Application of Real-Time Systems in Simulation of Water Tank Operation

CEC450: Real-Time Systems

Zackary Hagerty, Miles Osborne, Daniel Carter, Jesse Slager

Spring 2022

Dr. Omar Ochoa

Anton Kiselev

Document Control

Approval

All members of the Development Team and the customer, Dr. Omar Ochoa, shall approve of this document before release to stakeholders.

Distribution List

The following list of people will receive this document upon release. Other parties may receive this document with consent of all members of the Development Team.

Embry-Riddle Aeronautical University Professor:

Dr. Omar Ochoa

Development Team:

Zackary Hagerty
Daniel Carter
Miles Osborne
Jesse Slager

Change Summary

Document Created: 3/27/2022	**Version 1.0** **-** Document layout created, intro written
Version 1.1 Completed : 4/30/2022	**Version 1.1** - Each section written, refactoring to be done - Appendixes unwritten
Release Candidate Finished: 5/3/2022	**Version 2.2** **-**Added appendix -Completed main document, refactored written sections
Release Candidate Approved: 5/4/2022	

I. DEVELOPMENT ORGANIZATION

Developers worked individually and in groups. The below table depicts when more than one developer met to work on the project as a group, and does not reflect effort.

Meeting Date	**Description**	**Developers Present**
3/27/2022	Created project design, initialized original data flow diagram. Created original document	Jesse, Zack, Miles, Daniel
4/02/2022	Created project, created Github repository	Jesse, Zack, Miles, Daniel
4/03/2022	Began implementation of temperature sensors and pressure sensors. Miles begins GPIO work for audio system	Jesse, Miles
4/05/2022	Began implementation of water levels and valve control	Zack, Daniel
4/09/2022	Continued development of sensors. Changed to centralized development.	Jesse, Zack, Miles, Daniel
4/11/2022	Began fleshing out centralized manager, incorporating RTS objects	Jesse, Zack, Miles, Daniel
4/15/2022	Continued development of centralized manager, began incorporating semaphores	Jesse, Zack, Miles, Daniel
4/17/2022	Continued to implement RT Objects, began changing functionality of sensors. Miles completed GPIO driver development	Jesse, Zack, Miles
4/22/2022	Began development of message queue for system. Developing central looping task	Jesse, Zack, Miles, Daniel
4/23/2022	Working demo completed. Testing complete	Miles, Daniel
4/24/2022	Audio support fully implemented, video demos completed in time for presentation	Jesse, Zack, Daniel, Miles
4/25/2022	Further testing, presentation work	Jesse, Zack, Daniel, Miles

All developers put forth an insane amount of effort on this project, and are incredibly proud of the results

II. OBJECTIVES AND SYSTEM SPECIFICATIONS

Objectives

The objective of this assignment was to simulate the operation of a hot water tank in a real-time environment using VxWorks.

System Specifications

The contents of the project description are detailed below.

The project is a hot water tank simulator used to imitate the behavior of an automatic water tank monitoring system. There are several sensors and actuators involved in controlling the water level and the temperature in a tank.

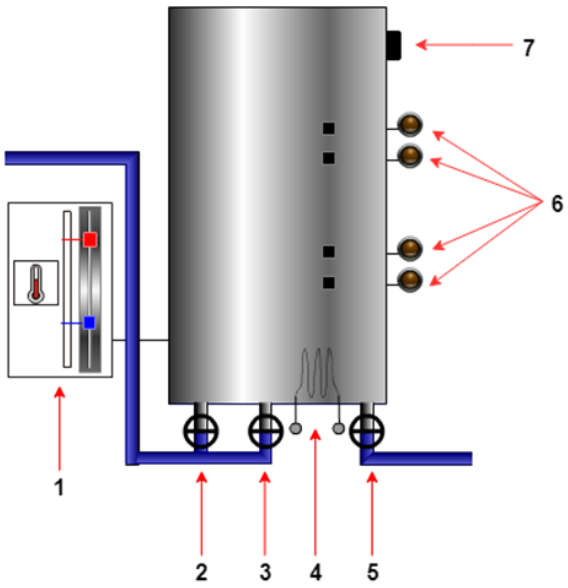

Figure 1. Graphical representation of the hot water tank. Numbers denote the following components of the system. [1] - Temperature sensors, [2] and [3] - inlet valves, [4] - water heater, [5] – outlet valve, [6] water level sensors, and [7] - pressure sensor.

As shown in the figure above, the system includes two inlets and one outlet valve. Water must start entering the tank, if any of the two inlets are open. In case if both inlets are open, the water level shall rise much quicker than if only one inlet was open. The outlet valve must drain water when opened. There are also four water level sensors in a tank. When inlet valves are open, and the tank is being filled up, the program shall indicate when water reaches a certain sensor. Additionally, if the water level reaches the highest sensor, the outlet valve must open to prevent overflow. Ideally, the water level should be kept moving between highest and lowest water level sensors.

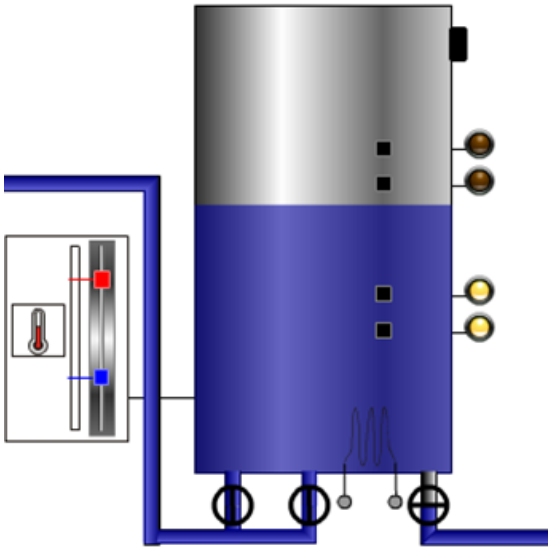

Figure 2. Graphical representation of the water tank being filled up. Two inlet valves are open and two sensors are indicating that the water level has reached and surpassed them.

Regarding the temperature, the water should be kept at a temperature that is between max and min temperature sensor values. The water heater should turn on if the temperature is too low, and turn off if the temperature is too high. The pressure sensor is set to make sure that the pressure inside the tank does not exceed normal operating pressure. The pressure should start building up if the water level surpasses the second to the highest water level sensor. Additionally, the pressure builds up quickly if the water temperature is kept close to 100 ℃. If the system reaches the critical pressure of 50 psi (344 kPa), the water drain must be open to provide pressure relief.

Here are some additional specifications of the water tank:

- Volume: 50 L
- Normal operating pressure: 20-30 psi (130 – 206 kPa)
- Critical pressure: 50 PSI
- Default max temperature: 100 ℃
- Default min temperature: 10 ℃

Some additional functionality includes the ability to move the sensors, (both water level and temperature), up and down, i.e. changing their max and min values.

III. DESIGN DESCRIPTION

For the hot water heater simulation system design, the developers designed a level zero Data Flow Diagram (DFD) as it goes over the main events and communications between entities and tasks within the system. As shown in Figure 3, the developers decided to implement a centralized system, with the goal of maximizing an architecture with low coupling and high cohesion. This minimizes the impact of changing one module and having it affect the rest of the system, and helps to contain any faults that arise within one subsystem before it affects others.. As shown, everything communicates with the main subsystem, “Tank Simulator”. All events that needed to communicate with each other did so via a struct that acted as a middle-ground basis for all subsystems through the centralized file, tanksim.c. Sensors can contact tankSim.c for important information without having to contact each other directly. This allows concurrent scheduling of tasks, calling each function when needed. Not only was the choice to develop a centralized system more time efficient due to this scheduling, but it also allowed the developers to troubleshoot the system easier along with capturing its functional behavior.

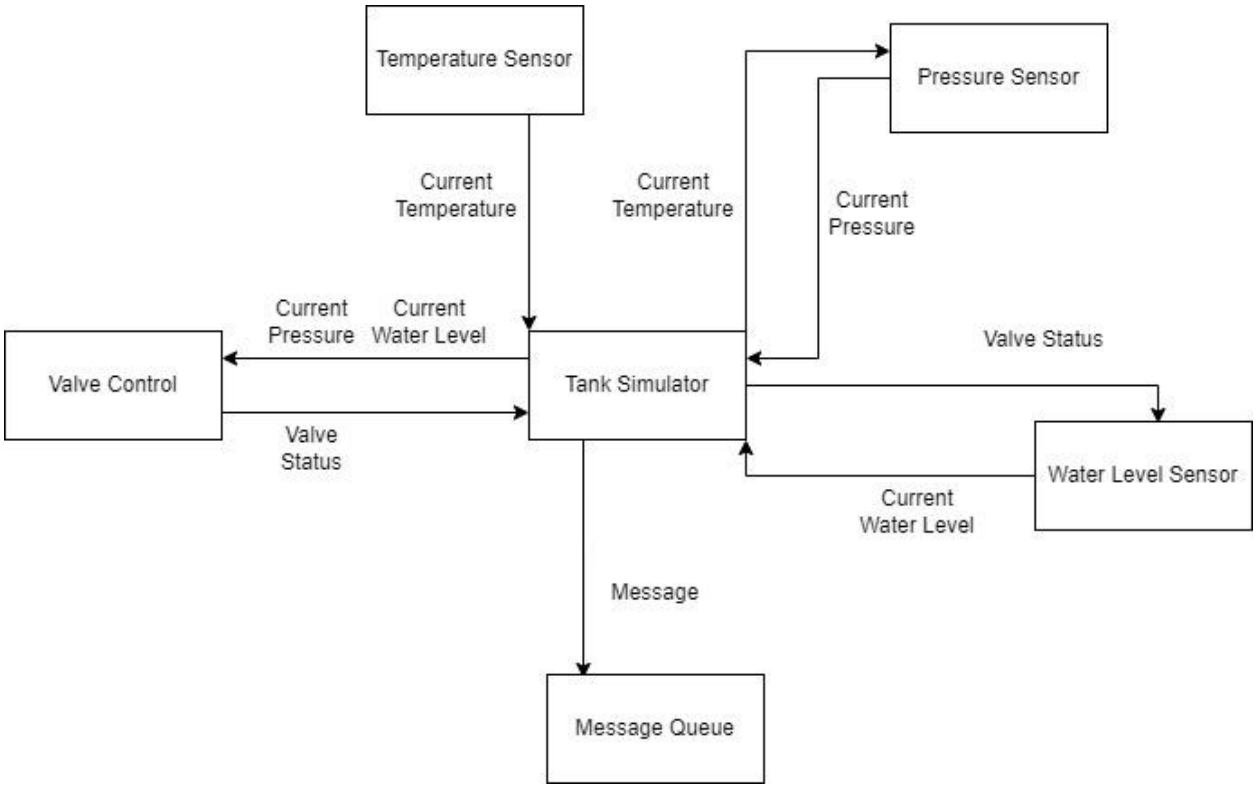

Figure 3: Level 0 Data Flow Diagram

The developers also designed a State Chart to model the behavior, flow, and communication between tasks and events of the system in more detail. As shown below in Figure 4, the system cycles through the following states:

1. The system starts with initializing the inlet valves and water heater element to the ‘on’ state, opening the valves and turning on the heater.
    a. When the water heating element is enabled, the water temperature will rise, but only if the current temperature is under the maximum, or below the minimum allotted temperature value set by the user when initializing the tank.
    b. If the current temperature in the tank is greater than the user-defined temperature value, the water heater will turn off, resulting in a decrease in water temperature.

c. By enabling the inlet valves, the water level in the tank increases as water enters the tank, which in turn also allows pressure to start building due to the outlet valve being closed.
    i. Consequently, water cannot drain out of the tank if the outlet valve is closed. This will in-turn enable the tanks pressure sensor, which will indicate the current pressure value within the tank.

2. As this is happening and water is entering the tank, assuming the current water level is below the first water level sensor, thes first sensor will turn on, indicating how much water is in the tank in liters.
    a. The same will occur for water level sensor two
    b. .However, when the water in the tank reaches water level sensor three, pressure in the tank will increase at a higher-than-normal rate.
    c. By the time water in the tank reaches water level sensor four, the inlet valves will be turned off and the outlet valve will open.
    d. Water will then begin draining out of the tank, decreasing the tank's pressure, and returning the water level in the tank to a safer level.

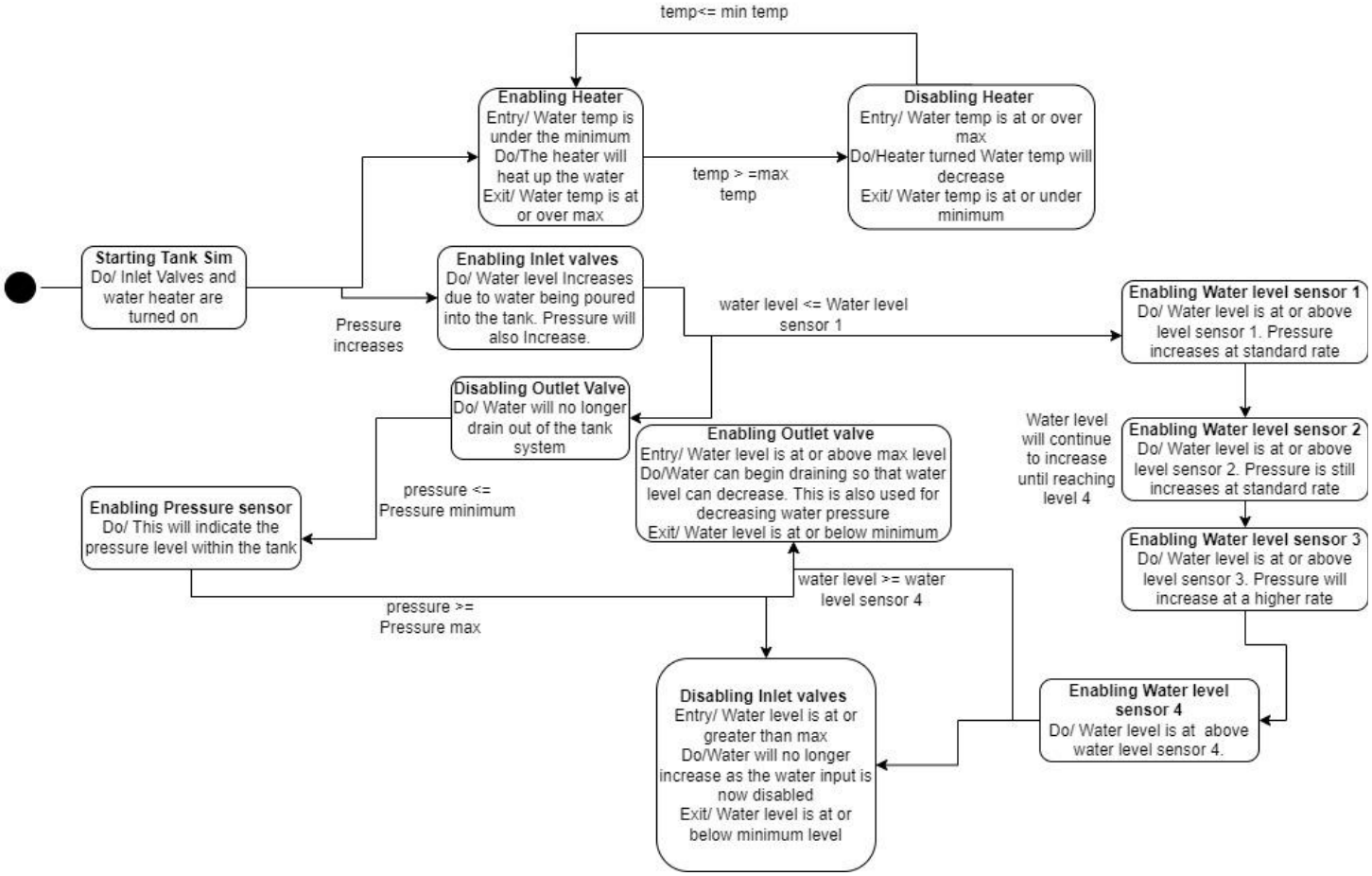

Figure 4: State Diagram

Required RTOS Technical Constraints

Following required real-time system technical constraints on this project, developers initiate each sensor as a task, and utilize semaphores to carry out communication between each sensor and the central tank struct. For this purpose, the programmers implemented seven tasks, four semaphores, one message queue, one watchdog, and six different instances of timestamping for the temperature, pressure, water level, inlet valves, and outlet valve subsystems.

System Characteristics and Assumptions

Developers also had to make a few assumptions about the functionality of the water tank. These assumptions were made to determine how the water tank will function:

1. The temperature changes at a constant rate of 2°C,
2. Both inlet valves open if any amount of water is entering the tank. There is never a case where one valve is open and the other is not.
3. Water flows in and out of the tank at a constant rate of 2 L/s
4. Pressure is building if the inlet valves are open (there will never be an instance of both the inlet and outlet valves being open at the same time), P
5. Pressure builds in the tank at a normal constant rate of 2.38psi and at a high constant rate of 7.14psi,T
6. The starting water level in the tank is five below the value of the minimum sensor level

The following sections go into great details about the software system's subsections. Each file mentioned can be observed in the

Central Subsystem: Tank Simulator

Starting with the main tankSim.c, this initialized a startup function, passing in the user defined temperature and water level value ranges within the tank, along with initializing the task tWaterTank, and incorporating functions that tested and determined if the users inputs to the system were valid. Furthermore, a startProgram function was defined, initializing a watchdog to determine if the system ever entered a deadlock state, or if the system timed out in any fashion. An updateConsole function was also defined, printing out the current values on the tanks temperature, pressure, water level, and both the inlet and outlet statuses. tankSim.c also had several functions setup which allowed data to be passed between itself and the other sensors, keeping everything centralized. This is where our four semaphores were defined and called throughout the program. Each of the subsystem files has a dedicated header file associated with it, prototyping functions and defining macros to be used and referenced in each program, but mainly, to allow tankSim.h to be in every subsystem, allowing concurrent communication between events.

Subsystem: Temperature Sensor

Starting with tempSensor.c, this subsystem moderates and regulates the current temperature in the tank. This is where the system's second task is created, tTemperatureControl, and where the developers define a current temperature check of the system, setting the water heater status to on or off. A current water heater check was also constructed, to increment or decrement the water heating rate of the tank at the given moment.

Subsystem: Pressure Sensor

The pressureSensor.c subsystem design was similar to that of the temperature sensor, i.e creating a task tPressureControl, however, developers had also defined a valve status check which incremented the current pressure in the tank at a normal or high rate if the inlet valves were open, or which decremented the normal pressure rate from the current pressure if the outlet valve was open.

Subsystem: Water Level Sensor

The waterLevelSensor.c subsystem design was similar to that of the temperature sensor, i.e creating a task tWaterControl, however, developers had also defined a check function, based on the outlet valve. If the valve was closed, the current water level in the tank would increase by a defined flowrate and indicate that the water level in the tank was in fact rising. However, if the outlet valve was open, the subsystem would close the two inlet valves, decrement the current water level by the same defined flow rate, and indicate a drop in water level in the tank.

Subsystem: Valve Control

The valveControl.c subsystem design was similar to that of the temperature sensor, i.e creating a task tValveControl, however, developers had also designed a definition based function, controlValves, which open and closes the input and output valves if the water level is below the first sensor or above the fourth sensor, or if the current pressure is outside the bounds of the allotted pressure threshold of the tank.

Subsystem: Message Queue

The messageQueue.c subsystem was designed to initialize the message queue for the system, allowing messages to be sent and received on a first-in first-out basis, allowing for general communication between each of the sensors and the main subsystem, Tank Simulator. This subsystem also defined the last task, tMessageQueueReciever, to initialize and moderate the flow of communication between each of the subsystems.

Additional Feature: Audio Integration

This addition was incorporated into the tankSim.c subsystem that would output a signal to a GPIO pin on the SABRE Lite Development Board, allowing a short audio track to be played whenever the water level reached one of the sensors. For this to work properly, a driver was written to configure IOMUX for GPIOs 5, 9, 16, 18, and 19 to ALT5 (GPIO mode), to have the ability to write a value to a designated GPIO pin, allowing an external device to output audio based on real-time functionalities of the centralized water tank simulator.

IV. OPERATION MANUAL AND ANALYSIS

NOTE: The following instructions assume the user(s) know how to properly set up the target system for development using the Workbench software. For assistance please refer to the Workbench documentation or Lab 0 instructions.

Required Hardware:

- SABRE Lite i.MX6 Development Board
- Adafruit Audio FX Sound Board
- 2w 8Ω
- 7 pin picoblade connector
- 8 pin picoblade connector
- 3~5v Power source*
- Host system with WindRiver workbench 3.0 (or greater) installed

NOTE: For demonstration purposes, developers utilized a 3 x AAA Battery Holder with On/Off Switch from Adafruit Industries.

NOTE: Refer to the SABRE lite development board for the location of connectors J7 (I2C3) and J14 (Android Buttons)

Hardware Setup:

1. Attach the power source or battery to the adafruit sound board.
2. Unscrew the terminals on the adafruit sound board
3. Insert the speaker wires into the terminals of the Adafruit sound board and rotate the screw clockwise to secure the wires.
4. Take the 7pin connector attached to the Adafruit sound board and install it into connector J7 (I2C3) on the SABRE Lite development board.
5. Take the 8pin connector attached to the Adafruit sound board and install it into connector J14 (Android Buttons) on the SABRE Lite development board.
6. Insert the ethernet cable into the SABRE lite development board and insert the USB end into the host system
7. Attach the serial cable to the wire labeled “CONSOLE” and insert the USB end into the host system.
8. Plug the DC power supply into a suitable wall outlet and insert the barrel jack into the connector on the SABRE Lite development board.

Program Instructions

1. Download the compiled project to the SABRE Lite development board.
2. To start the program, type startup in the program window. The four arguments of the startup function are as follows: maximum temperature, minimum temperature, height of the highest water sensor, and height of the lowest water sensor. As an example: type the following command: “startup(100, 10, 50, 10)”
3. Type the values of the four arguments mentioned in the previous step separated by commas with the entire sequence enclosed in parenthesis.
4. Press enter in the terminal window which will cause the program to start.
5. Upon successful activation, the Windows 98 sound should play from the speaker indicating the program is functioning as anticipated.

Analysis

The output of the system is constrained by a watchdog timer set to expire on the system clock rate multiplied by a factor of 10. Within this window, the status of the sensors, inlet valves, outlet valves, and audio settings must be managed during this time. Otherwise, an error message will display to the console via the logMsg function. Due to time constraints, the developers did not experiment with a more robust exception handling method. If time permits and future options for hardware were explored, provisions for more edge case scenarios and corresponding audio signals should be considered.

For the individual tasks, a round-robin scheduling algorithm is used to concurrently switch between tasks. Visually, the user can verify that the system performs as required and at least satisfies the functional requirements described above. However, due to time constraints, the developers are unable to perform a comprehensive review of task scheduling and formally verify if all tasks meet their deadlines. As a post project review, the developers would pursue a utilization bound test and response time tests gathering data from individual measurements and the system monitor. Other tools and methodologies are open to consideration.

V. OBSERVATION AND COMMENTS

Observations

The programmers observed that, during simulation time, the state of the water valves seems dictated by the rise and fall of the pressure within the system more than any other aspect. Because the pressure level rises so rapidly at higher temperatures, the water level will often not reach the highest sensor unless it is set rather low. This is a consequence of working with assumed values, rather than values given by the project description. The programmers chose not to take issue with this, because the project still fulfills all of its requirements. The pressure being the main dictator of the state of the water valves keeps the tank from ever approaching any values that would plunge the system into a critical state, which is valuable in a real time system.

Comments

The developers feel that the project description should be more specific in regards to the rates at which pressure, water flow, and temperature change. The programmers understand that requirement assumptions are a huge part of requirement specification, but having to make so many assumptions on top of an already difficult project was simply frustrating. A large percentage of the system's requirements ended up being based on assumptions because the system simply was not described in great detail. Normally, this would be fine, but not on a project where a large amount of the class has not gained extensive VxWorks development experience.

With that being said, this project was a fantastic way to gain experience in VxWorks. However, the developers feel that they could have accomplished so much more with their water tank if this project was introduced much earlier, perhaps around the time of lab three or four. The lack of actual VxWorks experience gained from the labs, combined with the difficulty of the project, made it difficult to find room for innovation within the cramped development window. Furthermore, a longer development window would have curbed a lot of the troubles that originated from the difficulties discussed in the next session.

VI. LESSONS LEARNED

Lessons Learned

The lessons learned by the programmers are listed below as bullet points for the sake of readability.

- The programmers learned extensively about the functionality of VxWorks Objects in creating an efficient real time system.
    - This is especially true for the use of watchdogs and message queues, both of which the developers felt insecure about using before working on the project.
- The programmers gained further familiarity with the VxWorks interface,
- The programmers gained more experience in building centralized software systems.
    - Though they acknowledge that it likely may not be the ideal project structure in other real time applications.
- While all programmers within the group know C, it's been a few years since they've done dedicated projects in C, so this project was a nice refresher.
- Writing device drivers for a real-time embedded system was a much different experience than writing drivers for non real-time embedded devices. Being able to pass pointers between functions quickly became an issue rather than a commodity. This problem stressed the importance of being able to adjust style to fit the development environment which the programmers were able to do.
- The NXP I.MX6 Quad reference manual was very difficult to understand and parse in order to determine what configuration would allow the use of GPIO pins. But it did improve the programmers ability to execute only from limited specifications and documentation.

Difficulties

The main difficulty the programmers faced during development was that they lacked the experience necessary to truly implement a good water tank system quickly and efficiently. While they are proud of the final product, it took them the better part of four weeks and about 45 hours of work. To remedy this situation in the future, the programmers humbly suggest that the instructor incorporate more actual coding tasks into the labs. Instead of utilizing the labs as basic introductions to VxWorks objects and nothing more, create small coding tasks so that the students gain experience implementing the objects themselves before moving on to this project. For the first week or so, it truly felt like being thrown into the deep end of the pool, which was quite overwhelming.

The only other difficulty the programmers had was the absolutely brutal difficulty of the month preceding the submission of this project. Every member of this team worked tirelessly to complete this project, but it became difficult to maintain morale when every other class demanded a similar amount of effort. There's nothing that can be done to remedy this situation- sometimes, that's just how it is- but it does mean that the programmers regard the finished product with pride in their work- a testament to their resilience during such a taxing semester.

Utilization of Software Engineering Concepts

The programmers utilized many standard software practices throughout the development of the project, albeit slightly modified to accommodate for such a short development cycle. Similar to subsection A, the lessons the developers learned are listed below for the sake of readability.

- As discussed in Section III, our design requirements informed every step of the solution's implementation.
- Because of the short development period, the programmers chose not to implement sprints.
    - However, the developers did have regular meetings to discuss the completion of work items, and the creation of new ones.
- Source control was utilized to ensure iterative development could still be effectively carried out.

APPENDIX A

The purpose of this section is to elaborate on the source code of the entire project. Appendix A is broken into three subsections, which are as follows:

Base Project: describes the base water tank project, and all the code that fulfills the technical requirement set forth by Ochoa Industries

Audio Functionality: describes the operations performed by the audio interface in relation to the GPIO and Adafruit sound board

NXP I.MX6 GPIO Driver: describes the operations performed by the partial GPIO driver of the NXP I.MX6 Quad processor

Each file group is accompanied by a basic description of the functionality and the role it plays in the system.

Base Project

messageQueue.c

Note: Message Queue.c and .h's primary purpose is to implement and maintain a message queue throughout the duration of simulation. Upon runtime, the timestruct, sender, and receiver are intialized; the sender listens for messages throughout sim time, which are sent every sixty ticks and displayed in the console.

```
/** @file messageQueue.c
 *
 *
 *  @author Daniel Carter, Zack Hagerty
 *
 *  this is a very stripped down version of the messageQueue from the labs.
 *
 */

#include <vxWorks.h>
#include <stdio.h>
#include <taskLib.h>
#include <sysLib.h>
#include <msgQLib.h>
#include <timers.h>
#include <time.h>
#include <stdint.h>
#include "messageQueue.h"


void InititalizeTimeStruct()
{
    timeStamping.tv_seconds = 0;
    timeStamping.tv_nanoseconds = 0;
    tickSet(0);
    clock_settime(CLOCK_REALTIME, &timeStamping);
    clock_gettime(CLOCK_REALTIME, &timeStamping);
}


void createMessageQueue(void) /* function to create the message queue and two tasks */
{
        if ((mqId = msgQCreate(MAX_MESSAGES,MAX_MESSAGE_LENGTH,MSG_Q_FIFO)) == NULL)
        {
        printf("msgQCreate in failed\n");
        }
}

void Sender(char msg[]) /* task that writes to the message queue */
{

        /* create and send message */
        if((msgQSend(mqId,msg,MAX_MESSAGE_LENGTH, WAIT_FOREVER,
        MSG_PRI_NORMAL)) == ERROR)
        {
             printf("msgQSend in Sender failed\n");
        }
```

```
48          else
49          {
50                  printf("\n");
51          }
52  }
53
54  void Receiver()
55  {
56          char  msg[MAX_MESSAGE_LENGTH];
57          while(1)
58          {
59                  //Message Received
60                  if(msgQReceive(mqId, msg, MAX_MESSAGE_LENGTH, WAIT_FOREVER) == ERROR)
61                  {
62                          printf("msgQReceive in Receiver failed\n");
63                  } else
64                  {
65                           clock_gettime(CLOCK_REALTIME, &timeStamping);
66                           printf("RECEIVER: %s at %d seconds %d nanoseconds and %d ticks\n", msg, timeStamping.tv_seconds, timeStamping.tv_nanoseconds, tickGet());
67                  }
68          }
69  }
70
71
72
73
74  void InitializeEverything() {
75      InititalizeTimeStruct();
76      createMessageQueue();
77      taskSpawn("tMessageQueueReceiver", 100, 0x100, 2000, (FUNCPTR) Receiver, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
78  }
```

messageQueue.h

```
/** @file messageQueue.h
 *
 *
 *  @author Daniel Carter
 *
 *
 *
 */
#ifndef _MESSAGE_QUEUE_H
#define __MESSAGE_QUEUE_H
#include <vxWorks.h>
#include <stdio.h>
#include <taskLib.h>
#include <sysLib.h>
#include <msgQLib.h>
#include <timers.h>
#include <time.h>
#include <stdint.h>

/* defines */

#define MAX_MESSAGES 100
#define MAX_MESSAGE_LENGTH 50
/* globals */

struct Time {
        time_t tv_seconds;
        time_t tv_nanoseconds;
}timeStamping ;

MSG_Q_ID mqId;

void Sender(char msg[]);
void Receiver();
void InititalizeTimeStruct();
void createMessageQueue(void);
void InitializeEverything();

#endif
```

pressureSensor.c

Note: The tankSim files act as our centralized system for the simulator. All messages sent to the tank sim are sent by the sensors. The messages received will allow for the messages to be sent to valve control or the water heater to determine whether or not it is necessary to activate or deactivate the valves.

```
/**
 * @file pressureSensor.c
 *
 * @brief This file defines the functions utilized for the water pressure sensor
 *
 * @author Jesse Slager
 *
 */

//pressure sensor needs to do these things:
// The  pressure  should  start  building  up  if  the  water  level  surpasses  the second  to  the  highest  water  level  sensor.
// If the system reaches the critical pressure of 50 psi (344 kPa), the water drain must be open to provide a pressure relief. <= drain valve I/O
// The  pressure  builds  up  quickly  is  the  water temperature is kept close to a 100 °C. <= temp sensor

#include "pressureSensor.h"
#include "tankSim.h"


// Checks for valve status and changes pressure build rate accordingly


void pressureSetup()
{
	taskSpawn("tPressureControl", 90, 0x100, 2000, (FUNCPTR) initializePressureSensor, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
}

void initializePressureSensor()
{
	while(1)
	{
		pressureValve();
		pressureLowering();
		taskDelay(sysClkRateGet() / 1 );
	}
}

void pressureValve()
{
	if((tankConfig.valves[0] == 1) && (tankConfig.valves[1] == 1) && (tankConfig.currentPressure < tankConfig.maxPressure)) {
		tankConfig.currentPressure += NORMAL_PRESSURE_RATE;
	}else if ((tankConfig.valves[0] == 1) && (tankConfig.valves[1] == 1) && (tankConfig.waterLevel > tankConfig.sensorLevelThree) && (tankConfig.currentTemperature >= 87)
		tankConfig.currentPressure += HIGH_PRESSURE_RATE;
	}
}


void pressureLowering()
{
	if (tankConfig.valves[2] == 1)
	{
		tankConfig.currentPressure -= NORMAL_PRESSURE_RATE;
	}
}

```

pressureSensor.h

```c
/** @file pressureSensor.h
 *
 * @brief
 *
 * @author Jesse Slager
 *
 */

#ifndef __PRES_SENSOR_H
#define __PRES_SENSOR_H

#include <vxWorks.h>
#include <stdio.h>
#include <taskLib.h>
#include <sysLib.h>
#include <msgQLib.h>
#include <timers.h>
#include <time.h>
#include <stdint.h>
#include "tankSim.h"

#define OUTLET_VALVE_OPEN   1
#define INLET_VALVE_OPEN    1
#define OUTLET_VALVE_CLOSED 0
#define INLET_VALVE_CLOSED  0


void pressureSetup();
/**
 * @name reportPressure
 * @param
 * @brief Get the current temp and sent message to water heater and pressure sensor
 */
void initializePressureSensor();

/**
 * @name getPres
 * @param
 * @brief Return the current temperature value
 */
void pressureValve(void);

/**
 * @name updatePressure
 * @param
 * @brief Get status from water heater and update/set water temperature accordingly
 */
```

```
42
43  /**
44   * @name updatePressure
45   * @param
46   * @brief Get status from water heater and update/set water temperature accordingly
47   */
48
49  void pressureLowering();
50
51  #endif
```

tankSim.c

Note: The tankSim files act as our centralized system for the simulator. All messages sent to the tank sim are sent by the sensors. The messages received will allow for the messages to be sent to valve control or the water heater to determine whether or not it is necessary to activate or deactivate the valves.

```
/* @file tankSim.c
 *
 * @brief
 *
 * @author Miles Osborne
 *
 *
 */

#include <stdlib.h>
#include "tempSensor.h"
#include "tankSim.h"
#include "Audio/audio_interface.h"
#include <stdio.h>

//Set up all the semaphores. Migth be intention of the above function, but I did not write that so I'm leaving it alone
void startup(int maxTemp, int minTemp, int waterSensorHighest,
                int waterSensorLowest) {
        valueTests(maxTemp, minTemp, waterSensorHighest, waterSensorLowest);
        InitializeEverything();
        int difference = waterSensorHighest - waterSensorLowest;
        int spacing = difference / 4;

        tankConfig.waterLevelSemaphore = semMCreate(0);
        tankConfig.valveSemaphore = semMCreate(0);
        tankConfig.temperatureSemaphore = semMCreate(0);
        tankConfig.pressureSemaphore = semMCreate(0);

        tankConfig.maxPressure = HIGH_PRES_THRESHOLD;
        tankConfig.currentPressure = 14.7;

        tankConfig.sensorLevelFour = waterSensorHighest;
        tankConfig.sensorLevelThree = waterSensorHighest - spacing;
        tankConfig.sensorLevelTwo = waterSensorLowest + spacing;
        tankConfig.sensorLevelOne = waterSensorLowest;
        tankConfig.waterLevel =
                        (waterSensorLowest - 5 < 0) ? 0 : waterSensorLowest - 5;
        tankConfig.waterRising = true;

        tankConfig.minTemperature = minTemp;
        tankConfig.maxTemperature = maxTemp;
        tankConfig.currentTemperature = minTemp;
        tankConfig.heaterOn = true;

        tankConfig.valves[INPUT_VALVE_2_ID] = ACTIVE;
        tankConfig.valves[INPUT_VALVE_1_ID] = ACTIVE;
        tankConfig.valves[OUTPUT_VALVE_ID] = INACTIVE;

        tankConfig.levelSensor[FOURTH_WATER_SENSOR] = INACTIVE;
```

```
50          tankConfig.levelSensor[THIRD_WATER_SENSOR] = INACTIVE;
51          tankConfig.levelSensor[SECOND_WATER_SENSOR] = INACTIVE;
52          tankConfig.levelSensor[FIRST_WATER_SENSOR] = INACTIVE;
53
54          taskSpawn("tWaterTank", 100, 0x100, 2000, (FUNCPTR) startProgram, 0, 0, 0,
55                          0, 0, 0, 0, 0, 0, 0);
56          //TODO: RUN AUDIO HERE!!!!!
57          audio_init();
58          play_track(1); //Play startup track
59          taskDelay(20);
60
61  }
62
63  void valueTests(int maxTemp, int minTemp, int waterSensorHighest,
64                  int waterSensorLowest) {
65          if (maxTemp < minTemp) {
66                  printf(
67                                  "MaxTemperature Less than Minumum Temperature. Reset board and attempt with correct paramaters.");
68                  exit(0);
69          }
70          if (waterSensorHighest < waterSensorLowest) {
71                  printf(
72                                  "WaterSensor Highest Lower than WaterSensor Lowest. Reset board and attempt with correct paramaters.");
73                  exit(0);
74          }
75          if (maxTemp < 0 || minTemp < 0 || waterSensorHighest < 0
76                          || waterSensorLowest < 0) {
77                  printf(
78                                  "Negative value detected. Reset board and attempt with correct paramaters.");
79                  exit(0);
80          }
81
82          if (waterSensorHighest > 50) {
83                  printf(
84                                  "waterSensorHighest above tank's limit. Reset board and attempt with correct paramaters.");
85                  exit(0);
86          }
87  }
88
89  // Function definitions
90  void startProgram() {
91          temperatureSetup();
92          valveSetup();
93          pressureSetup();
94          waterLevelSetup();
95          wd_tank_sim = wdCreate();
96
97          while (1) {
```

```
98                  wdStart(wd_tank_sim, sysClkRateGet() * 10, logMsg,
99                                  "Help, I've timed out!");
100
101                 //printf("Input valve 1: %d\n", tankConfig.valves[0]);
102                 //printf("Input valve 2: %d\n", tankConfig.valves[1]);
103                 //printf("Output valve: %d\n\n", tankConfig.valves[2]);
104                 taskDelay(60);
105                 updateConsole();
106                 updateAudio();
107                 //taskDelay(sysClkRateGet () / 60);
108         }
109
110 }
111
112 void updateConsole() {
113         char temp[MAX_MESSAGE_LENGTH];
114         char pressure[MAX_MESSAGE_LENGTH];
115         char waterlevel[MAX_MESSAGE_LENGTH];
116         char inlet_valve_1[MAX_MESSAGE_LENGTH];
117         char inlet_valve_2[MAX_MESSAGE_LENGTH];
118         char outlet_valve[MAX_MESSAGE_LENGTH];
119
120         sprintf(temp, "Temp = %hi ", tankConfig.currentTemperature);
121         sprintf(pressure, "Pressure = %f ", tankConfig.currentPressure);
122         sprintf(waterlevel, "Water Level = %u ", tankConfig.waterLevel);
123         if (tankConfig.valves[0] == 1) {
124                 sprintf(inlet_valve_1, "Inlet valve 1 status: OPEN");
125         } else {
126                 sprintf(inlet_valve_1, "Inlet valve 1 status: CLOSED");
127         }
128
129         if (tankConfig.valves[1] == 1) {
130                 sprintf(inlet_valve_2, "Inlet valve 2 status: OPEN");
131         } else {
132                 sprintf(inlet_valve_2, "Inlet valve 2 status: CLOSED");
133         }
134
135         if (tankConfig.valves[2] == 1) {
136                 sprintf(outlet_valve, "Outlet valve status: OPEN");
137         } else {
138                 sprintf(outlet_valve, "Outlet valve status: CLOSED");
139         }
140
141
142         Sender(temp);
143         Sender(pressure);
144         Sender(waterlevel);
145         Sender(inlet_valve_1);
```

```
146         Sender(inlet_valve_2);
147         Sender(outlet_valve);
148
149 }
150
151 void setHeaterStatus(bool state) {
152         semTake(tankConfig.temperatureSemaphore, WAIT_FOREVER);
153
154         tankConfig.heaterOn = state;
155
156         semGive(tankConfig.temperatureSemaphore);
157 }
158
159 void setValveStatus(int id, int state) {
160         semTake(tankConfig.valveSemaphore, WAIT_FOREVER);
161
162         //p
163         tankConfig.valves[id] = state;
164
165         semGive(tankConfig.valveSemaphore);
166 }
167
168 double getPressure() {
169         semTake(tankConfig.pressureSemaphore, WAIT_FOREVER);
170
171         double getPressure = tankConfig.currentPressure;
172
173         semGive(tankConfig.pressureSemaphore);
174
175         return getPressure;
176 }
177
178 int getWaterSensorStatus(int sensor) {
179
180         semTake(tankConfig.waterLevelSemaphore, WAIT_FOREVER);
181
182         if (tankConfig.waterRising) {
183                 if (tankConfig.waterLevel >= tankConfig.sensorLevelOne) {
184                         tankConfig.levelSensor[FIRST_WATER_SENSOR] = ACTIVE;
185                 }
186                 if (tankConfig.waterLevel >= tankConfig.sensorLevelTwo) {
187                         tankConfig.levelSensor[SECOND_WATER_SENSOR] = ACTIVE;
188                 }
189                 if (tankConfig.waterLevel >= tankConfig.sensorLevelThree) {
190                         tankConfig.levelSensor[THIRD_WATER_SENSOR] = ACTIVE;
191                 }
192                 if (tankConfig.waterLevel >= tankConfig.sensorLevelFour) {
193                         printf("Hey you reached the top!!!\n");
```

```
194                         tankConfig.levelSensor[FOURTH_WATER_SENSOR] = ACTIVE;
195                         tankConfig.waterRising = false;
196                 }
197         } else {
198                 if (tankConfig.waterLevel <= tankConfig.sensorLevelOne) {
199                         tankConfig.levelSensor[FIRST_WATER_SENSOR] = INACTIVE;
200                 }
201                 if (tankConfig.waterLevel <= tankConfig.sensorLevelTwo) {
202                         tankConfig.levelSensor[SECOND_WATER_SENSOR] = INACTIVE;
203                 }
204                 if (tankConfig.waterLevel <= tankConfig.sensorLevelThree) {
205                         tankConfig.levelSensor[THIRD_WATER_SENSOR] = INACTIVE;
206                 }
207                 if (tankConfig.waterLevel <= tankConfig.sensorLevelFour) {
208                         tankConfig.levelSensor[FOURTH_WATER_SENSOR] = INACTIVE;
209                 }
210         }
211         semGive(tankConfig.waterLevelSemaphore);
212         return tankConfig.levelSensor[sensor - 1];
213 }
214
215 void updateAudio(void) {
216         if (tankConfig.waterLevel <= tankConfig.sensorLevelOne) {
217                 play_track(2);
218         }
219         if (tankConfig.waterLevel <= tankConfig.sensorLevelTwo) {
220                 play_track(3);
221         }
222         if (tankConfig.waterLevel <= tankConfig.sensorLevelThree) {
223                 play_track(4);
224         }
225         if (tankConfig.waterLevel <= tankConfig.sensorLevelFour) {
226                 play_track(5);
227         }
228 }
229
```

tankSim.h

```
/** @file tankSim.h
 *
 * @brief
 *
 * @author Zack Hagerty, Jesse Slager
 *
 */
#ifndef __TANK_SIM_
#define __TANK_SIM_

#include <stdint.h>


//Input and Output valve array IDs
#define INPUT_VALVE_1_ID 0
#define INPUT_VALVE_2_ID 1
#define OUTPUT_VALVE_ID 2

//Water Level Array IDs
#define FOURTH_WATER_SENSOR 3
#define THIRD_WATER_SENSOR 2
#define SECOND_WATER_SENSOR 1
#define FIRST_WATER_SENSOR 0

// Sensor attributes
#define LOW_TEMP_THRESHOLD      10 //Water heater should turn on if sensor report value at or less than threshold
#define HIGH_TEMP_THRESHOLD     100//Water heater should turn off if sensor report value at or greater than threshold

#define WATERHEATINGRATE                2

// Sensor attributes
#define LOW_PRES_THRESHOLD      10 //Water heater should turn on if sensor report value at or less than threshold
#define HIGH_PRES_THRESHOLD     49//Water heater should turn off if sensor report value at or greater than threshold
#define TEMP_PRESSURE_THRESHOLD 100 //Pressure should rise if temperature is greater than or equal to specified value

//Temperature sensor predefined messages
#define ACTIVE 1
#define INACTIVE 0

#define FLOWRATE 2

//Water heating rates per 1C
#define NORMAL_PRESSURE_RATE    2.38 //TODO: Decide on a fixed/variable rate for temperature to increase by when water heater is active
#define HIGH_PRESSURE_RATE      7.14

#define MAX_MESSAGE_LENGTH 50

```

```
typedef int bool;
#define true 1
#define false 0

WDOG_ID wd_tank_sim;

 typedef struct{
    SEM_ID waterLevelSemaphore;
    SEM_ID valveSemaphore;
    SEM_ID temperatureSemaphore;
    SEM_ID pressureSemaphore;

        double maxPressure;
        double criticalPressure;
        double operatingPressure;
    double currentPressure;

        int8_t waterLevel;
        bool waterRising;

    int8_t sensorLevelFour;
    int8_t sensorLevelThree;
    int8_t sensorLevelTwo;
    int8_t sensorLevelOne;


    int8_t temperature;
    int8_t minTemperature;
    int8_t currentTemperature;
    int8_t maxTemperature;
    bool heaterOn;


    int valves[3];
    int levelSensor[4];

}tankConfig_t;


tankConfig_t tankConfig;


void startup(int maxTemp, int minTemp, int waterSensorHighest, int waterSensorLowest);
void startProgram();
void updateConsole();
void setHeaterStatus(bool state);
void setValveStatus(int id, int state);
```

```
95  double getPressure();
96  int getWaterSensorStatus(int sensor);
97  void setHeaterStatus(bool state);
98  void valueTests(int maxTemp, int minTemp, int waterSensorHighest, int waterSensorLowest);
99  void updateAudio(void);
100 
101 #endif
```

tempSensor.c

Note: The Temperature sensor files are meant to measure the temperature of the water within the tank. If the water gets too hot, the heater will be deactivated. If the water is too cold, the heater will be activated.

```c
/**
 * @file tempSensor.c
 *
 * @brief This file defines the functions utilized for the temperature sensor
 *
 * @author Miles Osborne, Jesse Slager, Daniel Carter
 *
 */

#include "tempSensor.h"
#include "tankSim.h"


// Manages temp Operation
void temperatureSetup()
{
	taskSpawn("tTemperatureControl", 90, 0x100, 2000, (FUNCPTR) initializeTemperatureWatchdog, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
}


void initializeTemperatureWatchdog()
{
	while(1)
	{
		heaterToggler();
		temperatureSensor();
		taskDelay(sysClkRateGet() * 1);
	}
}

void heaterToggler()
{
	if(tankConfig.currentTemperature >= tankConfig.maxTemperature)
	{
		setHeaterStatus(false);
	}
	else if(tankConfig.currentTemperature <= tankConfig.minTemperature)
	{
		setHeaterStatus(true);
	}
}

void temperatureSensor()
{
	if(tankConfig.heaterOn == true)
	{
		tankConfig.currentTemperature += WATERHEATINGRATE;
		tankConfig.currentTemperature += WATERHEATINGRATE;
	  }
	  else if(tankConfig.heaterOn == false)
	  {
		tankConfig.currentTemperature -= WATERHEATINGRATE;
	  }
}
```

tempSensor.h

```
1   /** @file tempSensor.h
2    *
3    * @brief
4    *
5    * @author Miles Osborne, Jesse Slager
6    *
7    */
8
9   #ifndef __TEMP_SENSOR_H
10  #define __TEMP_SENSOR_H
11
12  #include <vxWorks.h>
13  #include <stdio.h>
14  #include <taskLib.h>
15  #include <sysLib.h>
16  #include <msgQLib.h>
17  #include <timers.h>
18  #include <time.h>
19  #include <stdint.h>
20
21  void temperatureSetup();
22
23  void initializeTemperatureWatchdog();
24
25  void heaterToggler();
26
27  void temperatureSensor();
28
29
30  #endif
```

valveControl.c

Note: The purpose of the valve control files are to control the two inlet valves and the one outlet valve. It takes messages received by both the pressure sensors and the water level sensors to determine whether or not it is necessary to manipulate the valves to either increase the waterlevel or to decrease it. It also never allows the waterlevel to never stagnate/

```c
/** @file valveControl.c
 *
 *
 *  @author Daniel Carter, Zack Hagerty
 *
 *
 */

#include "valvecontrol.h"
#include "tankSim.h"
#include <wdLib.h>

#define OPENVALVE 1
#define CLOSEDVALVE 0

//Controls Valve Operation
void controlValves(){

    if(getWaterSensorStatus(1) == INACTIVE){

        setValveStatus(INPUT_VALVE_2_ID, OPENVALVE);
        setValveStatus(INPUT_VALVE_1_ID, OPENVALVE);
        setValveStatus(OUTPUT_VALVE_ID, CLOSEDVALVE);
        taskDelay(120);
    }

    if(getWaterSensorStatus(4) ==  ACTIVE) {

        setValveStatus(INPUT_VALVE_1_ID, CLOSEDVALVE);
        setValveStatus(INPUT_VALVE_2_ID, CLOSEDVALVE);
        setValveStatus(OUTPUT_VALVE_ID, OPENVALVE);
        taskDelay(120);
    }

    if(getPressure() >= HIGH_PRES_THRESHOLD)
    {

        setValveStatus(INPUT_VALVE_1_ID, CLOSEDVALVE);
        setValveStatus(INPUT_VALVE_2_ID, CLOSEDVALVE);
        setValveStatus(OUTPUT_VALVE_ID, OPENVALVE);
    }

    if(getPressure() <= LOW_PRES_THRESHOLD)
       {
                setValveStatus(INPUT_VALVE_2_ID, OPENVALVE);
                setValveStatus(INPUT_VALVE_1_ID, OPENVALVE);
                setValveStatus(OUTPUT_VALVE_ID, CLOSEDVALVE);
```

```
48          }
49
50  }
51
52
53  void initializeValveWatchdog()
54  {
55      while(1){
56          controlValves();
57                  taskDelay(sysClkRateGet() * 1);
58      }
59  }
60
61  void valveSetup()
62  {
63     taskSpawn("tValveControl",90, 0x100, 2000, (FUNCPTR) initializeValveWatchdog, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
64  }
```

valveControl.h

```
1  //author = zack
2
3  #include <vxWorks.h>
4  #include <time.h>
5  #include <taskLib.h>
6  #include <sysLib.h>
7
8
9  WDOG_ID wd_valve;
10
11 void controlValves();
12
13 void valveSetup();
14
15 void initializeValveWatchdog();
16
17
```

waterLevelSensor.c

Note: The water level sensors are for measuring the water level within the tank. There are a total of 4 sensors within the tank that are evenly spaced out.The water level sensor status is sent to the tank simulator which will then send the status to valve control to determine if the water level will need to increase or decrease based on the current waterlevel

```c
/** @file waterLevelSensor.c
 *
 *
 * @author Miles Osborne / Daniel Carter
 *
 *
 *
 *
 */
#include "waterLevelSensor.h"
#include "valveControl.h"


void waterLevelSetup()
{
    taskSpawn("tWaterControl",90, 0x100, 2000, (FUNCPTR) initializeWaterLevels, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
}



void initializeWaterLevels()
{
    while(1){
        updateWaterLevel();
        taskDelay(sysClkRateGet() / 1 );
    }
}


void updateWaterLevel()
{
    if(tankConfig.valves[2] == INACTIVE)
    {
        tankConfig.waterLevel += FLOWRATE;
        tankConfig.waterRising = true;
    }
    else if(tankConfig.valves[2] == ACTIVE)
    {
        tankConfig.valves[0] = 0;
        tankConfig.valves[1] = 0;
        tankConfig.waterLevel -= FLOWRATE;
        tankConfig.waterRising = false;
    }
}

```

waterLevelSensor.h

```
1   /** @file waterLevelSensor.h
2    *
3    *
4    *  @author Miles Osborne / Daniel Carter
5    *
6    *
7    *
8    */
9
10  #include <vxWorks.h>
11  #include <stdio.h>
12  #include <taskLib.h>
13  #include <sysLib.h>
14  #include <msgQLib.h>
15  #include <timers.h>
16  #include <time.h>
17  #include <stdint.h>
18  #include "tankSim.h"
19
20
21  void updateWaterLevel();
22
23  void waterLevelSetup();
24
25  void initializeWaterLevels();
```

Audio Functionality

The audio interface consists of a wrapper library of the GPIO drivers to interface with the Adafruit sound module. The purpose of module is to control what track plays and when based on the current status of the system and the associated GPIO pin of the SABRE Lite development boards.

audio_interface.c

```
1   #include "audio_interface.h"
2
3   void audio_init(void){
4       gpio_clock_en();
5       enable_GPIO09();
6       enable_GPIO16();
7       enable_GPIO5();
8       enable_GPIO18();
9       enable_GPIO19();
10
11      gpio09_write(1);
12      gpio16_write(1);
13      gpio05_write(1);
14      gpio18_write(1);
15      gpio19_write(1);
16  }
```

```
18  void play_track(int track_num){
19      switch(track_num){
20          case TRACK1:
21          gpio09_write(0);
22          taskDelay(10);
23          gpio09_write(1);
24          break;
25
26          case TRACK2:
27          gpio16_write(0);
28          taskDelay(10);
29          gpio16_write(1);
30          break;
31
32          case TRACK3:
33          gpio05_write(0);
34          taskDelay(10);
35          gpio05_write(1);
36          break;
37
38          case TRACK4:
39          gpio18_write(0);
40          taskDelay(10);
41          gpio18_write(1);
42          break;
43
44          case TRACK5:
45          gpio19_write(0);
46          taskDelay(10);
47          gpio19_write(1);
48          break;
49
50          default:
51          printf("Track not found!\n!");
52          break;
53      }
54  }
```

audio_interface.h

```
#ifndef __AUDIO_INTERFACE_
#define __AUDIO_INTERFACE_

#include "GPIO.h"

#define TRACK1  1
#define TRACK2  2
#define TRACK3  3
#define TRACK4  4
#define TRACK5  5

void audio_init(void);
void play_track(int track_num);

#endif
```

NXP I.MX6 GPIO Driver

The following files comprise a partial GPIO driver for the NXP I.MX6 Quad processor. This module was developed only with official support for the output mode of the GPIO pins. This module was designed before interfacing with VxWorks hence some functions were modified or unused in the final release but were retained for understanding the platform, debugging, and testing.

NOTE: The following files are not included in this appendix item: iomux_define..h and iomux_register.c. These files were provided by the NXP I.MX6 Platform SDK 1.1 and only used as a tool to help us develop the system. To obtain a copy of the SDK please refer to the NXP i.MX6 Platform SDK 1.1 Release page.

CCM.c

```
#include "CCM.h"

void enable_IOMUXC(void){
        int enable_iomux = (1 << IOMUX_IPT_CLK_IO_EN1) | (1 << IOMUX_IPT_CLK_IO_EN2);
        *CCGR2 |= enable_iomux;
}
```

CCM.h

```
1   #ifndef __CCM__H
2   #define __CCM__H
3
4   //#include "CCM_reg.h"
5   #include <stdint.h>
6
7   #define CCGR2 ((int32_t*) (0x02e0C4070))
8   #define IOMUX_IPT_CLK_IO_EN1 14
9   #define IOMUX_IPT_CLK_IO_EN2 15
10
11  void enable_IOMUXC(void);
12
13  #endif
```

GPIO_reg.h

```
1   #ifndef __NXP_IMX6_GPIO_REGS__
2   #define __NXP_IMX6_GPIO_REGS__
3
4   #include <stdint.h>
5
6   /**GPIOx base addresses**/
7   #define GPIO1_BASE 0x0209C000
8   #define GPIO2_BASE 0x020A0000
9   #define GPIO3_BASE 0x020A4000
10  #define GPIO4_BASE 0x020A8000
11  #define GPIO5_BASE 0x020AC000
12  #define GPIO6_BASE 0x020B0000
13  #define GPIO7_BASE 0x020B4000
14
15  /**GPIOx peripheral adddress offsets**/
16  #define GPIO_DR_OFFSET 0
17  #define GPIO_GDIR_OFFSET 0x4
18  #define GPIO_PSR_OFFSET 0x8
19  #define GPIO_ICR1_OFFSET 0xC
20  #define GPIO_ICR2_OFFSET 0x10
21  #define GPIO_IMR_OFFSET 0x14
22  #define GPIO_ISR_OFFSET 0x18
23  #define GPIO_EDGE_SEL_OFFSET 0x1C
```

```
25  /**GPIO1 peripherals**/
26  #define GPIO1_DR        ((int32_t *) (GPIO1_BASE+GPIO_DR_OFFSET))
27  #define GPIO1_GDIR      ((int32_t *) (GPIO1_BASE+GPIO_GDIR_OFFSET))
28  #define GPIO1_PSR       ((int32_t *) (GPIO1_BASE+GPIO_PSR_OFFSET))
29  #define GPIO1_ICR       ((int32_t *) (GPIO1_BASE+GPIO_ICR1_OFFSET))
30  #define GPIO1_ICR2      ((int32_t *) (GPIO1_BASE+GPIO_ICR2_OFFSET))
31  #define GPIO1_IMR       ((int32_t *) (GPIO1_BASE+GPIO_IMR_OFFSET))
32  #define GPIO1_ISR       ((int32_t *) (GPIO1_BASE+GPIO_ISR_OFFSET))
33  #define GPIO1_EDGE_SEL  ((int32_t *) (GPIO1_BASE+GPIO_EDGE_SEL_OFFSET))
34
35  /**GPIO2 peripherals**/
36  #define GPIO2_DR        ((int32_t *) (GPIO2_BASE+GPIO_DR_OFFSET))
37  #define GPIO2_GDIR      ((int32_t *) (GPIO2_BASE+GPIO_GDIR_OFFSET))
38  #define GPIO2_PSR       ((int32_t *) (GPIO2_BASE+GPIO_PSR_OFFSET))
39  #define GPIO2_ICR       ((int32_t *) (GPIO2_BASE+GPIO_ICR1_OFFSET))
40  #define GPIO2_ICR2      ((int32_t *) (GPIO2_BASE+GPIO_ICR2_OFFSET))
41  #define GPIO2_IMR       ((int32_t *) (GPIO2_BASE+GPIO_IMR_OFFSET))
42  #define GPIO2_ISR       ((int32_t *) (GPIO2_BASE+GPIO_ISR_OFFSET))
43  #define GPIO2_EDGE_SEL  ((int32_t *) (GPIO2_BASE+GPIO_EDGE_SEL_OFFSET))


45  /**GPIO3 peripherals**/
46  #define GPIO3_DR        ((int32_t *) (GPIO3_BASE+GPIO_DR_OFFSET))
47  #define GPIO3_GDIR      ((int32_t *) (GPIO3_BASE+GPIO_GDIR_OFFSET))
48  #define GPIO3_PSR       ((int32_t *) (GPIO3_BASE+GPIO_PSR_OFFSET))
49  #define GPIO3_ICR       ((int32_t *) (GPIO3_BASE+GPIO_ICR1_OFFSET))
50  #define GPIO3_ICR2      ((int32_t *) (GPIO3_BASE+GPIO_ICR2_OFFSET))
51  #define GPIO3_IMR       ((int32_t *) (GPIO3_BASE+GPIO_IMR_OFFSET))
52  #define GPIO3_ISR       ((int32_t *) (GPIO3_BASE+GPIO_ISR_OFFSET))
53  #define GPIO3_EDGE_SEL  ((int32_t *) (GPIO3_BASE+GPIO_EDGE_SEL_OFFSET))
54
55  /**GPIO4 peripherals**/
56  #define GPIO4_DR        ((int32_t *) (GPIO4_BASE+GPIO_DR_OFFSET))
57  #define GPIO4_GDIR      ((int32_t *) (GPIO4_BASE+GPIO_GDIR_OFFSET))
58  #define GPIO4_PSR       ((int32_t *) (GPIO4_BASE+GPIO_PSR_OFFSET))
59  #define GPIO4_ICR       ((int32_t *) (GPIO4_BASE+GPIO_ICR1_OFFSET))
60  #define GPIO4_ICR2      ((int32_t *) (GPIO4_BASE+GPIO_ICR2_OFFSET))
61  #define GPIO4_IMR       ((int32_t *) (GPIO4_BASE+GPIO_IMR_OFFSET))
62  #define GPIO4_ISR       ((int32_t *) (GPIO4_BASE+GPIO_ISR_OFFSET))
63  #define GPIO4_EDGE_SEL  ((int32_t *) (GPIO4_BASE+GPIO_EDGE_SEL_OFFSET))
```

```
65  /**GPIO5 peripherals**/
66  #define GPIO5_DR        ((int32_t *) (GPIO5_BASE+GPIO_DR_OFFSET))
67  #define GPIO5_GDIR      ((int32_t *) (GPIO5_BASE+GPIO_GDIR_OFFSET))
68  #define GPIO5_PSR       ((int32_t *) (GPIO5_BASE+GPIO_PSR_OFFSET))
69  #define GPIO5_ICR       ((int32_t *) (GPIO5_BASE+GPIO_ICR1_OFFSET))
70  #define GPIO5_ICR2      ((int32_t *) (GPIO5_BASE+GPIO_ICR2_OFFSET))
71  #define GPIO5_IMR       ((int32_t *) (GPIO5_BASE+GPIO_IMR_OFFSET))
72  #define GPIO5_ISR       ((int32_t *) (GPIO5_BASE+GPIO_ISR_OFFSET))
73  #define GPIO5_EDGE_SEL  ((int32_t *) (GPIO5_BASE+GPIO_EDGE_SEL_OFFSET))
74
75  /**GPIO6 peripherals**/
76  #define GPIO6_DR        ((int32_t *) (GPIO6_BASE+GPIO_DR_OFFSET))
77  #define GPIO6_GDIR      ((int32_t *) (GPIO6_BASE+GPIO_GDIR_OFFSET))
78  #define GPIO6_PSR       ((int32_t *) (GPIO6_BASE+GPIO_PSR_OFFSET))
79  #define GPIO6_ICR       ((int32_t *) (GPIO6_BASE+GPIO_ICR1_OFFSET))
80  #define GPIO6_ICR2      ((int32_t *) (GPIO6_BASE+GPIO_ICR2_OFFSET))
81  #define GPIO6_IMR       ((int32_t *) (GPIO6_BASE+GPIO_IMR_OFFSET))
82  #define GPIO6_ISR       ((int32_t *) (GPIO6_BASE+GPIO_ISR_OFFSET))
83  #define GPIO6_EDGE_SEL  ((int32_t *) (GPIO6_BASE+GPIO_EDGE_SEL_OFFSET))

85  /**GPIO7 peripherals**/
86  #define GPIO7_DR        ((int32_t *) (GPIO7_BASE+GPIO_DR_OFFSET))
87  #define GPIO7_GDIR      ((int32_t *) (GPIO7_BASE+GPIO_GDIR_OFFSET))
88  #define GPIO7_PSR       ((int32_t *) (GPIO7_BASE+GPIO_PSR_OFFSET))
89  #define GPIO7_ICR       ((int32_t *) (GPIO7_BASE+GPIO_ICR1_OFFSET))
90  #define GPIO7_ICR2      ((int32_t *) (GPIO7_BASE+GPIO_ICR2_OFFSET))
91  #define GPIO7_IMR       ((int32_t *) (GPIO7_BASE+GPIO_IMR_OFFSET))
92  #define GPIO7_ISR       ((int32_t *) (GPIO7_BASE+GPIO_ISR_OFFSET))
93  #define GPIO7_EDGE_SEL  ((int32_t *) (GPIO7_BASE+GPIO_EDGE_SEL_OFFSET))
94
95  #endif
```

GPIO.c

```c
#include "GPIO.h"

void gpio_init() {
        printf("Starting iomux_configure_gpio_pin\n");
        iomux_configure_gpio_pin(global_gpio.iomux_mux_pad_addr,
                        global_gpio.iomux_ctl_pad_addr);

        switch (global_gpio.mode) {
        case GPIO_MDOE_INPUT:
                gpio_input_init(global_gpio.port, global_gpio.pin_num);
                break;

        case GPIO_MODE_OUTPUT:
                gpio_output_init(global_gpio.port, global_gpio.pin_num);
                break;
        default:
                break;
        }
}

uint8_t gpio_pin_read(gpio_reg_t* gpio_port, uint8_t pin_num) {
        return *gpio_port->dr;
}

void gpio_pin_write(gpio_reg_t* gpio_port, uint8_t pin_num, uint8_t value) {
        if (value == 1) {
                *(gpio_port->dr) |= (1 << pin_num);
        } else {
                *(gpio_port->dr) &= ~(1 << pin_num);
        }
}

void gpio_pin_toggle(gpio_reg_t* gpio_port, uint8_t pin_num) {
        *gpio_port->dr ^= (1 << pin_num);
}
```

```
37  int32_t gpio_get_port_addr(gpio_reg_t* port) {
38          return &port;
39  }
40
41  void gpio_input_init(gpio_reg_t* port, uint8_t pin_num) {
42          *port->gdir |= (1u << pin_num);
43  }
44
45  void gpio_output_init(gpio_reg_t* port, uint8_t pin_num) {
46          *port->gdir &= ~(1u << pin_num);
47  }
48
```

```
49  void gpio_clock_en() {
50          #if 0
51          enable_IOMUXC();
52          printf("Complete enable_IOMUXC\n");
53          iomux_enable_armp_ahb_clk();
54          printf("Completed iomux_enable_armp_ahb_clk()\n");
55          iomux_enable_armp_ipg_clk();
56          printf("Completed iomux_enable_armp_ipg_clk()\n");
57          #endif
58
59          //Enable IOMUXC
60          int *CCM_CCGR2 = (int*)0x20C4070;
61          int enable_iomux = (1 << 14) | (1 << 15);
62          *CCM_CCGR2 |= enable_iomux;
63
64          int* iomuxc_gpr12 = (int*)0x20E0030;
65          //Enable ARMP_IPG_CLK
66          *iomuxc_gpr12 |= ARMP_IPG_CLK_EN;
67
68          //Enable AMP_AHB_CL_EN
69          *iomuxc_gpr12 |= ARMP_AHB_CLK_EN;
70
71  }
```

```
74  /**Dedicated GPIO output enable pins**/
75  void enable_GPIO09(void){
76          int *pad_reg = (int *)IOMUXC_SW_MUX_CTL_PAD_GPIO_9 ;
77          int *ctl_reg = (int *)IOMUXC_SW_PAD_CTL_PAD_GPIO_9;
78          *pad_reg &= ~(0x7);
79          *ctl_reg &= ~(0x7);
80
81          *pad_reg |= (ALT5);
82          *ctl_reg |= (ALT5);
83
84          int *gpio_gdir_reg = GPIO1_GDIR;
85          *gpio_gdir_reg |= (1<<GPIO_PIN_9); //Configure as output
86  }
87
88  /**Dedicated GPIO output enable pins**/
89  void enable_GPIO16(void){
90          int *pad_reg = (int *)IOMUXC_SW_MUX_CTL_PAD_GPIO_16;
91          int *ctl_reg = (int *)IOMUXC_SW_PAD_CTL_PAD_GPIO_16;
92          *pad_reg &= ~(0x7);
93          *ctl_reg &= ~(0x7);
94
95          *pad_reg |= (ALT5);
96          *ctl_reg |= (ALT5);
97
98          int *gpio_gdir_reg = GPIO7_GDIR;
99          *gpio_gdir_reg |= (1<<GPIO_PIN_11); //Configure as output
100 }
```

```
116  void enable_GPIO18(void){
117          int *pad_reg = (int *)IOMUXC_SW_MUX_CTL_PAD_GPIO_18;
118          int *ctl_reg = (int *)IOMUXC_SW_PAD_CTL_PAD_GPIO_18;
119          *pad_reg &= ~(0x7);
120          *ctl_reg &= ~(0x7);
121
122          *pad_reg |= (ALT5);
123          *ctl_reg |= (ALT5);
124
125          int *gpio_gdir_reg = GPIO7_GDIR;
126          *gpio_gdir_reg |= (1<<GPIO_PIN_13); //Configure as output
127  }
128
129  void enable_GPIO19(void){
130          int *pad_reg = (int *)IOMUXC_SW_MUX_CTL_PAD_GPIO_19;
131          int *ctl_reg = (int *)IOMUXC_SW_PAD_CTL_PAD_GPIO_19;
132          *pad_reg &= ~(0x7);
133          *ctl_reg &= ~(0x7);
134
135          *pad_reg |= (ALT5);
136          *ctl_reg |= (ALT5);
137
138          int *gpio_gdir_reg = GPIO4_GDIR;
139          *gpio_gdir_reg |= (1<<GPIO_PIN_5); //Configure as output
140  }
141
142  void gpio09_write(uint8_t value){
143          int *dr = GPIO1_DR;
144          if(value == 1){
145                  *dr |= (1u << GPIO_PIN_9);
146          }else{
147                  *dr &= ~(1u << GPIO_PIN_9);
148          }
149  }
```

```
151  void gpio16_write(uint8_t value){
152          int *dr = GPIO7_DR;
153          if(value == 1){
154                  *dr |= (1u << GPIO_PIN_11);
155          }else{
156                  *dr &= ~(1u << GPIO_PIN_11);
157          }
158  }
159
160  void gpio05_write(uint8_t value){
161          int *dr = GPIO1_DR;
162          if(value == 1){
163                  *dr |= (1u << GPIO_PIN_5);
164          }else{
165                  *dr &= ~(1u << GPIO_PIN_5);
166          }
167  }
168
169  void gpio18_write(uint8_t value){
170          int *dr = GPIO7_DR;
171          if(value == 1){
172                  *dr |= (1u << GPIO_PIN_13);
173          }else{
174                  *dr &= ~(1u << GPIO_PIN_13);
175          }
176  }
177
178  void gpio19_write(uint8_t value){
179          int *dr = GPIO4_DR;
180          if(value == 1){
181                  *dr |= (1u << GPIO_PIN_5);
182          }else{
183                  *dr &= ~(1u << GPIO_PIN_5);
184          }
185  }
```

GPIO.h

```
#ifndef __NXP_IMX6_GPIO
#define __NXP_IMX6_GPIO

#include <stdint.h>
#include "iomux.h"
#include "CCM.h"
#include "GPIO_reg.h"

typedef struct
{
    int32_t *dr;
    int32_t *gdir;
    int32_t *psr;
    int32_t *icr1;
    int32_t *icr2;
    int32_t *imr;
    int32_t *isr;
    int32_t *edge_sel;
} gpio_reg_t;

typedef struct GPIO_settings
{
    gpio_reg_t *port;
    uint8_t pin_num;
    uint8_t mode;
    int32_t iomux_ctl_pad_addr;
    int32_t iomux_mux_pad_addr;
} gpio_usr_config;

/*Macros to access GPIO and peripheral registers*/
#define GPIO1 ((gpio_reg_t *)(GPIO1_BASE))
#define GPIO2 ((gpio_reg_t *)(GPIO2_BASE))
#define GPIO3 ((gpio_reg_t *)(GPIO3_BASE))
#define GPIO4 ((gpio_reg_t *)(GPIO4_BASE))
#define GPIO5 ((gpio_reg_t *)(GPIO5_BASE))
#define GPIO6 ((gpio_reg_t *)(GPIO6_BASE))
#define GPIO7 ((gpio_reg_t *)(GPIO7_BASE))
```

```
41  /**GPIO modes**/
42  #define GPIO_MDOE_INPUT     1
43  #define GPIO_MODE_OUTPUT    0
44
45  /**GPIO pins**/
46  #define GPIO_PIN_0  0
47  #define GPIO_PIN_1  1
48  #define GPIO_PIN_2  2
49  #define GPIO_PIN_3  3
50  #define GPIO_PIN_4  4
51  #define GPIO_PIN_5  5
52  #define GPIO_PIN_6  6
53  #define GPIO_PIN_7  7
54  #define GPIO_PIN_8  8
55  #define GPIO_PIN_9  9
56  #define GPIO_PIN_10 10
57  #define GPIO_PIN_11 11
58  #define GPIO_PIN_12 12
59  #define GPIO_PIN_13 13
60  #define GPIO_PIN_14 14
61  #define GPIO_PIN_15 15
62  #define GPIO_PIN_16 16
63  #define GPIO_PIN_17 17
64  #define GPIO_PIN_18 18
65  #define GPIO_PIN_19 19
66  #define GPIO_PIN_20 20
67  #define GPIO_PIN_21 21
68  #define GPIO_PIN_22 22
69  #define GPIO_PIN_23 23
70  #define GPIO_PIN_24 24
71  #define GPIO_PIN_25 25
72  #define GPIO_PIN_26 26
73  #define GPIO_PIN_27 27
74  #define GPIO_PIN_28 28
75  #define GPIO_PIN_29 29
76  #define GPIO_PIN_30 30
77  #define GPIO_PIN_31 31
78
79  gpio_usr_config global_gpio;
```

```
81  uint8_t gpio_pin_read(gpio_reg_t *gpio_port, uint8_t pin_num);
82
83  void gpio_pin_write(gpio_reg_t *gpio_port, uint8_t pin_num, uint8_t value);
84
85  void gpio_pin_toggle(gpio_reg_t *gpio_port, uint8_t pin_num);
86
87  int32_t gpio_get_port_addr(gpio_reg_t* port);
88
89  void gpio_input_init(gpio_reg_t *port, uint8_t pin_num);
90
91  void gpio_output_init(gpio_reg_t *port, uint8_t pin_num);
92
93  void gpio_clock_en();
94
95  #endif
```

iomux.c

```
1   #include "iomux.h"
2
3   void iomux_enable_armp_ipg_clk(){
4       int32_t* iomuxc_gpr12 = (int32_t *)IOMUXC_GPR12;
5       *iomuxc_gpr12 |= ARMP_IPG_CLK_EN;
6   }
7
8   void iomux_enable_armp_ahb_clk(){
9       int32_t* iomuxc_gpr12 = (int32_t *)IOMUXC_GPR12;
10      *iomuxc_gpr12 |= ARMP_AHB_CLK_EN;
11  }
12
13  void iomux_configure_gpio_pin(int32_t mux_pad_address, int32_t ctl_pad_address){
14      printf("Declaring mux_pad and ctl_pad address\n");
15      int32_t *pad_reg = (int32_t *)mux_pad_address;
16          int32_t *ctl_reg = (int32_t *)ctl_pad_address;
17      printf("Finished declaring mux_pad and ctl_pad address\n");
18
19      printf("Storing value back at pad_reg and ctl_reg\n");
20      *pad_reg |= ALT5;
21          *ctl_reg |= ALT5;
22  }
```

iomux.h

```
1   #ifndef __IOMUX__H
2   #define __IOMUX__H
3
4   #include "iomux_register.h"
5   #include "iomux_define.h"
6   #include <stdint.h>
7
8   #define ARMP_IPG_CLK_EN (1 << 27)
9   #define ARMP_AHB_CLK_EN (1 << 26)
10
11  void iomux_enable_armp_ipg_clk();
12  void iomux_enable_armp_ahb_clk();
13  void iomux_configure_gpio_pin(int32_t mux_pad_address, int32_t ctl_pad_address);
14
15  #endif
```

APPENDIX B

Test Cases

Test Case	**Precondition**	**Steps**	**Expected Result**	**Pass/ Fail**
TC. 1 When pressure is at or above max level, the system opens the outlet valve.	VxWorks is running and the project has been downloaded to the board.	1.)User inputs the following command with desired parameters as follows: startup(maxTemp, minTemp, waterSensorHighest, waterSensorLowest)	The outlet valve will be open, and both inlet valves will be closed, this will allow the water to drain, which will decrease water level which will also allow the pressure to decrease as a result	Pass
TC. 2 When water level sensor four has been reached, the system opens the outlet valve.	VxWorks is running and the project has been downloaded to the board.	1.)User will input the following command with desired parameters as follows: startup(maxTemp, minTemp, waterSensorHighest, waterSensorLowest	The outlet valve will be open, and both inlet valves will be closed, which will allow the water to drain, thus decreasing water level.	Pass
TC. 3 When water level sensor one has been reached, the system opens the inlet valves.	VxWorks is running and the project has been downloaded to the board.	1.)User will input the following command with desired parameters as follows: startup(maxTemp, minTemp, waterSensorHighest, waterSensorLowest	The outlet valve will be closed and both inlet valves will be opened, this will allow water into the system, thus allowing the water level to rise	Pass
TC. 4 When temperature is at or above 87 degrees celsius, temperature increase rate multiplies by three.	VxWorks is running and the project has been downloaded to the board.	1.)User will input the following command with desired parameters as follows: startup(maxTemp, minTemp, waterSensorHighest, waterSensorLowest	The pressure will increase at a rate of 7.14 psi per second	Pass
TC. 5 When temperature is at or above maximum level, the heater turns off	VxWorks is running and the project has been downloaded to the board.	1.)User will input the following command with desired parameters as follows: startup(maxTemp, minTemp, waterSensorHighest, waterSensorLowest)	The water heater will be turned off, which will allow the water to cool back down.	Pass
TC. 6	VxWorks is	1.)User will input the	The pressure will increase at a rate of	Pass

When the temperature is below 87 Degrees Celsius, pressure increase rate is 2.38 psi/second.	running and the project has been downloaded to the board.	following command with desired parameters as follows: startup(maxTemp, minTemp, waterSensorHighest, waterSensorLowest)	2.38 psi per second.	
TC. 7 When the temperature is at or below minimum level, heater turns on.	VxWorks is running and the project has been downloaded to the board.	1.)User will input the following command with desired parameters as follows: startup(maxTemp, minTemp, waterSensorHighest, waterSensorLowest)	The water heater will turn back on, thus increasing the water temperature	Pass
TC. 8 When user-defined maximum temperature that was input is below the minimum temperature input, system throws exception	VxWorks is running and the project has been downloaded to the board.	1.)User will input the following command with desired parameters as follows: startup(maxTemp, minTemp, waterSensorHighest, waterSensorLowest) 2.)Ensure that maxTemp is less than minTemp.	The simulation will not launch as the parameters that were input for the max temperature was invalid	Pass
TC. 9 When user-defined minimum temperature that was input is above the maximum temperature input, system throws exception	VxWorks is running and the project has been downloaded to the board.	1.)User will input the following command with desired parameters as follows: startup(maxTemp, minTemp, waterSensorHighest, waterSensorLowest) 2.) Ensure that the specified minTemp value is higher than maxTemp	The simulation will not launch as the specified minTemp value is invalid	Pass
TC. 10 When user-defined highest water sensor is above 50L, system throws exception.	VxWorks is running and the project has been downloaded to the board.	1.)User will input the following command with desired parameters as follows: startup(maxTemp, minTemp, waterSensorHighest, waterSensorLowest) 2.) Ensure that the specified waterSensorHighest value is above 50	The simulation will not launch as the specified waterSensorHighest value is invalid	Pass
TC. 11 When user-defined lowest water sensor	VxWorks is running and the project	1.)User will input the following command with desired parameters as	The simulation will not launch as the specified waterSensorLowest value is invalid	Pass

is below 0L, system throws an exception	has been downloaded to the board.	follows: startup(maxTemp, minTemp, waterSensorHighest, waterSensorLowest) 2.) Ensure that the specified waterSensorHighest value is above 0		
TC 12. When user-defined lowest water sensor is above 50L, system throws an exception	VxWorks is running and the project has been downloaded to the board.	1.)User will input the following command with desired parameters as follows: startup(maxTemp, minTemp, waterSensorHighest, waterSensorLowest) 2.) Ensure that the specified waterSensorHighest value is above 50	The simulation will not launch as the specified waterSensorLowest value is invalid	Pass
TC. 13 When user-defined lowest water sensor is above highest water system, system throws an exception	VxWorks is running and the project has been downloaded to the board.	1.)User will input the following command with desired parameters as follows: startup(maxTemp, minTemp, waterSensorHighest, waterSensorLowest) 2.) Ensure that the specified waterSensorLowest value is above waterSensorHighest	The simulation will not launch as the specified waterSensorLowest value is invalid	Pass
TC. 14 When user-defined highest water sensor is below 0L, system throws exception.	VxWorks is running and the project has been downloaded to the board.	1.)User will input the following command with desired parameters as follows: startup(maxTemp, minTemp, waterSensorHighest, waterSensorLowest) 2.) Ensure that the specified waterSensorHighest value is below	The simulation will not launch as the specified waterSensorHighest value is invalid	Pass