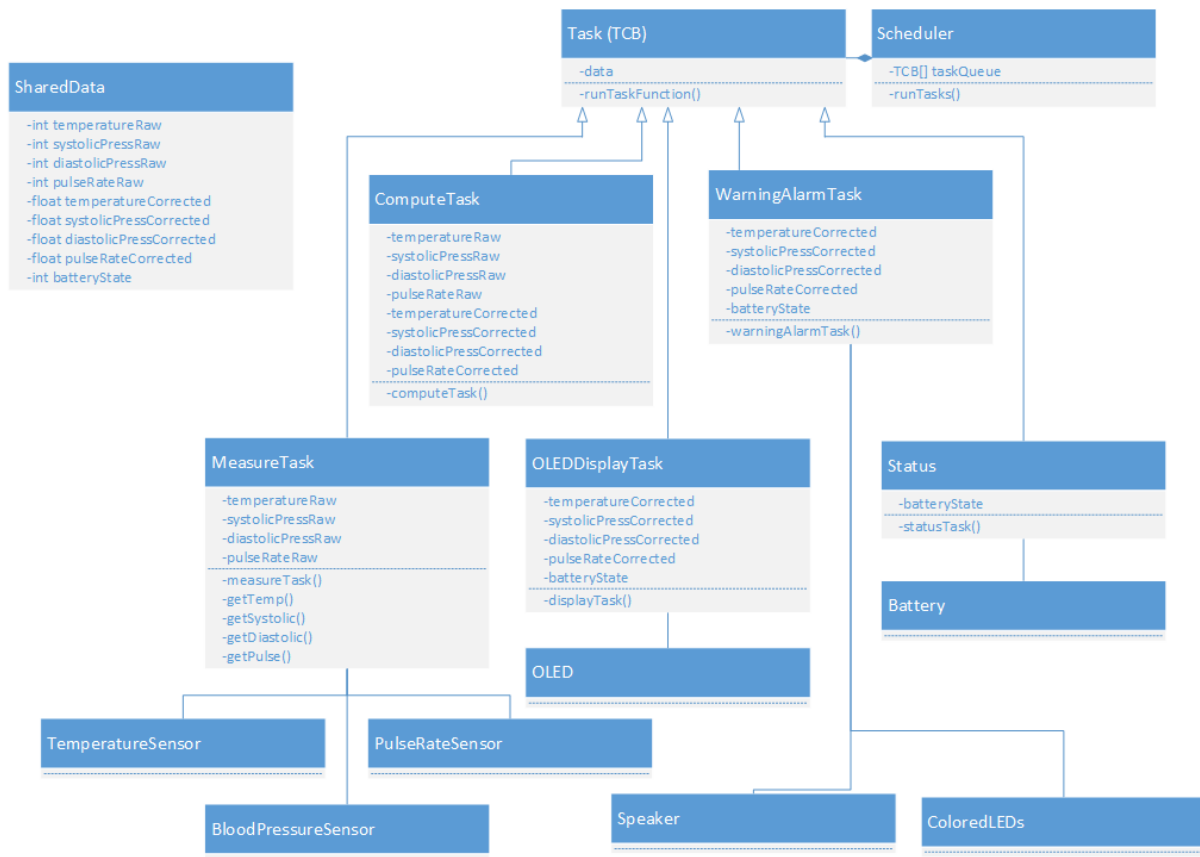
Figure 3: System Architecture Diagram

- **Warning/Alarm Task** - Interacts with the red, yellow, and green LEDs, as well as the speaker to annunciate warning and alarm information

- **Status Task** - Receives battery information from the device

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 multifile 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

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 follows:

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

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.

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 then runs into an infinite while loop. Inside the while loop a run method is called. The run method is part of the scheduler. This method keeps track of whether the device is on a minor cycle or a major cycle and runs the preform task method of each task. The tasks included in this project are Compute, Measure, Warning, oleddisplay, and status. Each task has a public interface of 2 void pointers. One that when initialized by the main method will point to the preform task function, and another that, when initialized, points to a struct containing pointers to the data required by that task. Each task has a task control block(TCB) in the scheduler. This TCB points contains pointers to the preform task function and the data for the task. The TCB is used by the scheduler to run the task. The scheduler contains an array of TCB, each element in the array corresponds to a new task. In this case there are 5 tasks so 5 elements in the array.The scheduler's run task contains a for loop that runs through the array of TCB and runs the function pointed to by the TCB with the argument of the data pointer stored in the TCB. After running all 5 tasks the TCB has a software defined wait of 250 milliseconds to implement the minor cycle delay. The software defined wait is simply a for loop that uses addition as a time consumption tool. it is known that system clock is 8 Mhz so having a for loop go from 0 to 2 million would take about a quarter of a second. The control flow is shown in the figure below in figure 5.
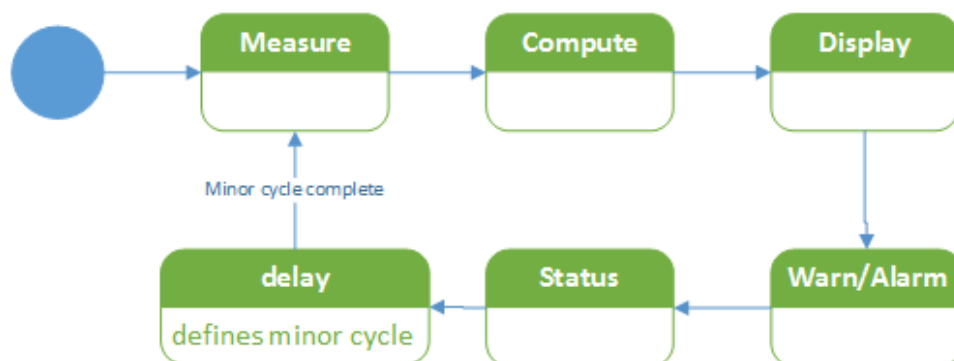

Figure 4: Control Flow Diagram

Each task has its own unique purpose in the system, and each uses a different part of the global data. The Measure task, with an activity diagram shown in Figure 6, deals only with the raw data from the instruments. This task is meant to act in place of the instruments that are unavailable. The task only runs if the scheduler has set the global value is Major Cycle to 1. On a major cycle the measure function either increments the data of each measurement by 1 or 2 or decrements the data by 1 or 2.

The compute task is very simple. Like the measure task it only runs on a minor cycle. It takes the raw data that has been set by the measure task, multiplies by a constand and adds a constant to each piece of raw data to get the corrected data. The compute task then puts the values for the corrected data in memory at the location of the global data pointer. Compute uses every data value except the battery.

Afer compute, the warning task begins checking for warning or alarm states. The warning task only deals with the corrected data from compute and the battery state. This task also must deal with the input and output signals used to display warning and sound an alarm. Unlike the other tasks, this task has more to its initialization than just initializing the data. In addition to setting up the pointers to the global data, during initialization, the task also enables peripheral banks C, E, F, and G. These are enabled using the SysCtlPeripheralEnable library call. Additionaly the task set up pins C 5, 6, and 7 as outputs, pins F 0 and G 1 as PWM outputs, and pin E 0 as a pull up resistor input. Additionaly the pwm outputsare set to use a 65 hz clock to play a sound at this frequency whenever enabled. The activity diagram is shown in Figure 8. There are 3 subsystems in the warning task. These subsystems each handle a different part of user notification. The first subsystem deals with the alarm. The subsystem checks to see if any of the given corrected data is outside of the range given to us as acceptable by more than 10%. If any value is then the PWM output is enabled using PWMGenEnable. If the values fall back within the acceptable range, or the user hits the acknowledge button, then the pwm is disabled with PWMGenDisable and the sound stops. The next subsystem checks the corrected data for being 5% out of range of the accepted values. If any value is more than 5% out of its range then a warning will be displayed on the red led connected to pin C 5 using the GPIOPinWrite function. Depending on the value that is out of range the period the led flashes at will vary. The final subsystem is the battery check. This system checks if there is more than 30% of battery left on the device. This is taken from the battery state data field. If there is less than 20% battery left then a yellow led connedted to pin C7 is illuminated, if there is more then 20% battery, and the device is not in a warning or alarm state then the green led on pin C 6 is illuminated.

To show a user their current medical measurements, the system also has an oleddisplay task. This task uses the corrected data from measurements, and the battery state. The display task has an activity diagram shown in Figure 9. This task uses the sprintf function in C to convert the data types that the corrected data is stored in, and properly format these data values into a string which is stored in a buffer. The string contained in the buffer is then printed to the oled screen using the driver library rit128x96x4 functions.

The last task is the status task. This task only deals with one piece of data which is the battery state data. The only thing the status task does is that on a major cycle, it decrements the battery state by 1. This is shown in the activity diagram in Figure 10

4 PRESENTATION, DISCUSSION, AND ANALYSIS OF THE RESULTS

4.1 Results

The project was completed and demonstrated on January 29, 2014.

Demonstration of the system to the interested parties showed that the system met the requrements initially presented at the onset of the lab project. Testing of the system prior to demonstration also verified that the system met the specifications listed in Section 3.1.

Additionally, during the demonstration, to show that several warning features worked as expected, source code was temporarily modified to speed up the progression of the system through various warning states. These changes were simple to execute and caused the desired effect on the system without causing unwanted abberrant behavior. Reverting the source code was similarly intuitive.

During the actual coding and implementation of the design, remarks were made several times about the ease of execution during that phase of the project. After the initial high level Design phase, very few changes were made, or required, to the functional design or system architecture.

Using an oscilloscope, the runtimes of each task were empirically determined. The results are given in Table 2.

Task	Runtime (μs)
Measure	23.4
Compute	55.4
Display	22900.0
Warning	27.4
Status	5.6

Table 2: Empirically determined task runtimes

Answers to the last three questions in the list of items to include in the project report:
You don't find the stealth submarine. That's why they are so expensive; at that cost, you take great pains to never lose one.
A helium balloon always rises. It just rises upside-down.
If you really managed to lose the stealth submersible, you first have to tell the government, which will deny it has any stealth submersibles, then you have to comb the seven seas until your comb hits the sub.

4.2 Discussion of Results

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 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.

Task	Instructions
Measure	187
Compute	443
Display	183,200
Warning	219
Status	45

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

4.3 Analysis of Any Errors

The project was completed without any residual errors or unsolved problems. See the following section for analysis of issues encountered while working on the project itself.

4.4 Analysis of Implementation Issues and Workarounds

The medical instrument design in this project was completed and tested succesfuly to meet all the requirements, the designers did face a number of errors and difficulties in the process. Most of the challanges faced in this design were in implementing the inputs and outputs from the ARM micro processor. In this implementation, the students orginally had consulted the driver library documentation and found how to enable a general purpose input output (GPIO) pand, GPIO pin, and how to read from a GPIO pin. However, when the code to enable a GPIO pin was executed, the execution of tasks within the scheduler froze completely until a reset. This problem was caused by the students failure to enable the peripheral bank that the GPIO pins existed on. After learning how to do this, code was added to enable the peripheral and the task execution no longer froze on enabling a pin.

Another problem encountered by the design team was that initially after setting up a GPIO pin as an input to take a button press from the Stellaris test board, pressing the button did not have the intended effect. The push button did not change the state of the input. The students were originally perplexed and tried a number of solutions to this problem. Testing indicated that the code for the GPIO input was correct as a 3.3 V signal directly to the pin could trigger the intended event on the input. Eventualy it was found that to use the push button switch the GPIO pin's pad must be configured to accept a pull up resistor type of input.

All problems were solved before demonstrating the product to the interested parties. As such, the final design did not have any errors.

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 slilence period

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

5.1 Test Specification Jon

Annotated description of what is to be tested and the test limits. This specification quantifies inputs, outputs, and constraints on the system. That is, it provides specific values for each.
Note, this does not specify test implementation...this is what to do, not how to do it.

5.1.1 Scheduler

The scheduler needs to be shown to correctly schedule and dispatch tasks. This means that task 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.

5.1.2 Measure Task

For this design, no external sensors were used; instead they were simulated. Each simulated sensor should be tested and verified against the specification, as follow.

- **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 requirements are very similar to measure, except the pulse should increase by three rather than two, and the range is between 15 and 40.

- **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.

5.1.3 Compute Task

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

- $CorrectedTemperature = 5 + 0.75 * RawTemperature$

- $CorrectedSystolicPressure = 9 + 2 * RawSystolicPressure$

- $CorrectedDiastolicPressure = 6 + 1.5 * RawTemperature$

- $CorrectedPulseRate = 8 + 3 * RawTemperature$

5.1.4 Display Task

The display task should be tested to print each corrected value properly on the screen, and update them as the computations are done.

5.1.5 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 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

5.1.6 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 Jarrett

The students begin testing by examining if the alarm sounds at the proper time. This is initialy 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 initialy tested this way. The initial values for raw data given in Table 1 on page 2 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 varried 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 individualy by being set to an initial raw value of 100. Once again, the green led started off because the system was not in a nirmal 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 acknowledgement 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 measuremet simulation functions were tested. This was done by re-enabing 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 reenabled, 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 intedned. This situation also gave the students an opprotunity 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 measurment function was once again disabled and the pulse rate measurement function was re-enabled. The oled display indicated that the pulse rate was increasing by 3 on a major cycle then falling by 6 on the following major cycle. This was consistent with the intended design. The alarm and warning being initiated as the pulse rate fell. The pulse rate measurement was then disabled and each blood pressure measurement was re-enabled individualy 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 our 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 fastsest blinking warning with a .5 second period.

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 our 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 oscciloscope 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.

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 varried 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 the students 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 individualy by being set to an initial raw value of 100. Once again, the green led started off because the system was not in a nirmal 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 acknowledgement 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 measuremet simulation functions were tested. This was done by re-enabing 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 reenabled, 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 intedned. This situation also gave the students an opprotunity 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 measurment function was once again disabled and the pulse rate measurement function was re-enabled. The oled display indicated that the pulse rate was increasing by 3 on a major cycle then falling by 6 on the following major cycle. This was consistent with the intended design. The alarm and warning being initiated as the pulse rate fell. The pulse rate measurement was then disabled and each blood pressure measurement was re-enabled individualy 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 fastsest blinking warning with a .5 second period.

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 ossciloscope 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

The students began creating their medical instrument through a rigorous design process at different levels of detail. The students then continued work by implementing their design in C code for the ARM Cortex A3 microprocessor. Next the code was tested and debugged using the IAR workbench debugging tool, as well as visual queues programmed into the design. Finally, after verifying that the system worked as specified, it was presented to the instructor.

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 thri design using visual and audio queues, the IAR embedded workbench debugger, and a few specifically programmed debug features. After the results of the testing verefied the design to be working correctly, the students proceeded to present their medical instrument to their instructor.
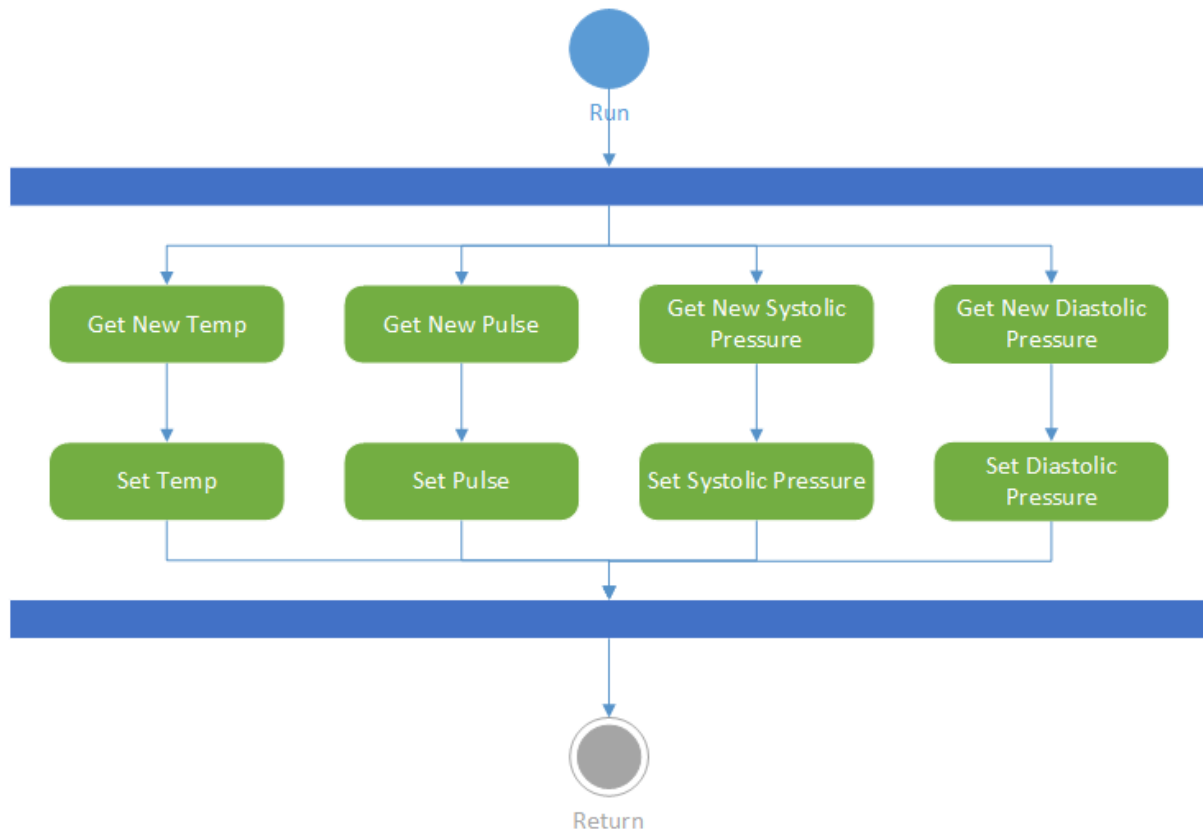
Figure 5: Measure Activity Diagram

A BREAKDOWN OF LAB PERSON-HOURS (ESTIMATED)

Person	Design Hrs	Code Hrs	Test/Debug Hrs	Documentation Hrs
Patrick	15	7	2	9
Jarret	x	x	x	x
Jonathan	x	x	x	x

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.

B ACTIVITY DIAGRAMS

C SOURCE CODE

Source code for this project is provided below.

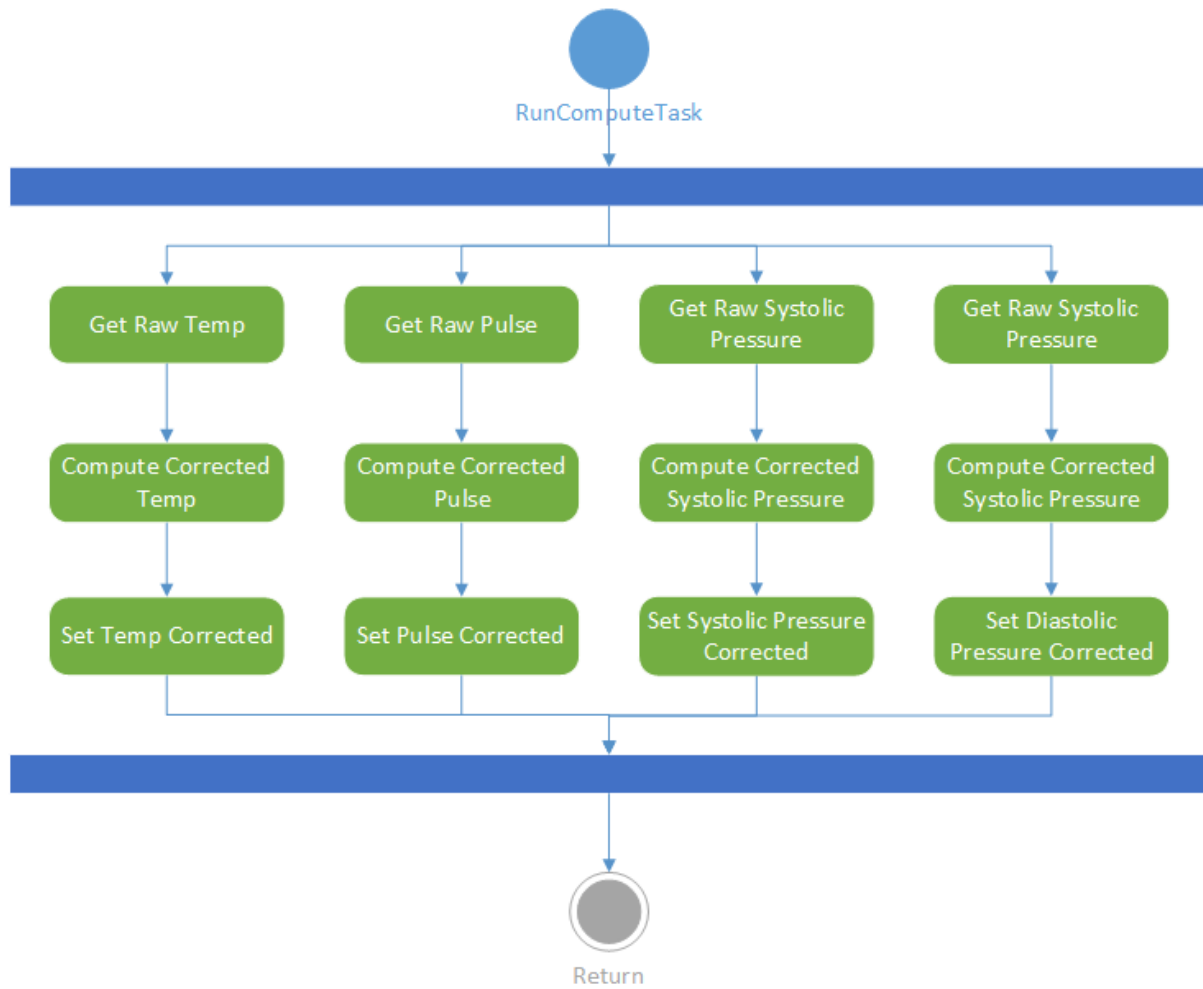### C.1 The first part

C.2 The second part

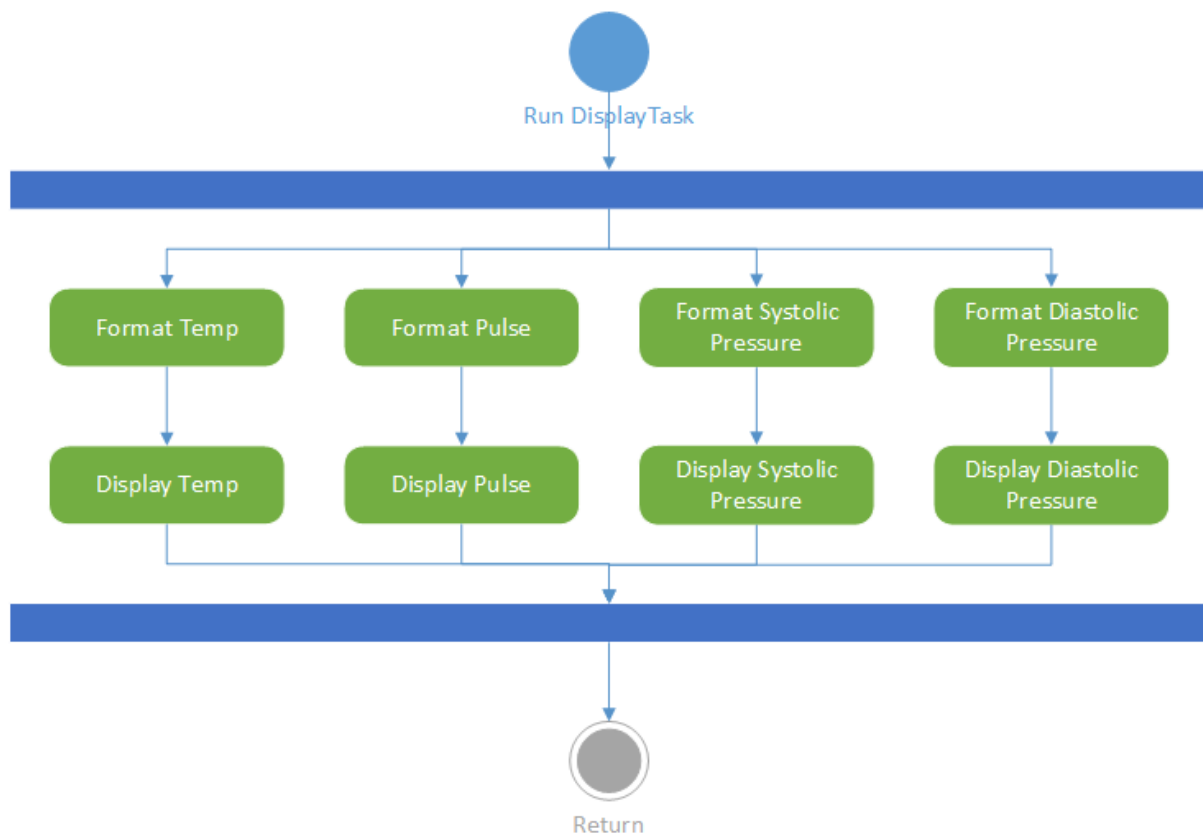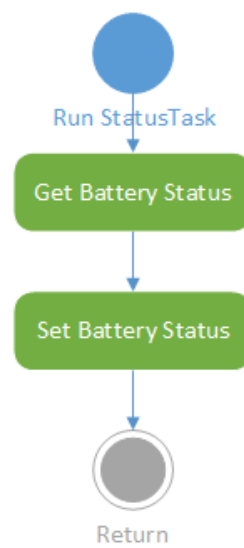Figure 6: Compute Activity Diagram

Figure 7: Warning Activity Diagram

Figure 8: Display Activity Diagram

Figure 9: Status Activity Diagram