Project Final Report:
Bread Proofing Box

Team 3

Caleb Williamson Ellias Palcu Aaron Young

December 1, 2016

Contents

1 Introduction

Bread making is as much an art as a science. There are many factors at work to determine how the final loaf will turn out. These variables include the activity of the starter, the water to-flour-ratio, and the time and temperatures for the various fermenting, proofing, and rising stages. The goal of this project is to add more science to the art of bread fermenting by creating an advanced, FPGA controlled, bread proofing box. This bread proofing box will allow precise control over the time, temperature, and humidity during the fermenting and rising stages. This fine control over the fermenting conditions will allow for experiments to be setup to determine the effects that temperature and humidity have on the final loaf of bread.

Upon accomplishing the basic functionality of allowing precise control and conclusive experimentation, several augmentations will be included in the final design. These augmentations include extending the availability of the data by displaying current and prior data though a web interface at real-time, minute resolution. By experimenting with different temperature and humidity conditions, the bread can take on many different characteristics. Through storing these characteristics in a database, users can view previous proofing runs and any min/max and average conditions experienced. Ultimately, in accessing this data, users may extend their control by analytically making statistic conclusions to determine optimal bread proofing conditions. Once the optional conditions are discovered, the box should be able to be used to produce consistently delicious bread.

2 Requirements

All of the requirements are working towards the final goal of creating an experiential bread proofing box that will allow experimenting with the effects temperature, humidity, and time have on bread making.

Primary Goals:

1. Monitor Temperature of Box.
2. Monitor Humidity of Box.
3. Adjust Humidity and Temperature.
4. Timer for rising time.
5. Alarm for rising done.
6. LCD interface for button control.

Stretch Goals:

1. Web interface/app for real-time, remote monitoring and control.
2. Database to store previous proofing stages.
3. Monitoring system to check bread rising via ultrasound.

3 Method

This project can be subdivided into four major aspects: input, control, output, and analysis. The system will use the control logic to make decisions based upon the input provided to it, and those decisions will affect the output of the system over time.

3.1 Component List

The components used to build the bread box are listed in Table 1.

Item	**Specific Part**	**Price**
Humidity and Temperature Sensor	HIH6130	$ 29.95
Ultrasonic Sensor	HC-SR04	$ 3.95
Keypad	PmodKYPD	$ 9.99
Time Keeping	PmodRTCC	$ 8.99
Pin out	PmodCON1	$ 4.99
Voltage level converter	SparkFun Logic Level Converter Bi- Directional	$ 2.95
Relay module	SunFounder2 Channel	$ 6.79
Piezo Buzzer	PS1240	$ 1.50
Humidifier	HealthSmart Tabletop Humidifier[1]	$ 15.00
Light bulb for Heat	125 watt incandescent heat lamp bulb	$ 4.97
Light socket	Bayco 8.5" Clamp Light	$ 6.47
Enclosure walls	Foam Board	$ 5.31
Enclosure Window	18" X 42" Impact Acrylic	$ 11.97
Humidifier Connecting Tube	Foil Dryer Duct	$ 9.97

Table 1: Parts List

3.2 System Input

The system needs to accept programming input directly from the user as well as take sensor input from the sensors. The user must be able to input timing, temperature, and humidity settings using a keypad on the device. The humidity and temperature sensor must also send its input to the FPGA for processing. Finally, an ultrasonic sensor may potentially be used to monitor the rising state of the bread in future work. Figure 1 shows the particular temperature and humidity sensor, the keypad, and the ultrasonic sensor that were selected for this project.

[1]Obtained on clearance do to broken/missing top redirector. Originally $64.31.

Figure 1: Temperature and humidity sensor HIH6130, keypad pmodKYPD, and ultrasonic sensor HC-SR04.

3.3 System Control

The control logic initially processes the inputs from the user and sets up the system environment as required. Throughout the proofing process, this control logic continues to make temperature and humidity decisions based upon the sensor input provided, as well as the initial configuration that was set up by the user. The user input specifies multiple time points and the desired temperature/humidity at those corresponding time points. The controller stores this input and uses this data to maintain the temperature and humidity between the bread rising stages. The LCD screen and web interface are used to display the current stage and time remaining for that phase.

3.4 System Output

There are two types of output from the system. One type of output is an audible alert to let the user know that the proofing process is complete. The alarm will sound to let the user know that the bread is ready for the next step. The second kind of output involves manipulating the temperature and humidity of the enclosure via the heat lamp and the humidifier.

3.5 System Analysis

By developing a web interface, the temperature and humidity progression through time within the bread proofing box can be graphically analyzed. By being able to view temperature and humidity values graphically, users can compare final bread results with their specified set points. Furthermore, in having access to these chart analyses, a better understanding can be developed in how temperature and humidity impact one another. To

accomplish this, data would not only have to be graphed on at a fine-grained resolution, but must be stored to allow for further review, if necessary.

4 Design

4.1 Architecture Design

The architecture contains several key components that are directly related to the methods described above. A high-level block diagram of the system can be found in Figure 2. The main purpose of this project is to control a system based on a pre-configured, programmed schedule of environmental conditions and on sensors that monitor the system. The output are signals that manipulate the system by turning on and off power to a heat lamp and humidifier. In order to streamline the creation of the control logic, a MicroBlaze soft-core processor makes up the central core of the architecture. This soft-core processor is in charge of more advanced input processing as well as communication via ethernet to the web server. The set points will be based on the user's programmed input and the current time. A state machine outside of this soft-core processor is be used to make decisions based on the current humidity and temperature and the set-points programmed by the user, then drives the output devices to reach the desired humidity and temperature. The state machine's set-points are updated via a queue controller which keeps track of the current time and status of the proofing box.

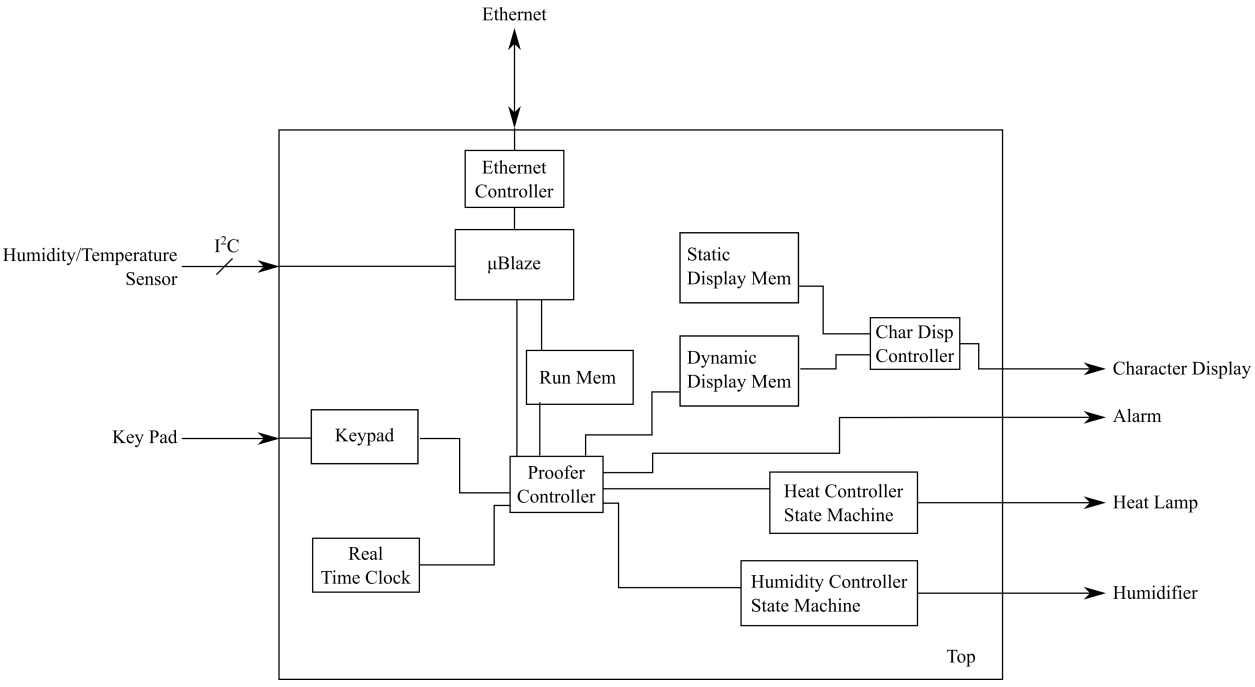

Figure 2: Finalized high level block diagram

4.1.1 Changes from Proposal

Most major changes to the project revolved around the use of the MicroBlaze soft-core processor. In the beginning design stages there was some uncertainty about the viability of creating a successful design based on MicroBlaze due to its apparent complexity. Therefore, much of the work was anticipated to be done via IP blocks outside of the processor, meaning only the control logic would be run on MicroBlaze. The block diagram for this initial architectural design can be seen in Figure 3. However, after the team gained a better understanding of the fundamental workings of the MicroBlaze architecture and encountering some difficulty in acquiring IP cores through OpenCores for both Ethernet and I^2C modules, more tasks were offloaded back onto the soft-core processor.

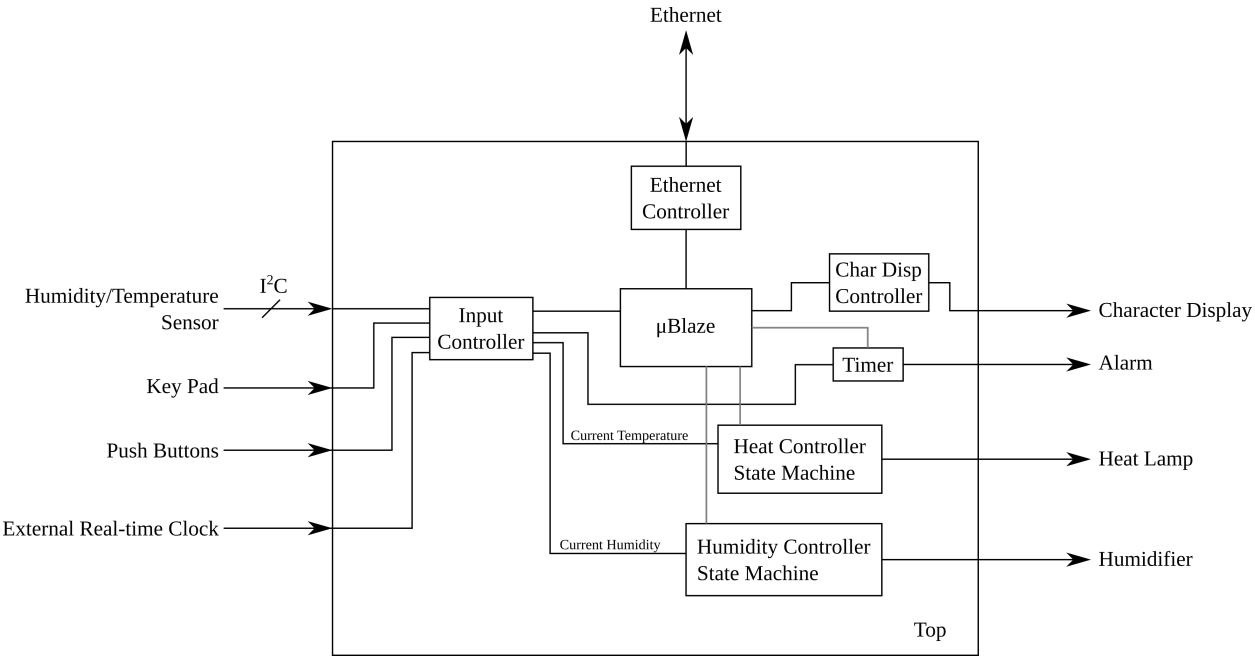

Figure 3: Initial high level block diagram

In particular, utilizing MicroBlaze made available the AXI IIC IP for I^2C communication and the AXI Ethernet Lite IP for basic Ethernet communications along with simple usage examples. To allow for more custom VHDL logic to be written and implemented on the FPGA, the control and decision state machines were migrated away from the soft-core processor. This resulted in a more balanced design that takes into consideration both logic level design flow as well as higher level architectural considerations.

4.2 Behavior Design

The high level block diagram shows the major components of the design and from it, the RTL modules can be inferred. However, it does not show how the system will behave or how the different components will work together as a whole. In order to design the behavior of the bread box, a high level logic diagram was created and is shown as Figure 4. When the bread box is first powered on the text "Bread Box" will appear on the screen. When

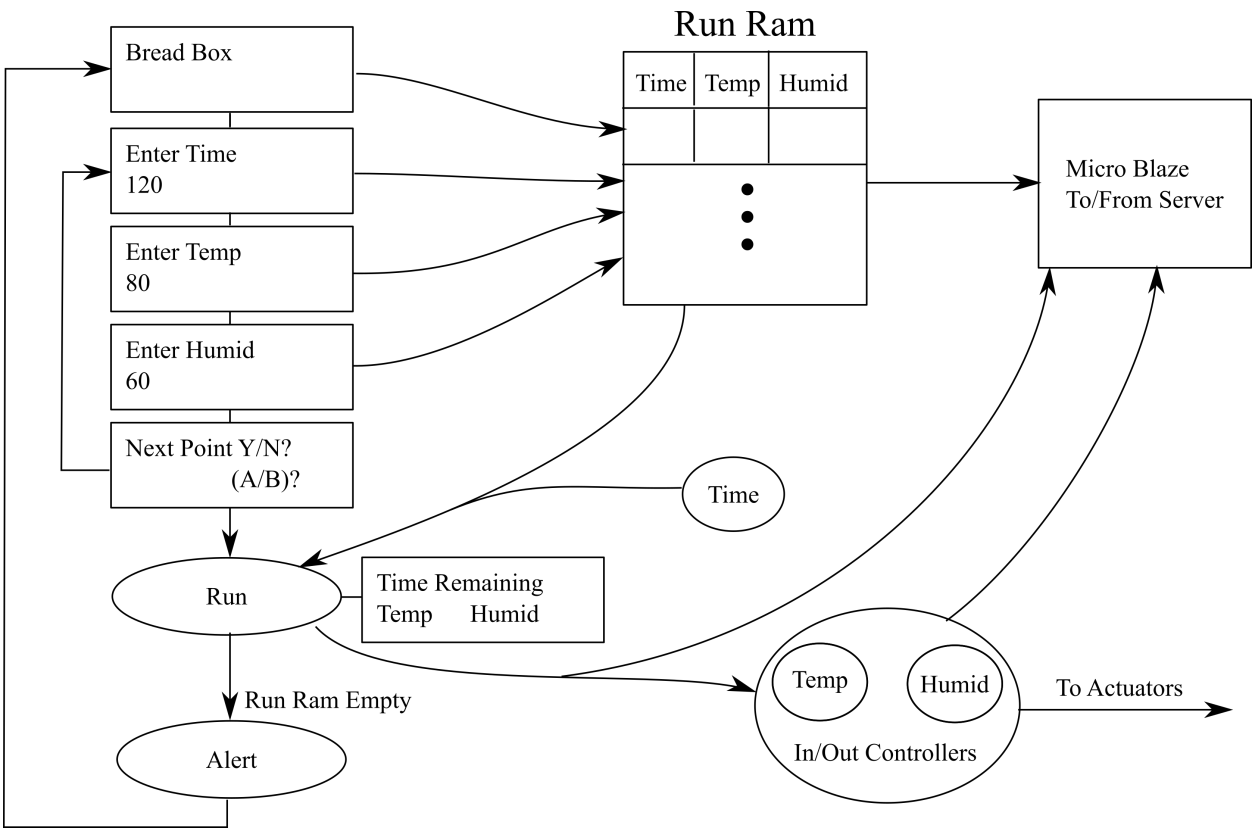

Figure 4: High level logic diagram

any button is pressed the user will be prompted to enter a time for the the stage. Hitting a number key will cause the number to be entered into the system and displayed on the character display. If the wrong number is pressed the 'B' key, or backspace, can be used to remove it one digit at a time. Once a time is entered and the user presses the 'A' key, the user will be prompted to enter the target temperature for the stage. The unit of temperature is degrees Fahrenheit. After the temperature is entered and 'A' is pressed, the user will be prompted to enter the target humidity for the run. Humidity is entered as percent humidity. Once the humidity value is entered and 'A' is pressed, the user will be prompted for the next point. If the user presses 'A' they will be sent back to the enter time page and will be prompted to enter data for the next stage in the run. Each proofing run is built up from multiple smaller stages. Each stage has its own duration and set-points. As the user enters the stages of the run, the points will be stored into the run RAM.

If the user presses 'B' then the first stage of the run will start. The display will show the current state number, the time remaining, the current temperature, and the current humidity. The data for the current stage will be read in from the run RAM. The information about the current run will be sent to the input/output controllers, which will read the current values from the sensors and control the heat lamp and humidifier, so that the set-points for the current stage are reached.

The output control module utilizes a counter that ranges from 0 to 50 for both humidity and temperature to give historical weight to the decision process. For example, if the temperature readings from the sensor have recently been lower than the temperature set point, the weighted counter is closer to 50. Assuming the temperature continues to be lower than the set point, the counter will continue to increment to up until its max weight of 50. Now, if the temperature suddenly shifts to be higher than the set point, the counter reverses its count, slowly shifting its weight towards 0. Once the counter shifts past the threshold of 25 in either direction the corresponding output is toggled. To ensure that this toggling does not occur too quickly and thus degrade or destroy the humidifier and heat lamp components over time, a slow clock is used to update the counters and an even slower clock is used to update the outputs based on these counters.

When the time remaining for the current stage reaches zero the stage will be read in and started. If there are no more stages in the run RAM, the display will show alert and a buzzer will sound. The buzzer will keep sounding until any button is pressed. Once a button is pressed the bread box will return to its initial state and display "Bread Box" once again.

5 Division of Labor

It was important that the work laid out in this document could be divided into easily divided parts. Each of these parts had to be discrete enough to allow members of the team to work on their parts independently, without having to wait on unmet dependencies from another team member. The major components of the design are fairly independent from one another. For example, the state machines that control the temperature and humidity could be written without knowing the specifics of how the sensor data is read in. Likewise the sensor interface could be written independently of the other components. The divisions of labor correspond to each of the major aspects of the project: reading and processing

sensor input, processing and tracking user run programming, developing state machines for decision making and output control, assembling the physical components, and designing and interfacing with the web application. By assigning a single group member to predominantly reside over one or more aspects of the design, each design aspect can be accounted for and properly verified. However, throughout the design some of these areas required more work than the others, so each major part was further broken down into minor components and frequently re-assigned to balance the workload between group members.

6 Design Flow and Verification Strategy

The design flow of this project closely followed the design flow as laid out in class. This document describes the initial idea and describes the process used to develop the architecture. It also includes a behavioral concept of this design and the results of building a RTL design in which the signal interactions of each component are be mapped and better understood. This involved further development of the behavioral code that describes the functionality of the design. The RTL code can be synthesized into a netlist and placed onto the FPGA for manual testing. All throughout the design, each stage was simulated and verified before the next step in the design proceeded.

The ideal approach to simulating and verifying the functionality of this design would be to break the design into submodules and testing each submodule individually throughout every stage of the design. The most vital submodules are the output control module, the proofing state module, both of which fit into the larger bread proofing top module. By creating test benches for each submodule we can verify that expected outputs are properly generated from the various combinations of inputs. Furthermore, because this project has many other submodules besides these, it becomes necessary to combine the testing of some of these submodules to save time. This combination is achieved by developing a single, higher level test bench for testing various submodules of the design at once. For example, a single test bench can be created for testing the functionality of the I^2C controller submodule in relation to the temperature and humidity output controller submodule. By combining modules in this way, the tests can verify that not only do the modules achieve their individual intended functionality, but also effectively function across the system in harmony with one other. Once it is verified that multiple combinations of submodules function as intended, even higher level test benches that encompass more modules can be written and run to ensure the entire system of modules works cohesively.

This test bench model did not fit well when developing the web interface. Instead, a more traditional software engineering test structure was used in which a variety of test cases and edge cases were created to exercise and debug the web application to a reasonable standard of operation. This approach may be viewed as analogous to verifying fireproofing via flamethrower rather than by match, as it is much more difficult to control all of the variables in a live and real-world test environment than in a simulated test bench; however, it was decided constructing a mock server would be in the best interest in progressing the interface's development. By facilitating this server to test potential edge cases by sending dummy data, the integration into the eventual Microblaze server proved seamless.

7 Verification and Testing

7.1 Proofing State Machine

The proofing state machine is one of the main components of the FPGA design. It is in charge of keeping track of the state of the bread box. It also has the important job of reading and writing values to the various memories and providing current run data to all of the other components of the design. Since the proofing state machine is so central to the design, it was verified with the test bench and also incrementally verified using the board. When there were problems with the design, the test bench along with temporary debug values displayed on the LED outputs were used to find the issue.

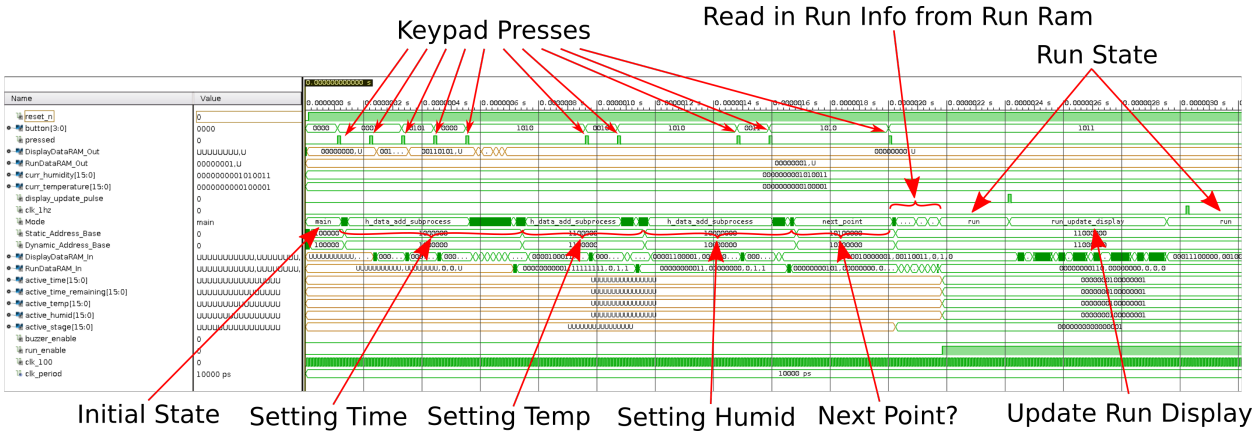

Figure 5: Proofing State Test Bench

Figure 5 shows the test bench used to verify the behavior of the proofing state machine. As the proofing state machine was incrementally built up, the test bench also was incrementally built up to test out the new features of the proofing state machine. The figure shows the final version of the test bench. The test bench was run for the duration of a mock setup run. The transitions of the main state machine and the places where keys were pressed are labeled. This test bench was a great help in verifying that the state machine would function correctly.

Figure 5 shows the waveform for the entire duration of the test bench, because of this, zooming in is necessary to be able to clearly read the state transitions. Let'0s now zoom in on the start of the run to verify this portion of the larger state machine. Figure 6 shows the region from the test bench we will zoom in to focus on. Figure 7 is the zoomed in region.

Figure 7 verifies the start of a bread proofing run once there are no more stages to be entered. At the start of the waveform the Mode is still at `next_point`. In this mode the board is waiting for the user to indicate if another stage in the proofing run will be added. The first event that happens is the pressing of the 'B' button by the user. This is shown in the figure by the arrow that points to the pulse signal and the b on the button line. Pulse is used to indicate when a button press should be processed. Pulse will be high for one clock cycle when a button is first pressed. The button press then causes the mode to transition to `run_setup`. The `run_setup` state is used to initialize the base addresses for

the various RAMs, the `run_sub_counter`, and the `run_stage`. Next the mode is changed to `run_read_time`. In this mode, the active run time is read from the run RAM into the local signal `run_time`. Since the data is stored as shorts, and the RAM reads a byte at a time, a `run_sub_counter` signal is used to set the signals to correctly read in the value. At the end of the `run_read_time` mode, the `run_time_remaining` is set equal to the `run_time`. The next mode is `run_read_temp`, and it reads the current active temperature into the `run_temp` signal just like the time was read in. The following mode is `run_read_humid`, and it reads the current active humidity into the `run_humid` signal just like the time was read in. The next mode is `run`. In the `run` mode, the read in active values are put on the external `active_time`, `active_time_remaining`, `active_temp`, and `active_humid` signals. Additionally, the `run_enable` line is set high to indicate that the run has started. Thus we have now verified the correct behavior of this portion of the proofing state machine. The other portions of the state machines were also likewise verified; however, their in-depth verification is not included in this report.

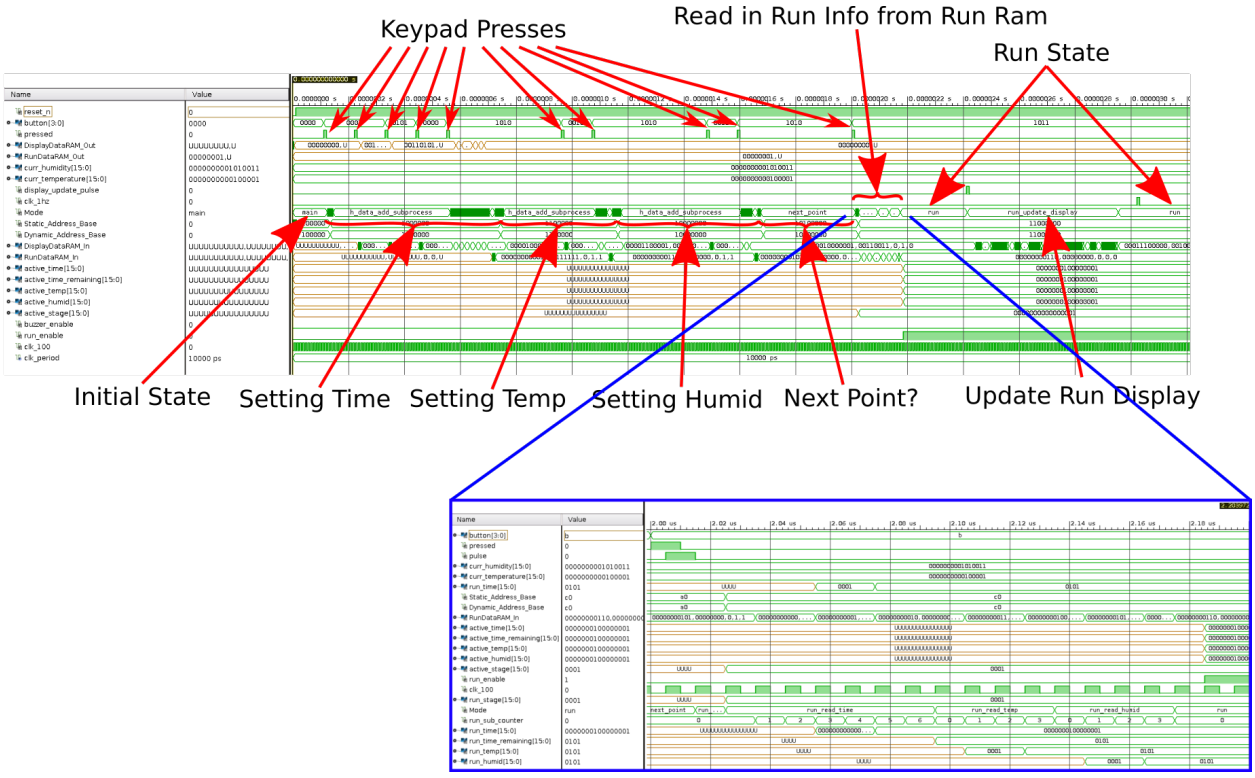

Figure 6: Proofing State Test Bench Zoom Region

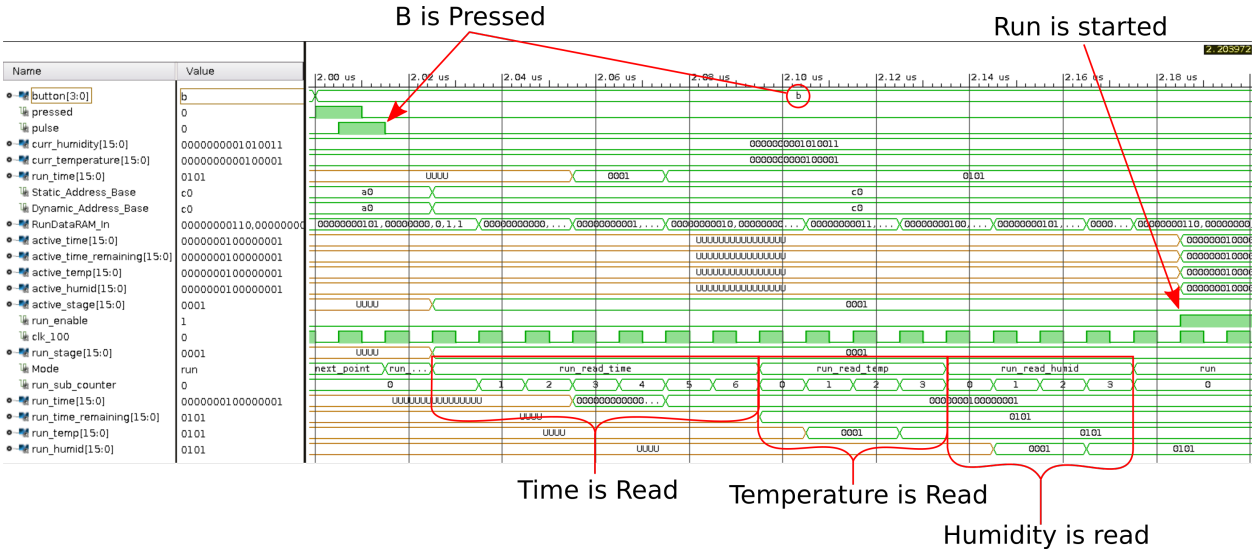

Figure 7: Proofing State Test Bench Zoomed and Labeled

7.2 Output Control Machine

The output control fundamentally operates as a state machine which relies on a set of stored values and a set of values read from the temperature humidity sensor. In order to test the functionality of this state machine, is is necessary to design a test bench that is capable of simulating both preset values and sensor readings on which the state machine will operate. The state machine has predictable outputs based on the inputs provided to it. In reality, this machine is two combined state machines operating independently within the same module. One state machine deals with temperature readings and outputs while the other deals with humidity readings and outputs. The code for these state machines can be found in Appendix A.3. From this code it can be seen that if the set temperature and humidity are higher than the temperature and humidity read from the sensor, then both the humidity and temperature outputs will be high, otherwise they will be low. In the test bench provided in Appendix B.2, the humidity being fed into the control state machine is purposefully set lower than the humidity set point. It is also important to note that the test bench sets the `enable_output` signal high in order to signal for the module that it is allowed to change its temperature and humidity outputs. The results can be seen in Figure 8.

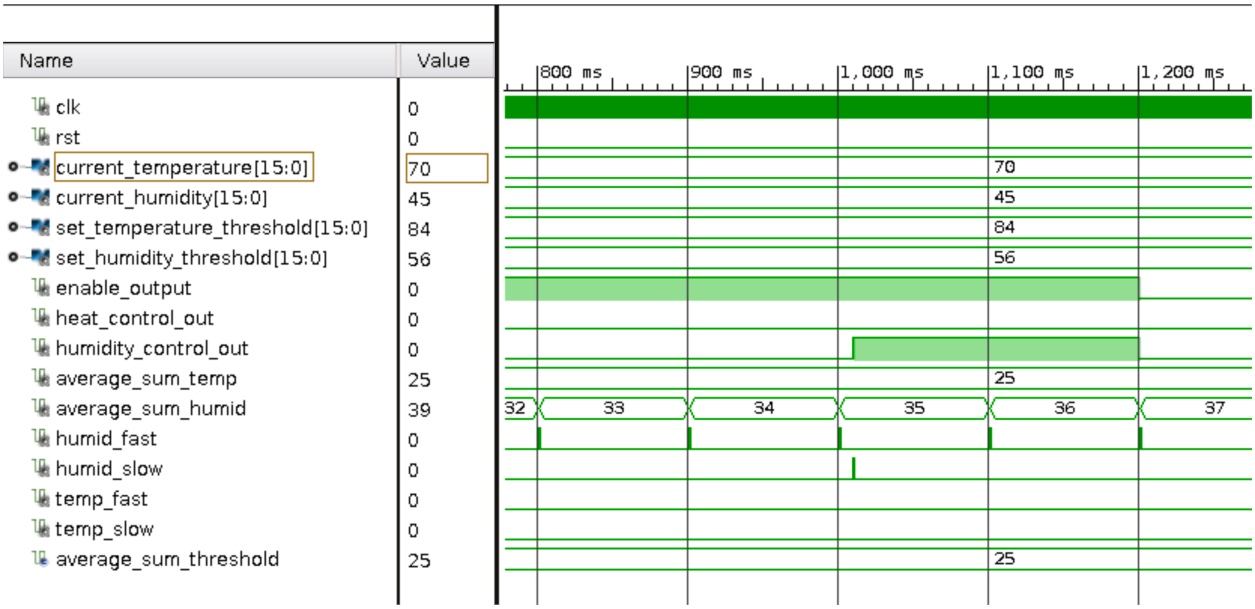

Figure 8: Heat lamp and humidifier output control test bench results.

The important signals to watch in this simulation include the ones that affect the humidity output. It can be seen that every pulse from the fast humidity clock brings about an increment in the humidity counter, causing the counter to go above the "on" threshold of decimal value 25. However, the humidity output does not immediately toggle on as soon as this counter reaches 25. This is expected behavior and is done by design to prevent unnecessary strobing of the heat lamp. Therefore, once the slow humidity clock pulses every 15 seconds, the humidity output is finally updated to logic high, meaning the humidifier should now be on, thus increasing the humidity in the box. This particular simulation has successfully verified the anticipated behavior of the VHDL code.

7.3 Touch Pad Input

The touch pad input component is used to determine which buttons are pressed on the keypad by scanning the keypad column by column. The component outputs the column to power and reads the rows that have pressed keys. The component outputs the decoded output of a single press, a boolean for when there is a key pressed and the bit masks for the pressed buttons when there is more than one button pressed. When there are multiple buttons pressed the decoded output should hold the value of the last button pressed by itself. The test bench emulates the pressing of no buttons, button "9", multiple buttons simultaneously, button "1", and finally back to no presses. The test bench has asserts throughout it that are used to verify correct behavior. Figure 9 shows the output from running the test bench. From the output one can see that there were no failed asserts in the run.

```
Note: Testing No presses
Time: 0 ps
Note: Testing press 9
Time: 10 ms
Note: Testing press multiple
Time: 20 ms
Note: Testing press 1
Time: 30 ms
Note: Testing No presses
Time: 40 ms
```

Figure 9: Output from the VHDL test bench asserts.

Figure 10 shows the waveform of the test bench. Current state is the current step being tested. The corresponding tests are labeled below the waveform. The DecodeOut does indeed correspond to the button pressed; it also holds the previous value when multiple buttons are pressed. The buttons bit field correctly updates to the pressed buttons and the pressed boolean also changes correctly.

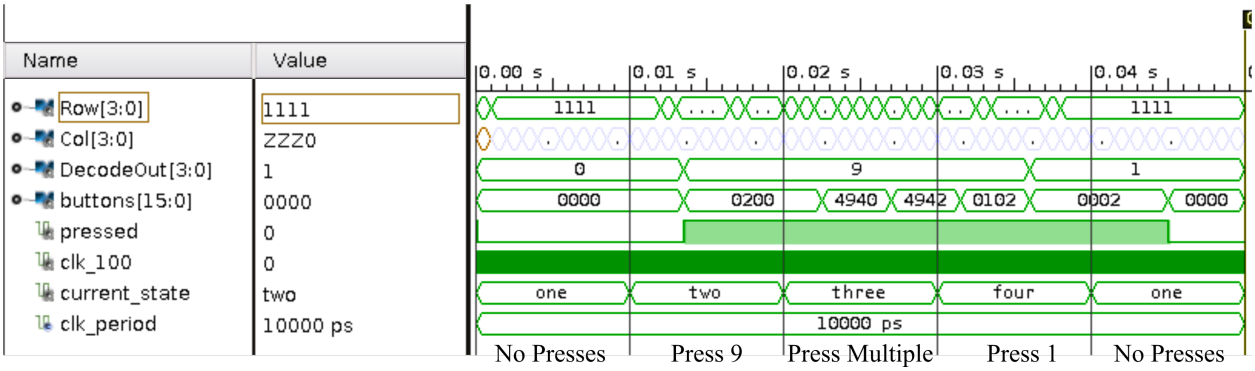

Figure 10: Keypad Test Bench

7.4 DoubleDabble

The double dabble component uses the double dabble algorithm to convert binary numbers to binary-coded decimal numbers. The test bench runs through all the possible input values. The waveform shown in figure 11 shows some abbreviated results from the test. The test shows that the binary input is indeed being correctly mapped to a binary-coded decimal.

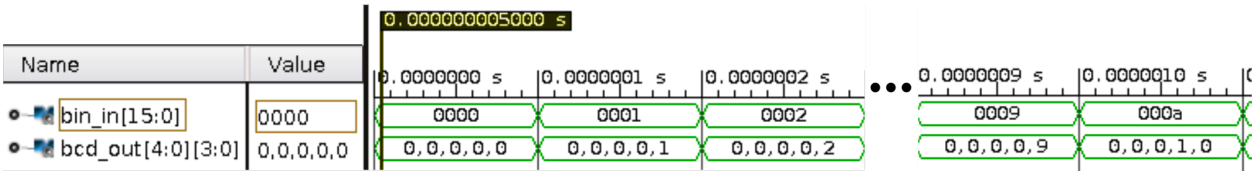

Figure 11: Double Dabble Test Bench

7.5 BCD 2 Bin

The BCD 2 Bin component does the opposite of the double dabble component. It takes a binary-coded decimal number and converts it into binary. The test bench runs through many of the input combinations. Figure 12 shows the abbreviated results from the test bench. The test shows that the binary-coded decimal number is being correctly converted into a binary number.

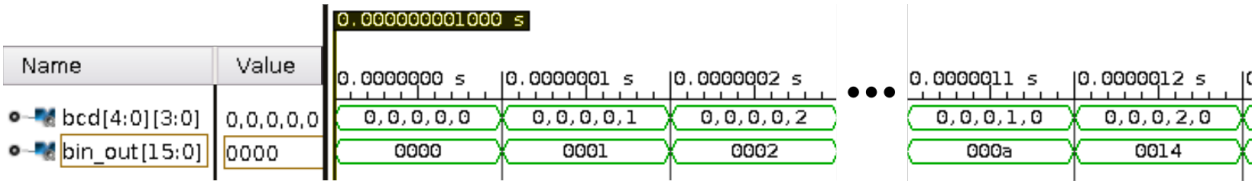

Figure 12: BCD 2 Bin Test Bench

7.6 Buzzer

The buzzer module is in charge of creating a square wave at 20 KHz that will be used to sound the Piezo buzzer. The component has a clock and an enable line for input and generates the 10 KHz wave for output. Figure 13 is used to verify the correct behavior of the module. From the waveform it can be verified that half of the pulse lasts for 0.1 ms which corresponds to a frequency of 20 KHz.

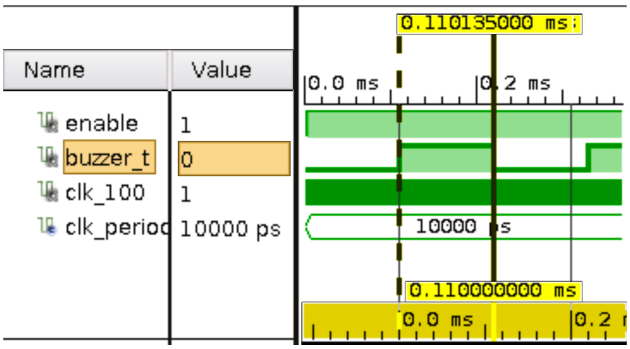

Figure 13: Buzzer Test Bench

7.7 Server-Client Communication

Initially, the primary goal was to set up a server on the FPGA. By utilizing a Microblaze soft processing core, the team was able to modify an echo server for sending a concatenated string containing information related to current temperature, humidity, time remaining, current stage, and stage duration. Because the web interface and Microblaze server communication could not necessarily be test benched using Vivado or ModelSim, other methods of verification had to be utilized. On the client-side, to ensure no time gaps in data, the server is polled every second. This translates into the application sending an acknowledgment to the server indicating that it is ready to receive current readings. On the client-side, output

is displayed indicating successful connection to server and corresponding data retrieval. As aforementioned, the application polls the server every second, and retrieves a concatenated string of temperature, humidity, stage, time remaining, and stage duration values. Figure 14 below details typical connection verification and data retrieval.

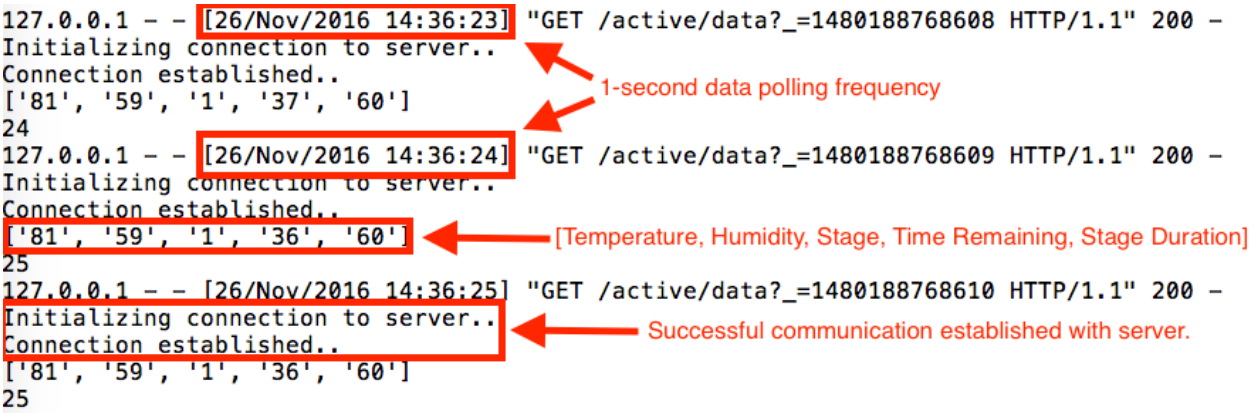

Figure 14: Typical client-side output.

On the server side, upon receiving an acknowledgment from the client that it is prepared to receive measurements, the temperature/humidity is read. Alongside gathering sensor readings, the server also accesses the Run RAM and retrieves information related to current stage, time remaining in corresponding stage, and the duration of the run. After all of the requested data is collected, they are concatenated into a single string and sent to the interface. To ensure the data is properly transferred to the application, the string of readings is displayed on-screen to indicate proper communication and transmission are achieved. Figure 15 below details typical polling and data transmission experienced by the Microblaze server. As can be seen from the same figure, the assigned network parameters are displayed. This indicates that the server setup was successful and it is listening for any potential clients. Alongside displaying the network parameters, the second-by-second sending of data to the web application is also displayed to verify valid readings were taken and data transfer was successful on the server end.

```
------lwIP TCP echo server -------
TCP packets sent to port 6001 will be echoed back
link speed: 100
Board IP: 192.168.1.10
Netmask : 255.255.255.0
Gateway : 192.168.1.1
TCP echo server started @ port 7

Temp reading: 68
Hum reading: 59
Concatenated reading: 68,59,1,2,3

Temp reading: 68
Hum reading: 59
Concatenated reading: 68,59,1,2,3

Temp reading: 68
Hum reading: 59
Concatenated reading: 68,59,1,2,3

Temp reading: 68
Hum reading: 58
Concatenated reading: 68,58,1,2,3
```

Figure 15: Server-side communication.

7.8 System Level Testing

In addition to using test benches to verify the behavior of the bread box, the team also perform high level system tests on the hardware. These high level tests verified the correct behavior of the system and led to useful discoveries.

The first discovery was that bounds were needed on the numbering so that users could not add numbers past what the screen could show and also so that the users can erase more than the beginning of the screen. These bounds were subsequently put into place and now the users can only enter a max of 5 digits for the numbers and they can also no longer backspace past the beginning of the number.

A second discovery was that a run cancel button would be very useful for the run page. If the user wants to cancel a run gracefully without resetting the board, they can now press the 'E' key to cancel the run and go back to the initial page.

8 Results

The final product of the careful design plans that were laid out and executed throughout this project is a fully realized and operational bread proofing box. Although some issues were encountered, these problems were either anticipated or overcome with enough effort. While only a prototype, the design successfully meets and exceeds the goals initially set out in the proposal, and even achieved some stretch goals.

8.1 Bread Proofing Box

Figure 16: Bread Proofing Box

The constructed bread-proofing box can be viewed in Figure 16 above. The walls, base, and top of the box were constructed with foam boards and adhered together with duct tape. The front of the box utilized a plastic, transparent sheet for viewing the state of the bread. To the left side of the box is the humidifier. By severing a hole into the side of the box, the humidifier was successfully attached via drier duct. By strategically cutting an opening atop the box, the heat lamp is able to securely rest directly over the bread. Inside the enclosure, near the back-right corner, is where the temperature/humidity sensor resides. The team tactically positioned it to be mid-height, away from direct light to achieve balanced humidity and temperature readings.

In the Figure 17 below, the electronics for the bread proofing box can be seen. To the top-right is the functional keypad for entering in set time, set temperature, and set humidity. To the bottom-right is the character display for displaying information related to the current stage, current temperature, current humidity, and time remaining in the stage. To the left of the character display, is the Nexys 4 board itself. Connected to the Nexys 4 board is a piezo buzzer and wires leading to a switch box just above the FPGA. The Arduino board on the left-side is exclusively used to provide 5V power to the relays in the switch box that control the humidifier and heat lamp. Finally, the gray-wired pin connector at the bottom-left extends to connect to the temperature/humidity sensor situated within the box.

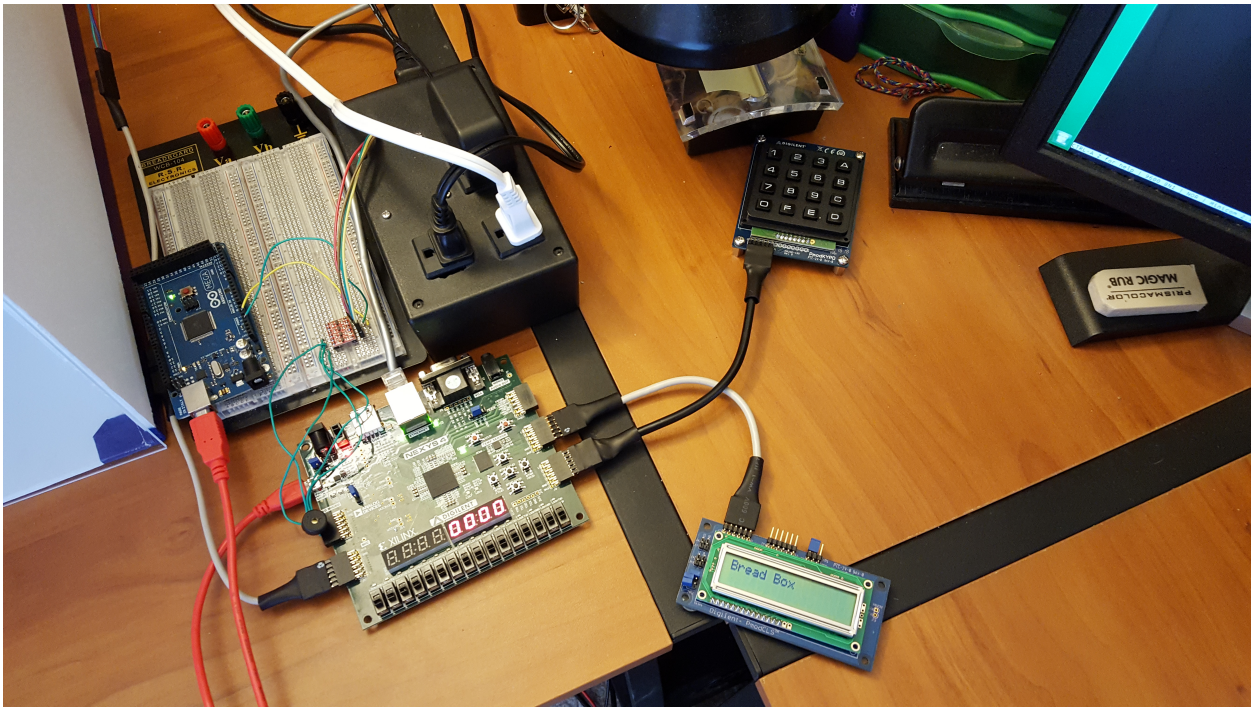

Figure 17: View of bread box electronics

8.2 Controls

The bread proofing box worked correctly according to the behavior design outlined in Section 4.2. The following pictures show the control of the bread box with the keypad and character display. Figure 18 shows the initial page and welcome screen. Figure 19 shows the page used to set the duration of a stage. 120 minutes have been entered for the duration. Figure 20 shows the page used to set the temperature set point for the stage. 80 Degrees has been entered as the target temperature. Figure 21 shows the page used to set the humidity set point. 70% humidity has been entered. Figure 22 shows the next point page. 'A' is pressed for yes and 'B' is pressed for no. Figure 23 shows the page that displays information regarding the current stage while the bread box is running. 'S' stands for state and shows the number of the current state. This number starts at 1 and increases for each additional point in the run. 'R' stands for remaining and shows the time remaining in the current stage. 'T' stands for temperature and shows the current temperature in the box. 'H' stands for humidity and shows the current humidity in the box.

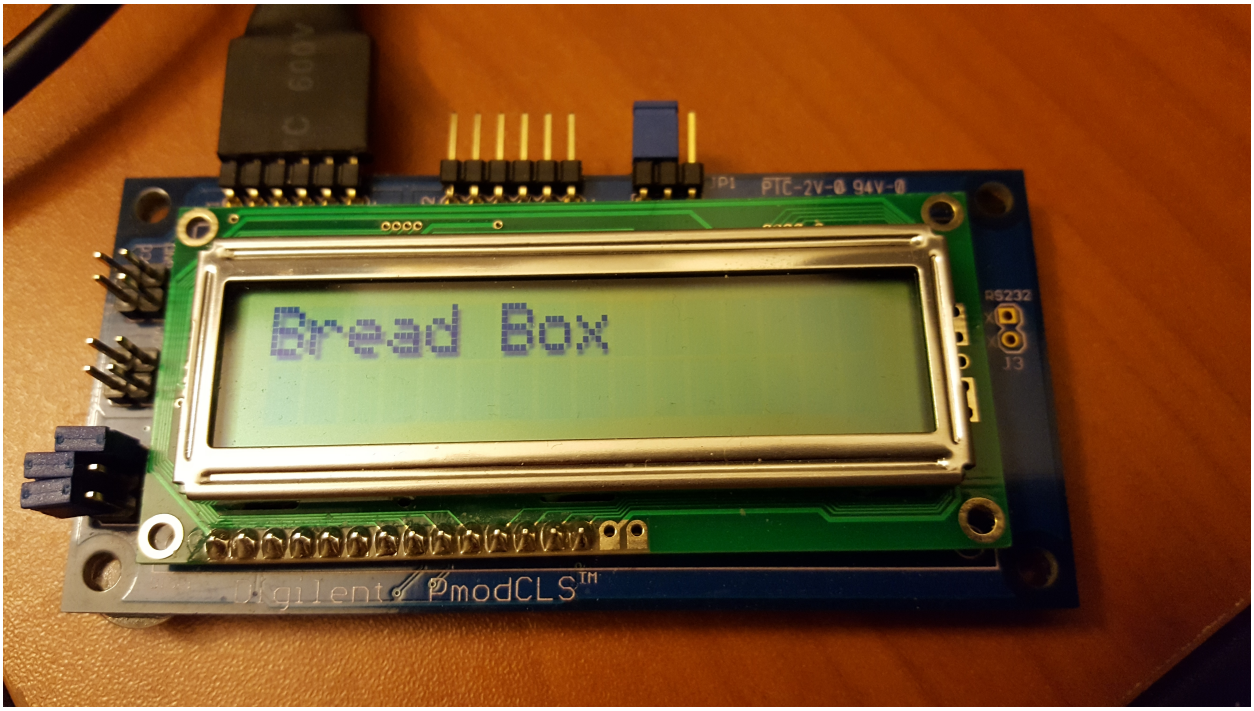

Figure 18: Initial Page

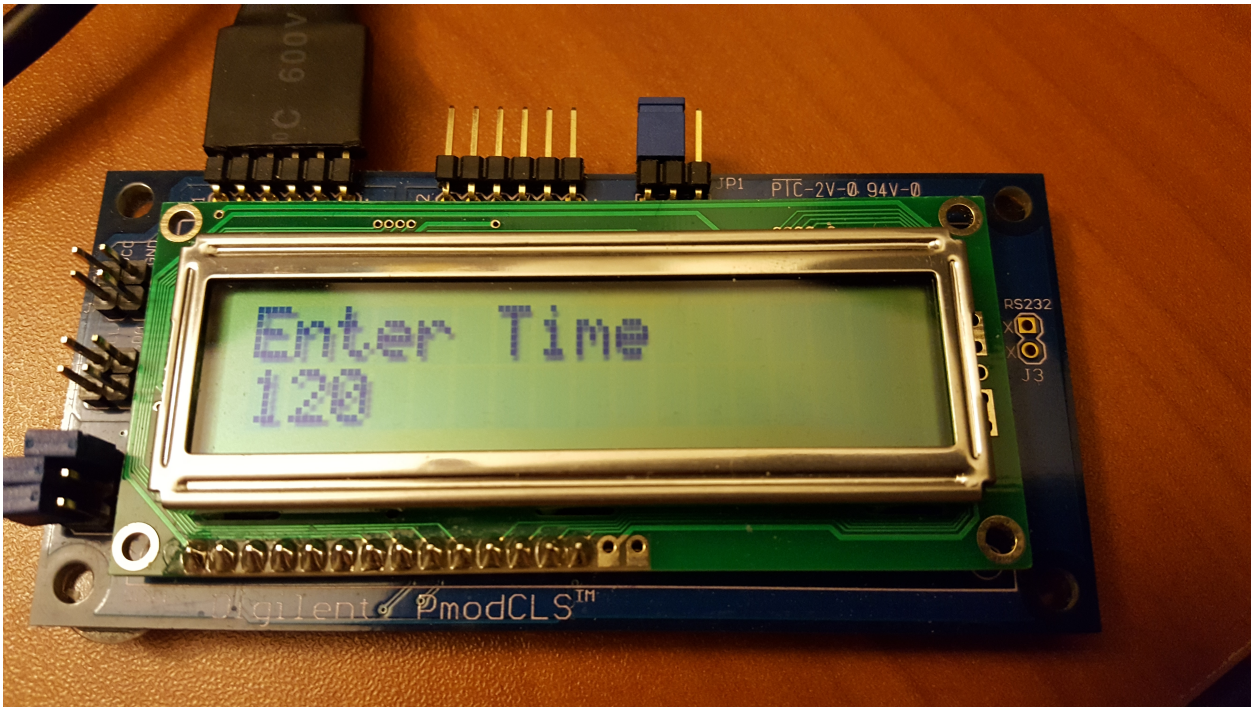

Figure 19: Time Page

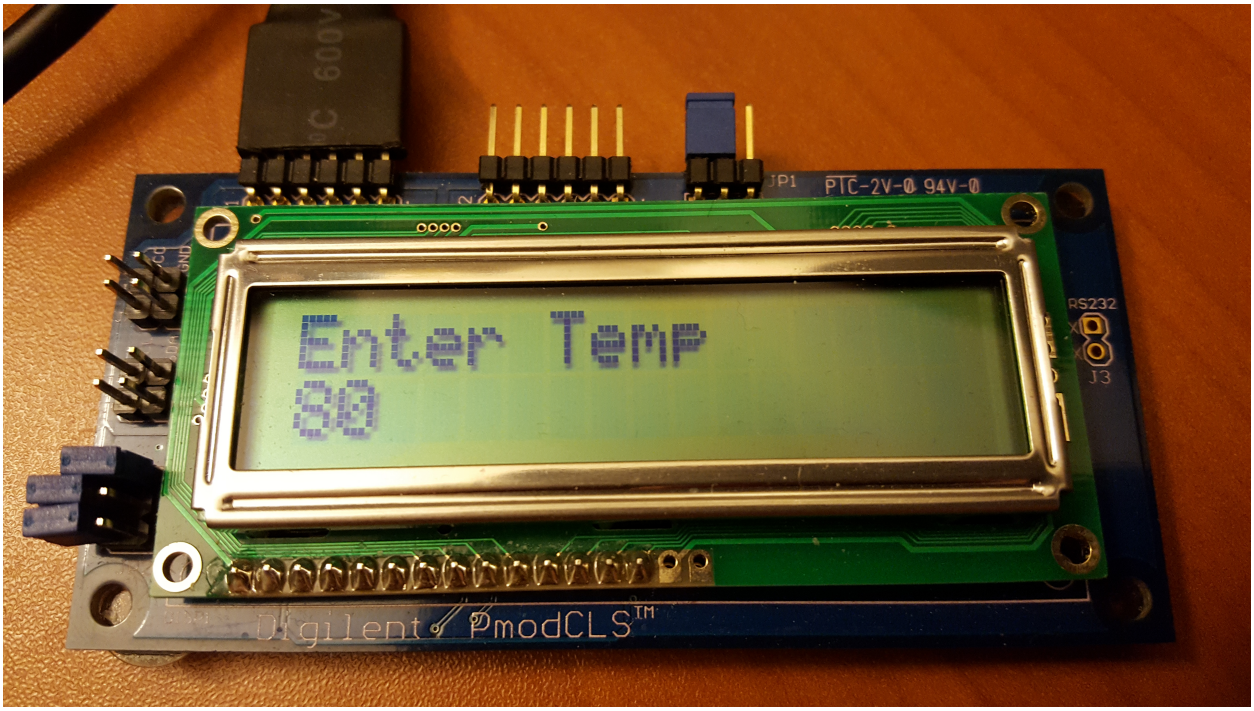

Figure 20: Temperature Page

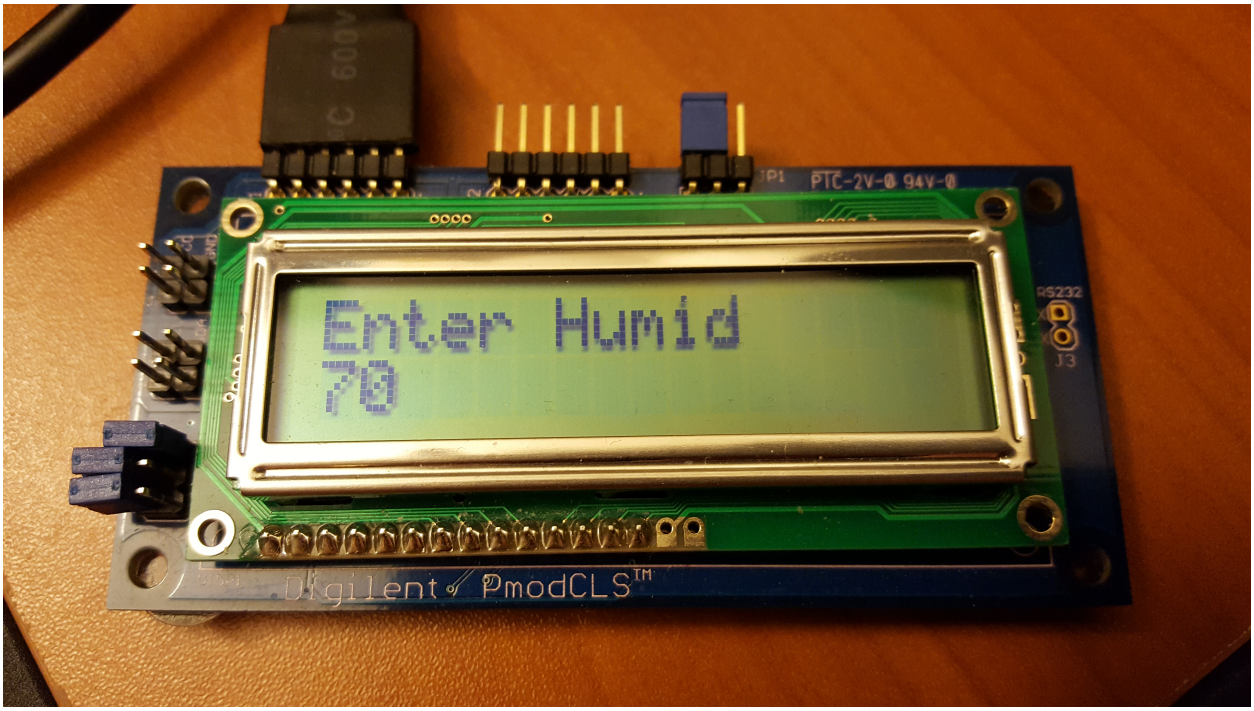

Figure 21: Humidity Page

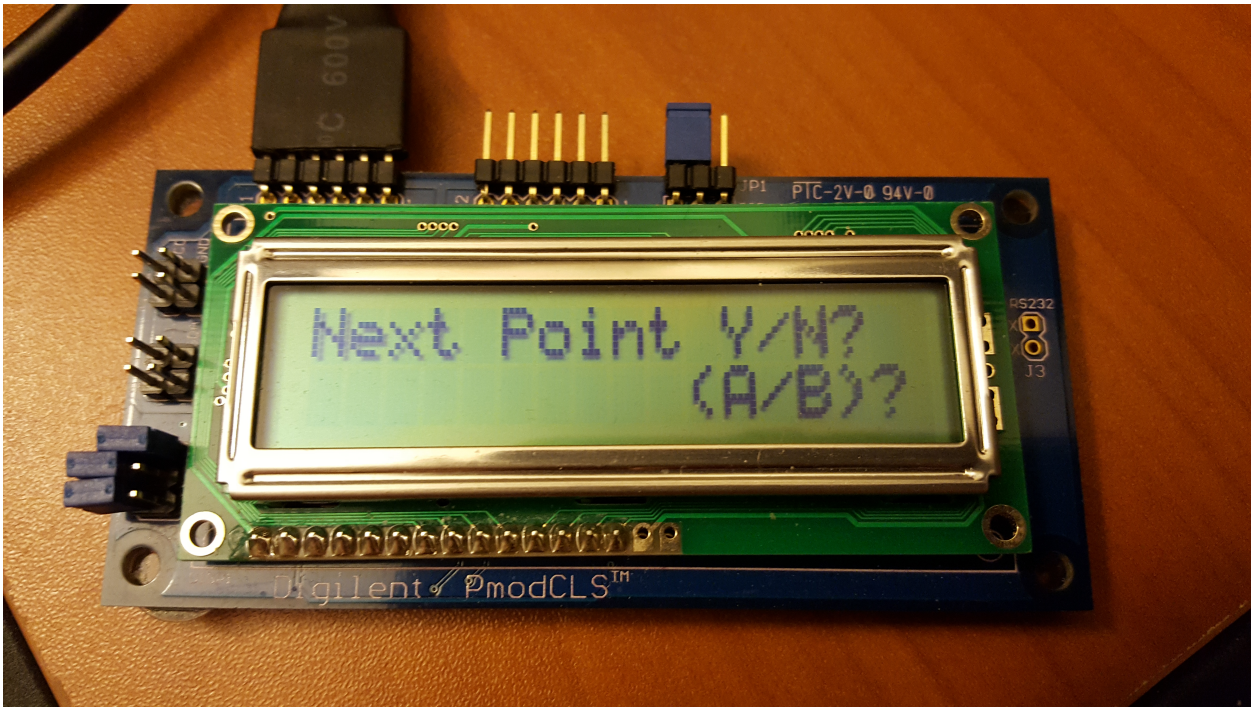

Figure 22: Next Point Page

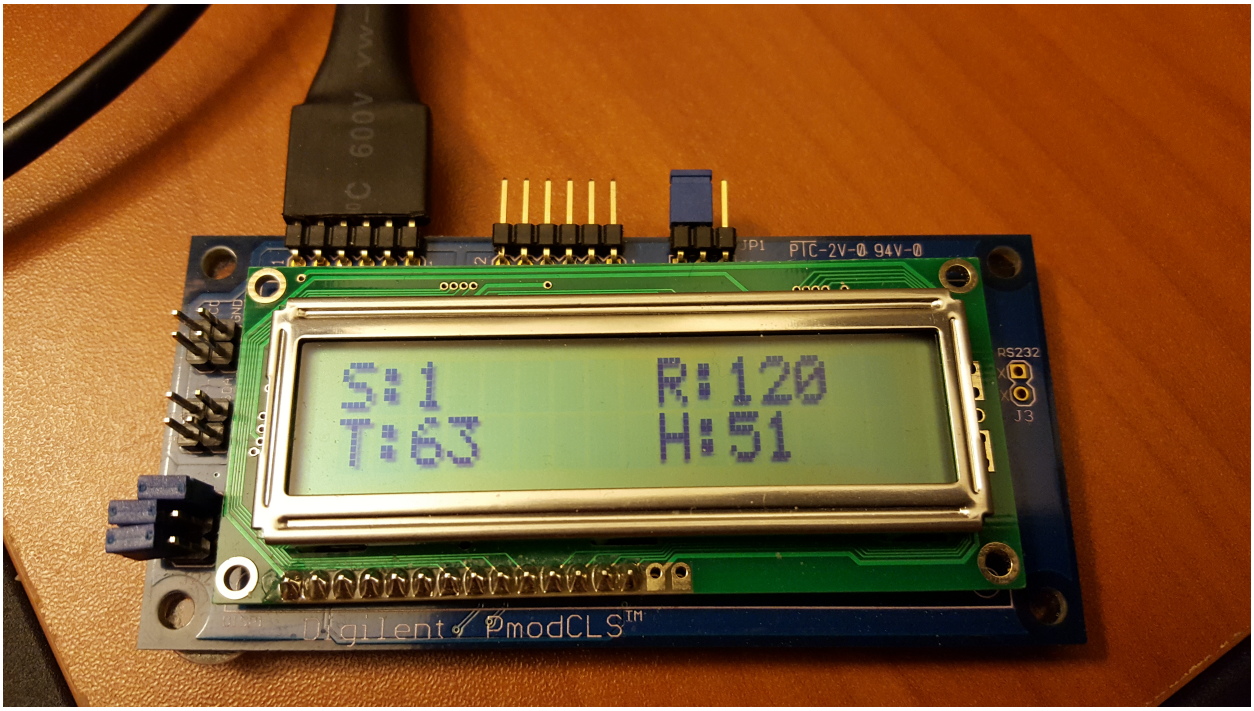

Figure 23: Run Page Page

8.3 Web Interface

As detailed previously, the web interface acts as a client that frequently polls the Microblaze for temperature, humidity, and timing information. By parsing through the received data, the values are then displayed graphically on a minute-by-minute basis for the first thirty minutes. As time progresses, the resolution of graphical display decreases, as depicted in the Table 2 below. This is due to the specific chart package for the interface. Without the premium package, the values would get truncated to the point of unreadability; therefore, specific set-points are dynamically selected for displaying based upon the number of data points themselves.

Datapoints	**Resolution**
30	1-min
60	2-min
90	3-min
120	4-min
150	5-min
180	6-min
210	7-min
240	8-min
270	9-min
300	10-min
330	11-min
360	12-min

Table 2: Data resolutions for graphical analysis.

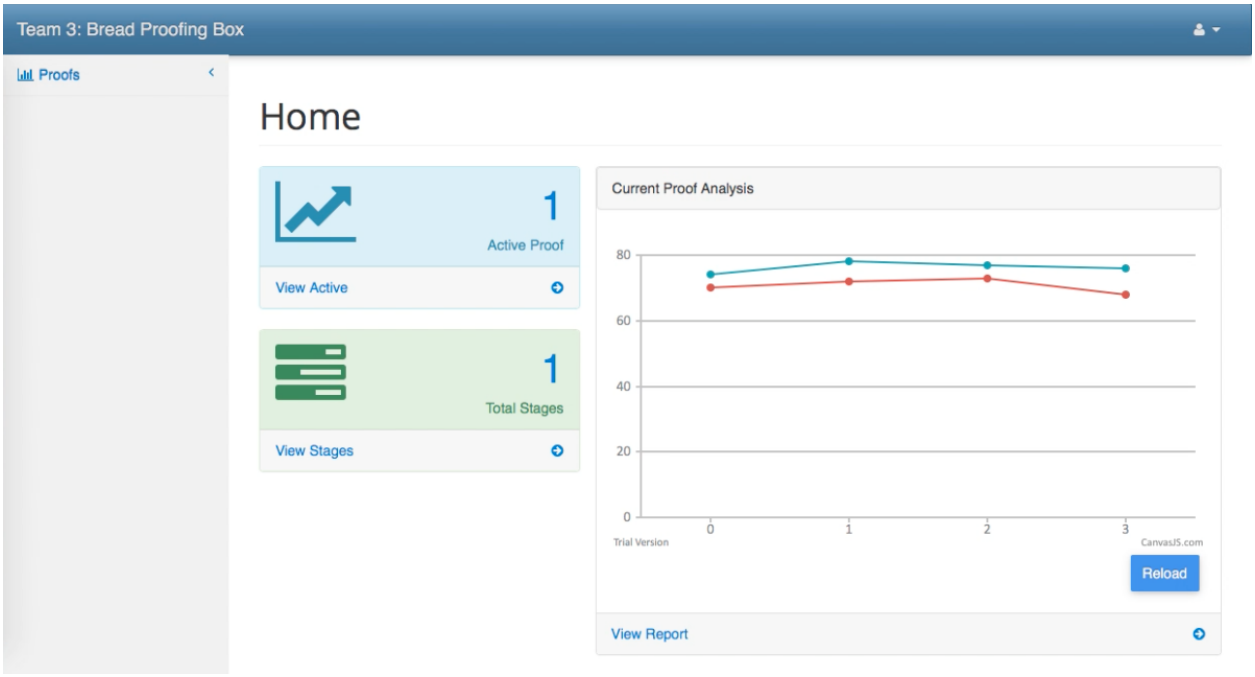

Figure 24: Home screen of interface at minute 3 of an ongoing stage.

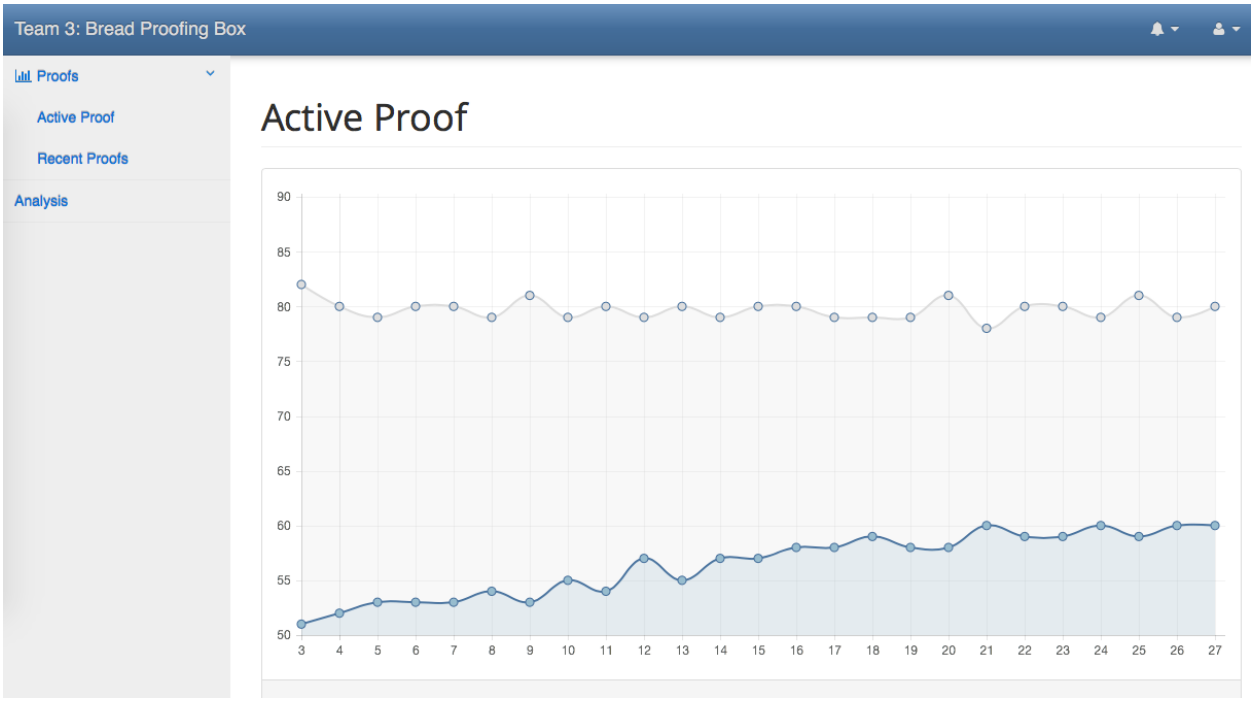

Figure 25: Active proof at minute 27 of an ongoing stage.

Alongside the displaying of temperature and humidity values, the team felt that saving previous stage data would also be necessary. To accomplish this predefined stretch goal, a MySQL server was facilitated. By allowing the web framework to gain access to the MySQL server, the first three stages are capable of being stored. The manner in which the stages are saved are determined by the stage value provided by the Microblaze server. Specifically, by storing each current stage value as a session variable, the application compares the new stage value to the old one. If the new stage value is "0" (indicating that the entire bread proofing run has finalized) or greater than the previous stage value, the application knows to store the previous stage data. This requires deleting any previous runs, resetting the previous stage's temperature, humidity, and time lists, and incrementing the stage counter. Figures 26 and 27 detail two saved runs available for users to view. As shown in the images, by clicking on a saved stage, a user can view the duration, min/max, and average values experienced throughout the stage.

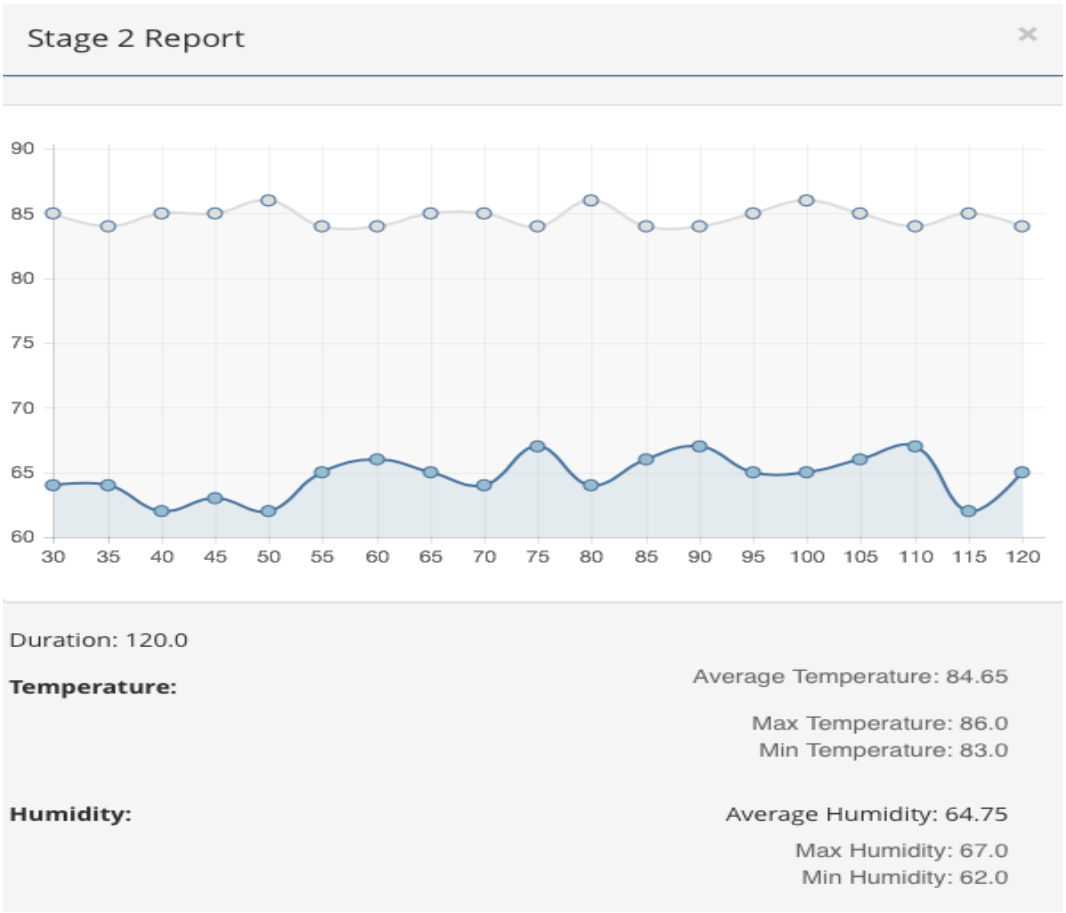

Figure 26: Stage 2 report.

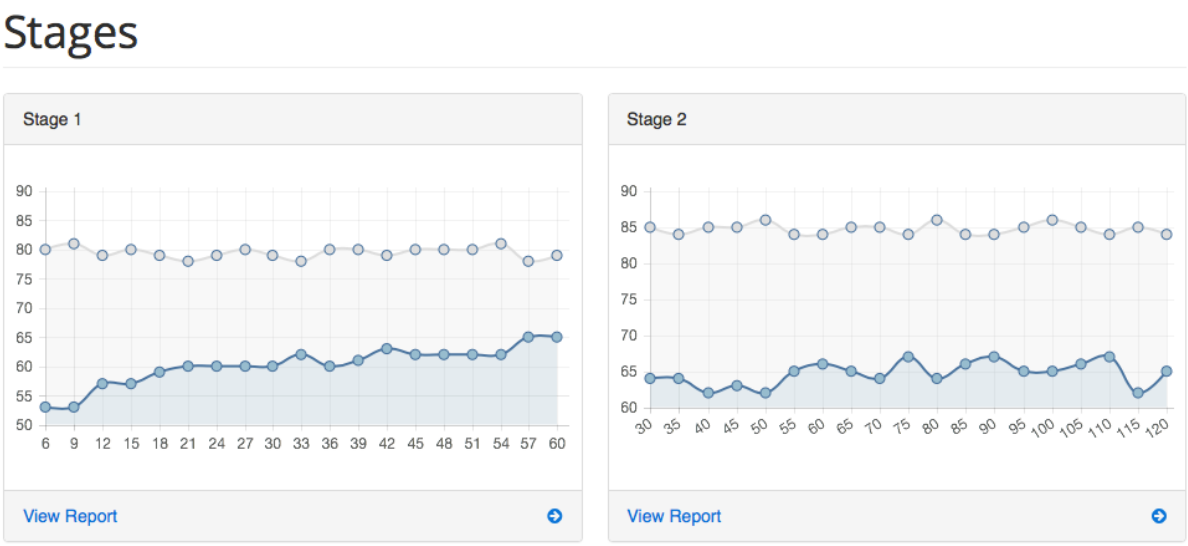

Figure 27: Saved stages available for viewing.

8.4 Bread Making

Once the bread box became functional, it was time to make bread. The actual steps to making the bread is a product of experience and no longer follows a fixed recipe; however, the recipe that was used when starting out is included in Appendix D. The main steps to bread making are mixing, autolyse, first knead, bulk ferment and folding, second knead, shaping, final proof, scoring, and baking. Different bakers follow different methods and bread making is sill very much an art form. The essential ingredients are water, flour, and salt. Starter is also needed but can be cultivated by leaving a mixture of water and flour sitting out. The starter our team used originally came from King Arthur Flour and has since been kept going for 3 years. In addition to the necessary ingredients, sugar and olive oil were also added. The sugar and the olive oil provide additional flavor. Additionally, the olive oil helps to soften the crust, but can also lead to smaller air pockets.

The following figures are all from the various steps of making bread. The pictures are from two separate uses of the bread box. Figures 28 and 29 are a before and after for the bulk fermenting step for the first use. The resulting bread is shown in Figure 30. Figures 31 and 32 show the interior porous crumb of the bread.

Figure 33 shows the second run bread after the final proof step. The finished bread is shown in Figure 34. The crumb is shown in Figure 35.

During the first run, the dough was not folded during the bulk fermentation stage. Not folding the bread allowed time lapse footage to be made and allowed for an impressive before and after picture. During the second run, the dough was folded during the bulk fermentation stage. This caused the glutton structure to form better, which caused the loaf to rise better in the oven. This change is the main reason the bread from the second run is taller and airier than the first run.

Figure 28: Before bulk ferment.

Figure 29: After bulk ferment.

Figure 30: Bread produced from first use.

Figure 31: Crumb of first artisan loaf.

Figure 32: Crumb of first sandwich loaf.

Figure 33: Final Proof of second bread.

Figure 34: Bread produced from second use.

Figure 35: Crumb of second artisan loaf.

9 Future Work

9.1 Improvements

Although the project successfully met all the goals laid out for it, the team did uncover various areas of future work while implementing the bread proofing box.

9.1.1 Enclosure

While testing the bread proofing box, the team discovered that the humidity will naturally rise when moist dough is placed in a warm enclosure. Unfortunately, it is unknown if the increase in moisture was caused by water evaporating from the surface of the bread or if it was evaporating from the water reservoir of the humidifier. Since the humidity of the enclosure rose past the set point without the humidifiers input, future work would be to determine the source of the increased humidity and to determine is a humidity removal system is needed.

The team also discovered that the light is not all that great at holding a steady temperature reading by the sensors. With a max toggle rate of 15 seconds, the temperature is able to fluctuate by about ± 2 degrees. This issue could be resolved by using a dimmer switch to control the light. The dimmer switch would allow the light to be dimmed to hold the temperature at a steady level.

Another observation was that the foam board is not water proof, which could potentially cause problems with the longevity of the box, especially since running at high humidity levels can cause condensation to form bubbles on the walls. The box also began to warp after only 3 days of use. This could be a result of moisture inside the box affecting the walls or from the weight of the lid itself. Moister is the most likely cause since the acrylic, with the back wall edges, should be able to hold the lid without problems. The side walls would need some sort of reinforcement to increase durability. Figure 36 shows the gap that has formed between the acrylic and the side wall. When the box was first built, the acrylic was flush against the outer wall.

Figure 36: Walls are starting to warp outward.

9.1.2 Web Interface

In the saving and viewing previous stages within a run, the team encountered several challenges. These challenges pertained to the linking of the MySQL server. Occasionally, upon deleting and saving new runs, the database would lock up and freeze upon executing queries. The remedy to this was simply restarting the MySQL server, and initializing the run again. However, for stages lasting hours with a vast array of data, it would be tremendously inconvenient for the data to not save properly due to database server lock-ups. Therefore, a new approach to database linking is recommended. For instance, because the interface and server has internet access, utilizing more consistent database options such as Google Firebase could prove advantageous. However, drawbacks may correspond to always requiring internet access to save proofing runs and potential network security concerns.

9.2 Additional Features

9.2.1 Enclosure

Additional future work also includes adding new features to the bread box. One such feature is the addition of the real-time clock module to allow the timer to be more accurate. Although using a counter for the on-board 100 MHz clock seems to allow the timer to keep accurate time, a more accurate external real-time clock could improve the timers accuracy for long runs. It could also enable the creation of additional features, like delayed start and ending at a given clock time. Another feature possibility, would be the addition of sensors to monitor the bread as it rises. One possible sensor for this end is an ultrasonic sensor to measure the height of the bread.

9.2.2 Web Interface

For expanding the capabilities of the web interface, the database can be linked within the Microblaze server itself. Ideally, this would be the preferred approach as the data would be stored on the server-side as opposed to the client-side. Due to time constraints and the lack of hardware availability (single FPGA board for the entire group), this approach was unattainable. Currently, the interface is only capable of displaying data once it connects to the server, which could potentially disregard many previous data points; however, by accessing the database on the server-side, the run data will be available, from their initial stages, for display across all prospective clients.

10 Conclusion

Sourdough bread making is age-old process that is as much as an art form as it is science in action. Experience and scientific knowledge is needed to make great bread. With the help of modern technology, the kitchen can become a scientific playground and a laboratory of art, allowing great discoveries to come about from chance and controlled experiments. This project set out to design and build a custom bread proofing box that utilizes the skills of modern engineers and takes advantage of the modern technological capabilities found on the Nexys 4 board. By documenting and following a well-defined design specification, the project found greater success in not only laying out realistic goals, but also in reaching and surpassing those goals. As a result, a fully-functioning bread proofing box was created. By allowing users the capability of monitoring their bread in a controlled environment, they are able to establish optimal proofing conditions to further enhance and progress the art of bread making.

Appendix A VHDL Source Code

A.1 Bread Box Top-level

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.common.all;

entity top is
    port(
        -- General Ports
        clk_100       : in std_logic; -- 100 MHz clock
        reset_n       : in std_logic; -- reset signal
        sw            : in std_logic_vector(15 downto 0); -- Switches for Debug
        led           : out std_logic_vector(15 downto 0); -- LED for debug
        RGB1_Red      : out std_logic;
        RGB2_Blue     : out std_logic;
        seg           : out std_logic_vector(7 downto 0); -- Seg for debug
        an            : out std_logic_vector(7 downto 0); -- Seg for debug

        -- Character Display
        JA : inout  STD_LOGIC_VECTOR (7 downto 0);

        -- Keypad
        JB : inout std_logic_vector (7 downto 0);

        -- RTC & Humidity and Temperature Sensor
        JC : inout std_logic_vector(7 downto 0);
        JD : inout std_logic_vector(7 downto 0);

        -- USB Uart for debug messages from micro blaze
        usb_uart_rxd : in std_logic;
        usb_uart_txd : out std_logic;

        -- Ethernet
        PhyClk50Mhz   : out std_logic;
        PhyMdc        : out STD_LOGIC;
        PhyCrs        : in STD_LOGIC;
        PhyRxErr      : in STD_LOGIC;
        PhyRxd        : in STD_LOGIC_VECTOR ( 1 downto 0 );
        PhyTxEn       : out STD_LOGIC;
        PhyTxd        : out STD_LOGIC_VECTOR ( 1 downto 0 );
        PhyMdio       : inout STD_LOGIC;

        -- Cellular memory
        MemAdr        : out STD_LOGIC_VECTOR ( 22 downto 0 );
        RamADVn       : out STD_LOGIC;
        RamLBn        : out std_logic;
        RamUBn        : out std_logic;
        RamCEn        : out STD_LOGIC;
        RamCRE        : out STD_LOGIC;
        RamOEn        : out STD_LOGIC;
        RamWait       : in STD_LOGIC;
```

```
        RamWEn          : out STD_LOGIC;
        MemDB           : inout STD_LOGIC_VECTOR ( 15 downto 0 )
    );
end entity top;

architecture structural of top is
    ---------------------------------
    -- Components
    ---------------------------------

    -- Micro Blaze
    component processing_core_wrapper
         port (
              ETH_mdio_mdc_mdc : out STD_LOGIC;
              ETH_rmii_crs_dv : in STD_LOGIC;
              ETH_rmii_rx_er : in STD_LOGIC;
              ETH_rmii_rxd : in STD_LOGIC_VECTOR ( 1 downto 0 );
              ETH_rmii_tx_en : out STD_LOGIC;
              ETH_rmii_txd : out STD_LOGIC_VECTOR ( 1 downto 0 );
              USB_Uart_rxd : in STD_LOGIC;
              USB_Uart_txd : out STD_LOGIC;
              cellular_RAM_addr : out STD_LOGIC_VECTOR ( 22 downto 0 );
              cellular_RAM_adv_ldn : out STD_LOGIC;
              cellular_RAM_ben : out STD_LOGIC_VECTOR ( 1 downto 0 );
              cellular_RAM_ce_n : out STD_LOGIC;
              cellular_RAM_cre : out STD_LOGIC;
              cellular_RAM_oen : out STD_LOGIC;
              cellular_RAM_wait : in STD_LOGIC;
              cellular_RAM_wen : out STD_LOGIC;
              cellular_ram_dq_io : inout STD_LOGIC_VECTOR ( 15 downto 0 );
              curr_humid_tri_o : out STD_LOGIC_VECTOR ( 15 downto 0 );
              curr_temperature_tri_o : out STD_LOGIC_VECTOR ( 15 downto 0 );
              duration_tri_i : in STD_LOGIC_VECTOR ( 15 downto 0 );
              eth_mdio_mdc_mdio_io : inout STD_LOGIC;
              eth_ref_clk : out STD_LOGIC;
              gpio_stage_tri_i : in STD_LOGIC_VECTOR ( 15 downto 0 );
              iic_rtl_scl_io : inout STD_LOGIC;
              iic_rtl_sda_io : inout STD_LOGIC;
              remaining_tri_i : in STD_LOGIC_VECTOR ( 15 downto 0 );
              reset : in STD_LOGIC;
              sys_clock : in STD_LOGIC
          );
    end component;

    -- Counter to generate SCL
    component c_counter_binary_0 IS
      PORT (
        CLK : IN STD_LOGIC;
        Q : OUT STD_LOGIC_VECTOR(11 DOWNTO 0)
      );
    END component;

    -- Seven segment display controller
    component display
```

```
Port ( data        : in  STD_LOGIC_VECTOR (15 downto 0);
       carry       : in  STD_LOGIC;
       clk         : in  STD_LOGIC;
       seg_number  : out STD_LOGIC_VECTOR (63 downto 0) );
end component;

-- Seven segment component
component sSegDisplay
    Port(ck       : in  std_logic;                          -- 100MHz system
        clock
         reset_n  : in std_logic;                           -- Reset signal
         number   : in  std_logic_vector (63 downto 0);     -- eight digit
            number to be displayed
         seg      : out  std_logic_vector (7 downto 0);     -- display
            cathodes
         an       : out  std_logic_vector (7 downto 0));    -- display anodes
            (active-low, due to transistor complementing)
end component;

component RAM_2K_8
    port (
            DOA : out std_logic_vector(7 downto 0);
            DOB : out std_logic_vector(7 downto 0);
            DOPA : out std_logic_vector(0 downto 0);
            DOPB : out std_logic_vector(0 downto 0);
            ADDRA : in std_logic_vector(10 downto 0);
            ADDRB : in std_logic_vector(10 downto 0);
            CLKA : in std_logic;
            CLKB : in std_logic;
            DIA : in std_logic_vector(7 downto 0);
            DIB : in std_logic_vector(7 downto 0);
            DIPA : in std_logic_vector(0 downto 0);
            DIPB : in std_logic_vector(0 downto 0);
            ENA : in std_logic;
            ENB : in std_logic;
            SSRA : in std_logic;
            SSRB : in std_logic;
            WEA : in std_logic;
            WEB : in std_logic
        );
end component;

component Constant_Character_RAM
    port (
            DOA : out std_logic_vector(7 downto 0);
            DOB : out std_logic_vector(7 downto 0);
            DOPA : out std_logic_vector(0 downto 0);
            DOPB : out std_logic_vector(0 downto 0);
            ADDRA : in std_logic_vector(10 downto 0);
            ADDRB : in std_logic_vector(10 downto 0);
            CLKA : in std_logic;
            CLKB : in std_logic;
            DIA : in std_logic_vector(7 downto 0);
            DIB : in std_logic_vector(7 downto 0);
```

```
            DIPA : in std_logic_vector(0 downto 0);
            DIPB : in std_logic_vector(0 downto 0);
            ENA : in std_logic;
            ENB : in std_logic;
            SSRA : in std_logic;
            SSRB : in std_logic;
            WEA : in std_logic;
            WEB : in std_logic
        );
end component;

component Message_Setter
Port (
    CLK : in  STD_LOGIC;                                    -- Clock
    Update_Pulse : in std_logic;                            -- Pulse to update
        the screen
    Reset : in  STD_LOGIC;                                  -- Reset
    Static_Address_Base : in integer range 0 to 2047;       -- Base address for
        static display
    Dynamic_Address_Base : in integer range 0 to 2047;      -- Base address for
        dynamic display
    ProcDataRAM_In : out  T_RAM_Port_In;                    -- Processed Data
        Ram
    ProcDataRAM_Out : in  T_RAM_Port_Out;                   -- Processed Data
        Ram
    ConstCharRAM_In : out T_RAM_Port_In;                    -- Constant Char RAM
    ConstCharRAM_Out : in T_RAM_Port_Out;                   -- Constant Char RAM
    DispCLR : out  STD_LOGIC;                               -- Clear Display
    DispClearMem : out  STD_LOGIC;                          -- Clear Display Mem
    DispAddr : out integer range 0 to 31;                   -- Display Mem Addr
    DispData : out STD_LOGIC_VECTOR (7 downto 0);           -- Display Data
    DispDataEnable : out  STD_LOGIC;                        -- Display Data
        Enable
    UpdateDisplay : out STD_LOGIC;                          -- Update Char
        Display
    DispInProg : in  STD_LOGIC);                            -- Display Updating
end component;

component CharDisplay
    Port ( CLK : in  STD_LOGIC;                             --clock
           Reset : in STD_LOGIC;                            -- Reset

           CLR : in  STD_LOGIC;                             --clears screen
               only
           ClearMem : in STD_LOGIC;                         --clears memory
           RST : in STD_LOGIC;                              --sets screen to
               power up state - clears screen

           DAddr : in integer range 0 to 31;                --address of new
               character
           DataIn : in  STD_LOGIC_VECTOR (7 downto 0);      --New character
           DataInEnable : in  STD_LOGIC;                    --signal to read
               in new character
```

```
            UpdateDisplay : in STD_LOGIC;
            DispInProg : out STD_LOGIC;                    --Indicates new
                line display
            JA : inout  STD_LOGIC_VECTOR (3 downto 0));    --pins of CLS
                Display
end component;

component Keypad
    Port (
            clk         : in  std_logic;
            Row         : in  std_logic_vector (3 downto 0);
            Col         : out std_logic_vector (3 downto 0);
            -- Last number pressed by itself
            DecodeOut : out std_logic_vector (3 downto 0);
            -- Vector of the buttons pressed
            buttons    : out std_logic_vector (15 downto 0);
            -- Boolean if buttons are pressed.
            pressed    : out std_logic
        );
end component;

-- RTC
component RTC is
    port (
        clk : in std_logic;
        rst : in std_logic;
        SCL : inout std_logic;
        SDA : inout std_logic
    );
end component;

component output_control is
    Port (
            clk : in STD_LOGIC;
            rst : in std_logic;
            curr_temperature : in STD_LOGIC_VECTOR (15 downto 0);
            curr_humidity : in STD_LOGIC_VECTOR (15 downto 0);
            set_temperature : in std_logic_vector(15 downto 0);
            set_humidity : in std_logic_vector(15 downto 0);
            enable : in std_logic;
            heat_control : out STD_LOGIC;
            humid_control : out STD_LOGIC
        );
end component;

component Proofing_State
Port (
       CLK,                                                -- System
           clock
       reset_n : in std_logic;                             -- Reset
       display_update_pulse : in std_logic;                -- pulse to
           update run display
       clk_1hz : in std_logic;                             -- One hertz
           clock used to update the timer
```

```
        button : in std_logic_vector (3 downto 0);              -- Button
            press from keypad
        pressed : in std_logic;                                 -- Pressed
            from keypad
        curr_humidity : in std_logic_vector(15 downto 0);       -- Current
            humidity from the sensor
        curr_temperature : in std_logic_vector(15 downto 0); -- Current
            temperature from the sensor
        Mode : out  t_proofer_state;                            -- Current
            mode of the state machine
        Static_Address_Base : out integer range 0 to 2047;  -- Base
            address for static display
        Dynamic_Address_Base : out integer range 0 to 2047; -- Base
            address for dynamic display
        DisplayDataRAM_In : out  T_RAM_Port_In;                 -- Display
            Data Ram
        DisplayDataRAM_Out : in  T_RAM_Port_Out;                -- Display
            Data Ram
        RunDataRAM_In : out  T_RAM_Port_In;                     -- Run Data
            Ram
        RunDataRAM_Out : in  T_RAM_Port_Out;                    -- Run Data
            Ram
        buzzer_enable : out std_logic;                          -- Enable the
            buzzer
        active_time : out std_logic_vector(15 downto 0);        -- Active Time
        active_time_remaining : out std_logic_vector(15 downto 0); --
            Remaining time for the run
        active_temp : out std_logic_vector(15 downto 0);        -- Active
            Temperature
        active_humid : out std_logic_vector(15 downto 0);       -- Active
            Humidity
        active_stage : out std_logic_vector(15 downto 0);       -- Active
            stage
        run_enable : out std_logic                              -- Is the
            device running?
    );
end component;

component CDiv
        GENERIC (
            TC : integer := 18 --Time Constant. 15 for ~1khz -- 50000000/(
                TC^4) Hz
        );
        PORT (
            Cin   : IN  STD_LOGIC;
            Reset : IN STD_LOGIC;
            Cout : OUT STD_LOGIC);
    end component;

component buzzer
    port (
            clk_100 : in std_logic;
            enable  : in std_logic;
            buzzer  : out std_logic
```

```
        );
end component;


-------------------------------
-- Signals
-------------------------------
signal reset          : std_logic;
signal clk            : std_logic;

--
-- Signals for output display
--
signal result_data    : std_logic_vector(15 downto 0);
signal carry          : std_logic;
signal display_number : std_logic_vector (63 downto 0);

--
-- Microblaze
--
signal cellular_RAM_ben  : std_logic_vector(1 downto 0);

--
-- Bread Machine Control
--
signal current_state : t_proofer_state;
signal real_time_clk : std_logic;
signal update_run_display  : std_logic;

--
-- Output control signals
--
signal buzzer_enable : std_logic;

--
-- Character display signals
--
signal Static_Address_Base : integer range 0 to 2047;  -- Base address for
    static display
signal Dynamic_Address_Base : integer range 0 to 2047; -- Base address for
    dynamic display

--
-- Keypad
--
signal keypad_pressed : std_logic;
signal keypad_button : std_logic_vector(3 downto 0);

-- Parser to Ram
signal parser_to_RAM_In : t_RAM_Port_In;
signal parser_to_RAM_Out : t_RAM_Port_Out;

-- Processed Ram to Message Setter
signal Proc_to_Msg_Setter_In : t_RAM_Port_In;
```

```
signal Proc_to_Msg_Setter_Out : t_RAM_Port_Out;

-- Const Ram to Message Setter
signal const_to_Msg_Setter_In : t_RAM_Port_In;
signal const_to_Msg_Setter_Out : t_RAM_Port_Out;

-- Data Ram to Calculation Block
signal Data_RAM_in: t_RAM_Port_In;
signal Data_RAM_out: t_RAM_Port_Out;

-- Calculation Block to Processed Ram
signal Proc_RAM_in: t_RAM_Port_In;
signal Proc_RAM_out: t_RAM_Port_Out;

-- Slow update pulse for Message Setter
signal update_char : std_logic;

-- Char Display
type t_char_disp is record
   DispCLR : std_logic;
   DispClearMem : std_logic;
   DispAddr : integer range 0 to 31;
   DispData : STD_LOGIC_VECTOR (7 downto 0);
   DispDataEnable : std_logic;
   UpdateDisplay : std_logic;
   DispInProg : std_logic;
end record;
signal char_disp : t_char_disp;

-- I2C Signals
signal q : std_logic_vector(11 downto 0);
signal err : std_logic;
signal Dout : std_logic_vector(7 downto 0);
signal clk_i2c : std_logic := '0';

--
-- Live data signals
--
signal curr_humidity : std_logic_vector(15 downto 0);
signal curr_temperature : std_logic_vector(15 downto 0);

--
-- Set data signals
--
signal set_temperature : std_logic_vector(15 downto 0);
signal set_humidity    : std_logic_vector(15 downto 0);

signal enable_output : std_logic;


--
-- Time Signals
--
signal curr_duration : std_logic_vector(15 downto 0);
```

```
    signal curr_remaining: std_logic_vector(15 downto 0);
    signal active_stage  : std_logic_vector(15 downto 0);

    --
    -- Output control
    --
    signal humidity_out    : std_logic;
    signal temperature_out : std_logic;

begin

    -- Set the reset
    reset <= not reset_n;

    display_logic : display
        port map (
                  data       => result_data,
                  carry      => carry,
                  clk        => clk_100,
                  seg_number => display_number
    );
    carry  <= '0';

    seven_seg_driver : sSegDisplay
        port map (
                  ck             => clk_100,
                  reset_n        => reset_n,
                  number         => display_number,
                  seg            => seg,
                  an             => an
    );

    -- RTC
--    real_time_clock : RTC
--        port map (
--                    clk => clk_i2c,
--                    rst => reset,
--                    SCL => JC(2),
--                    SDA => JC(3)
--                  );

    Touch_pad : Keypad
    Port map (
                clk => clk_100,
                Row => JB(7 downto 4),
                Col => JB(3 downto 0),
                DecodeOut => keypad_button,
                pressed => keypad_pressed
            );

    control : output_control
    Port map (
                clk => clk_100,
                rst => reset,
```

```
            curr_temperature => curr_temperature,
            curr_humidity => curr_humidity,
            set_temperature => set_temperature,
            set_humidity => set_humidity,
            enable => enable_output,
            heat_control => temperature_out,
            humid_control => humidity_out
        );

JD(6) <= not humidity_out;
JD(7) <= not temperature_out;

c_state : Proofing_State
Port map (
             CLK  => clk_100,
             reset_n => reset_n,
             display_update_pulse => update_run_display,
             clk_1hz => real_time_clk,
             button => keypad_button,
             pressed => keypad_pressed,
             curr_humidity => curr_humidity,
             curr_temperature => curr_temperature,
             Mode => current_state,
             Static_Address_Base => Static_Address_Base,
             Dynamic_Address_Base => Dynamic_Address_Base,
             DisplayDataRAM_In => Proc_RAM_in,
             DisplayDataRAM_Out => Proc_RAM_out,
             RunDataRAM_In => Data_RAM_in,
             RunDataRAM_Out => Data_RAM_out,
             buzzer_enable => buzzer_enable,
             active_temp => set_temperature,
             active_humid => set_humidity,
             run_enable => enable_output,
             active_time => curr_duration,
             active_time_remaining => curr_remaining,
             active_stage => active_stage
        );

-- Clock divider used to choose when to show new data from the sensors
display_update : CDiv
generic map(
            TC => 118)
port map(
            Cin => CLK_100,
            Reset => '0',
            Cout => update_run_display);

-- Clock divider used to generate a 1 hertz clock
real_time : CDiv
generic map(
            TC => 100)
port map(
            Cin => CLK_100,
            Reset => '0',
```

```
            Cout => real_time_clk);

micro_proc : processing_core_wrapper
    port map (
                sys_clock                   => clk_100,
                reset                       => reset,
                USB_Uart_rxd                => usb_uart_rxd,
                USB_Uart_txd                => usb_uart_txd,
                ETH_mdio_mdc_mdc            => PhyMdc,
                ETH_rmii_crs_dv             => PhyCrs,
                ETH_rmii_rx_er              => PhyRxErr,
                ETH_rmii_rxd                => PhyRxd,
                ETH_rmii_tx_en              => PhyTxEn,
                ETH_rmii_txd                => PhyTxd,
                cellular_RAM_addr           => MemAdr,
                cellular_RAM_adv_ldn        => RamADVn,
                cellular_RAM_ben            => cellular_RAM_ben,
                cellular_RAM_ce_n           => RamCEn,
                cellular_RAM_cre            => RamCRE,
                cellular_RAM_oen            => RamOEn,
                cellular_RAM_wait           => RamWait,
                cellular_RAM_wen            => RamWEn,
                cellular_ram_dq_io          => MemDB,
                eth_mdio_mdc_mdio_io        => PhyMdio,
                eth_ref_clk                 => PhyClk50Mhz,
                iic_rtl_scl_io              => JC(2),
                iic_rtl_sda_io              => JC(3),
                curr_temperature_tri_o      => curr_temperature,
                curr_humid_tri_o            => curr_humidity,
                duration_tri_i              => curr_duration,
                remaining_tri_i             => curr_remaining,
                gpio_stage_tri_i            => active_stage

);

RamUBn <= cellular_RAM_ben(1);
RamLBn <= cellular_RAM_ben(0);

-- Run Ram
data_ram : RAM_2K_8
    port map (
       DOA => parser_to_RAM_Out.DO,
       DOB => Data_RAM_out.DO,
       DOPA => parser_to_RAM_Out.DOP,
       DOPB => Data_RAM_out.DOP,
       ADDRA => parser_to_RAM_In.ADDR,
       ADDRB => Data_RAM_in.ADDR,
       CLKA => CLK_100,
       CLKB => CLK_100,
       DIA => parser_to_RAM_In.DI,
       DIB => Data_RAM_in.DI,
       DIPA => parser_to_RAM_In.DIP,
       DIPB => Data_RAM_in.DIP,
       ENA => parser_to_RAM_In.EN,
```

```
        ENB => Data_RAM_in.EN,
        SSRA => Reset,
        SSRB => Reset,
        WEA => parser_to_RAM_In.WE,
        WEB => Data_RAM_in.WE
        );

-- Processed Date Ram
Proc_RAM : RAM_2K_8
    port map (
        DOA => Proc_RAM_Out.DO,
        DOB => Proc_to_Msg_Setter_Out.DO,
        DOPA => Proc_RAM_Out.DOP,
        DOPB => Proc_to_Msg_Setter_Out.DOP,
        ADDRA => Proc_RAM_in.ADDR,
        ADDRB => Proc_to_Msg_Setter_In.ADDR,
        CLKA => CLK_100,
        CLKB => CLK_100,
        DIA => Proc_RAM_in.DI,
        DIB => Proc_to_Msg_Setter_In.DI,
        DIPA => Proc_RAM_in.DIP,
        DIPB => Proc_to_Msg_Setter_In.DIP,
        ENA => Proc_RAM_in.EN,
        ENB => Proc_to_Msg_Setter_In.EN,
        SSRA => Reset,
        SSRB => Reset,
        WEA => Proc_RAM_in.WE,
        WEB => Proc_to_Msg_Setter_In.WE
        );

-- Constant Date Ram
    Const_RAM : Constant_Character_RAM
        port map (
            DOA => const_to_Msg_Setter_Out.DO,
            DOB => open,
            DOPA => const_to_Msg_Setter_Out.DOP,
            DOPB => open,
            ADDRA => const_to_Msg_Setter_In.ADDR,
            ADDRB => (others => '1'),
            CLKA => CLK_100,
            CLKB => '1',
            DIA => const_to_Msg_Setter_In.DI,
            DIB => (others => '1'),
            DIPA => const_to_Msg_Setter_In.DIP,
            DIPB => (others => '1'),
            ENA => const_to_Msg_Setter_In.EN,
            ENB => '0',
            SSRA => Reset,
            SSRB => '0',
            WEA => const_to_Msg_Setter_In.WE,
            WEB => '0'
            );

    -- Message Setter
```

```
msg_setter : Message_Setter
   port map (
      CLK => CLK_100,
      Update_Pulse => update_char,
      Reset => Reset,
      ProcDataRAM_In => Proc_to_Msg_Setter_In,
      ProCDataRAM_Out => Proc_to_Msg_Setter_Out,
      ConstCharRAM_In => const_to_Msg_Setter_In,
      ConstCharRAM_Out => const_to_Msg_Setter_Out,
      DispCLR => char_disp.DispCLR,
      DispClearMem => char_disp.DispClearMem,
      DispAddr => char_disp.DispAddr,
      DispData => char_disp.DispData,
      DispDataEnable => char_disp.DispDataEnable,
      UpdateDisplay => char_disp.UpdateDisplay,
      DispInProg => char_disp.DispInProg,
      Static_Address_Base => Static_Address_Base,
      Dynamic_Address_Base => Dynamic_Address_Base
      );

-- Communication to the CharDisplay
CDisplay : CharDisplay port map (
   CLK => CLK_100,
   Reset => Reset,
   CLR => char_disp.DispCLR,
   ClearMem => char_disp.DispClearMem,
   RST => Reset,
   DAddr => char_disp.DispAddr,
   DataIn => char_disp.DispData,
   DataInEnable => char_disp.DispDataEnable,
   UpdateDisplay => char_disp.UpdateDisplay,
   DispInProg => char_disp.DispInProg,
   JA => JA(3 downto 0));

-- Clock divider used to get a slow clock for the Seven Segment display
divset : CDiv
generic map(
               TC => 35)
port map(
            Cin => CLK_100,
            Reset => '0',
            Cout => update_char);

 generate_scl : process(q)
 begin
     if(q = x"1F4") then
         clk_i2c <= not clk_i2c;
     end if;
 end process generate_scl;

 alert : buzzer
     port map (
         clk_100 => clk_100,
         enable  => buzzer_enable,
```

```
            buzzer   => JD(3)
        );

end architecture;
```

A.2 Proofing State Machine

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library work;
use work.common.all;

entity Proofing_State is
    Port (
            CLK,                                                    --
                System clock
            reset_n : in std_logic;                                 -- Reset
            display_update_pulse : in std_logic;                    -- pulse
                 to update run display
            clk_1hz : in std_logic;                                 -- One
                hertz clock used to update the timer
            button : in std_logic_vector (3 downto 0);              --
                Button press from keypad
            pressed : in std_logic;                                 --
                Pressed from keypad
            curr_humidity : in std_logic_vector(15 downto 0);       --
                Current humidity from the sensor
            curr_temperature : in std_logic_vector(15 downto 0);    --
                Current temperature from the sensor
            Mode : out  t_proofer_state;                            --
                Current mode of the state machine
            Static_Address_Base : out integer range 0 to 2047;      -- Base
                address for static display
            Dynamic_Address_Base : out integer range 0 to 2047;     -- Base
                address for dynamic display
            DisplayDataRAM_In : out  T_RAM_Port_In;                 --
                Display Data Ram
            DisplayDataRAM_Out : in  T_RAM_Port_Out;                --
                Display Data Ram
            RunDataRAM_In : out  T_RAM_Port_In;                     -- Run
                Data Ram
            RunDataRAM_Out : in  T_RAM_Port_Out;                    -- Run
                Data Ram
            buzzer_enable : out std_logic;                          --
                Enable the buzzer
            active_time : out std_logic_vector(15 downto 0);        --
                Active Time
            active_time_remaining : out std_logic_vector(15 downto 0); --
                Remaining time for the run
            active_temp : out std_logic_vector(15 downto 0);        --
                Active Temperature
            active_humid : out std_logic_vector(15 downto 0);       --
                Active Humidity
            active_stage : out std_logic_vector(15 downto 0);       --
                Active stage
            run_enable : out std_logic                              -- Is
```

```
            the device running?
    );
end Proofing_State;

architecture Behavioral of Proofing_State is

    component bcd_2_bin
    Port (
          bcd_in : in t_bcd;
          bin_out : out  STD_LOGIC_VECTOR (15 downto 0) := (others => '0')
     );
    end component;

    component DoubleDabble
        port (
              bin_in  : in std_logic_vector (15 downto 0);
              bcd_out : out t_bcd
         );
    end component;

    signal current_state : t_proofer_state := main;

    -- Signals to create a single pulse when a button is first pressed.
    signal previous_pressed : std_logic := '0';
    signal pulse : std_logic := '0';

    -- Return state for subprocess.
    -- Allows for "calls" to be made in the state machine
    signal return_state : t_proofer_state;

    -- Addresses used for accessing the display data ram and the run data ram
    signal current_display_data_address : integer range 0 to 2047;
    signal base_display_data_address : integer range 0 to 2047;
    signal current_run_data_address : integer range 0 to 2047;
    signal base_run_data_address : integer range 0 to 2047;

    -- Number conversion signals
    signal bcd_2_bin_bcd : t_bcd;
    signal bcd_2_bin_bin : std_logic_vector(15 downto 0);

    signal dd_bcd : t_bcd;
    signal dd_bin : std_logic_vector(15 downto 0);

    signal bcd_index : integer range 0 to 5;
    signal bin_index : integer range 0 to 1;

    -- Signals for run state
    signal run_sub_counter : integer range 0 to 31;
    signal run_time : std_logic_vector(15 downto 0);
    signal run_time_remaining : std_logic_vector(15 downto 0);
    signal run_temp : std_logic_vector(15 downto 0);
    signal run_humid : std_logic_vector(15 downto 0);
    signal run_leading : std_logic;
    signal run_stage : std_logic_vector(15 downto 0);
```

```
begin
    -- Component used to convert bcd a number into a binary number
    convert_bcd : bcd_2_bin
    Port map(
             bcd_in => bcd_2_bin_bcd,
             bin_out => bcd_2_bin_bin
         );

    dd : DoubleDabble
        port map (
                 bin_in  => dd_bin,
                 bcd_out => dd_bcd
            );

    -- Send out the current mode
    Mode <= current_state;

    -- Send out the current run stage
    active_stage <= run_stage;

    -- Enable the buzzer when the state is alert
    buzzer_enable <= '1' when current_state = alert else '0';

    -- Main state machine
    state : process (clk, reset_n)
        -- Temporary variable to aid in conversion
        variable tmp : std_logic_vector(7 downto 0);

        -- Counter for the timer
        variable minute_count : integer range 0 to 255 := 0;
    begin
        -- Reset state for the state machine
        if (reset_n = '0') then
            current_state <= main;
            DisplayDataRAM_In.EN <= '0';
            RunDataRAM_In.EN <= '0';
            current_run_data_address <= 0;
            base_run_data_address <= 0;
            run_enable <= '0';
        -- On the clock preforms actions.
        elsif(rising_edge(clk)) then
            case (current_state) is
                -- Main state displays the welcome page
                when main =>
                    current_run_data_address <= 0;
                    base_run_data_address <= 0;
                    Static_Address_Base  <= Const_RAM.main;
                    Dynamic_Address_Base <= Proc_RAM.main;
                    if pulse = '1' then
                        current_state <= enter_time;
                    end if;

                -- Setup the memory addressed for entering time then run sub-
```

```
    routine
when enter_time =>
    Static_Address_Base  <= Const_RAM.enter_time;
    Dynamic_Address_Base <= Proc_RAM.enter_time;
    return_state <= enter_temp;
    current_state <= h_data_add_subprocess_setup;
    base_display_data_address <= Proc_RAM.enter_time;

-- Setup the memory addressed for entering temp then run sub-
    routine
when enter_temp =>
    Static_Address_Base  <= Const_RAM.enter_temp;
    Dynamic_Address_Base <= Proc_RAM.enter_temp;
    return_state <= enter_humid;
    current_state <= h_data_add_subprocess_setup;
    base_display_data_address <= Proc_RAM.enter_temp;

-- Setup the memory addressed for entering humidity then run
    sub-routine
when enter_humid =>
    Static_Address_Base  <= Const_RAM.enter_humid;
    Dynamic_Address_Base <= Proc_RAM.enter_humid;
    return_state <= next_point;
    current_state <= h_data_add_subprocess_setup;
    base_display_data_address <= Proc_RAM.enter_humid;

-- Prompt the user to see if there is another step in the
    proofing run
when next_point =>
    Static_Address_Base  <= Const_RAM.next_point;
    Dynamic_Address_Base <= Proc_RAM.next_point;
    if pulse = '1' then
        case (button) is
            -- If 'a' is pressed then add another data point
            when x"a" =>
                current_state <= h_display_ram_clear_setup;
            -- If 'b' is pressed then move on to the run state
            when x"b" =>
                current_state <= run_setup;
            when others =>
                null;
        end case;
    end if;

-- Setup values for reading run
when run_setup =>
    Static_Address_Base  <= Const_RAM.run;
    Dynamic_Address_Base <= Proc_RAM.run;
    base_display_data_address <= Proc_RAM.run;
    base_run_data_address <= 0;
    run_sub_counter <= 0;
    current_state <= run_read_time;
    run_stage <= x"0001";
```

```
-- Read in the active time
when run_read_time =>
    -- reset the minute counter
    minute_count := 0;
    run_sub_counter <= run_sub_counter + 1;
    case (run_sub_counter) is
        when 0 =>
            current_run_data_address <= base_run_data_address;
            RunDataRAM_In.EN <= '1';
            RunDataRAM_In.WE <= '0';
            RunDataRAM_In.ADDR <= std_logic_vector(to_unsigned
                (base_run_data_address,11));
        when 2 =>
            run_time <= x"00" & RunDataRAM_Out.DO;
            current_run_data_address <=
                current_run_data_address + 1;
            RunDataRAM_In.ADDR <= std_logic_vector(to_unsigned
                (current_run_data_address + 1,11));
        when 4 =>
            run_time <=  RunDataRAM_Out.DO & run_time(7 downto
                 0);
            current_run_data_address <=
                current_run_data_address + 1;
            RunDataRAM_In.ADDR <= std_logic_vector(to_unsigned
                (current_run_data_address + 1,11));
        when 5 =>
            if (run_time = x"0000") then
                current_state <= alert;
                run_stage <= x"0000";
            end if;
        when 6 =>
            current_state <= run_read_temp;
            run_sub_counter <= 0;
            run_time_remaining <= run_time;
        when others =>
            null;
    end case;

-- Read in the active temperature
when run_read_temp =>
    run_sub_counter <= run_sub_counter + 1;
    case (run_sub_counter) is
        when 0 =>
            run_temp <= x"00" & RunDataRAM_Out.DO;
            current_run_data_address <=
                current_run_data_address + 1;
            RunDataRAM_In.ADDR <= std_logic_vector(to_unsigned
                (current_run_data_address + 1,11));
        when 2 =>
            run_temp <=  RunDataRAM_Out.DO & run_temp(7 downto
                 0);
            current_run_data_address <=
                current_run_data_address + 1;
            RunDataRAM_In.ADDR <= std_logic_vector(to_unsigned
```

```
                (current_run_data_address + 1,11));
            when 3 =>
                current_state <= run_read_humid;
                run_sub_counter <= 0;
            when others =>
                null;
        end case;

    -- Read in the active humidity
    when run_read_humid =>
        run_sub_counter <= run_sub_counter + 1;
        case (run_sub_counter) is
            when 0 =>
                run_humid <= x"00" & RunDataRAM_Out.DO;
                current_run_data_address <=
                    current_run_data_address + 1;
                RunDataRAM_In.ADDR <= std_logic_vector(to_unsigned
                    (current_run_data_address + 1,11));
            when 2 =>
                run_humid <=  RunDataRAM_Out.DO & run_humid(7
                    downto 0);
                current_run_data_address <=
                    current_run_data_address + 1;
                RunDataRAM_In.ADDR <= std_logic_vector(to_unsigned
                    (current_run_data_address + 1,11));
            when 3 =>
                current_state <= run;
                RunDataRAM_In.EN <= '0';
                run_sub_counter <= 0;
                base_run_data_address <= current_run_data_address;
            when others =>
                null;
        end case;

    -- Run state is used to while the proofing run is active
    when run =>
        -- Update the minute counter
        if (clk_1hz = '1') then
            minute_count := minute_count + 1;
        end if;

        -- Update the time remaining
        if (minute_count >= 60) then
            minute_count := 0;
            run_time_remaining <= run_time_remaining - 1;
        end if;

        -- Update the Outputs
        active_time <= run_time;
        active_temp <= run_temp;
        active_humid <= run_humid;
        active_time_remaining <= run_time_remaining;
        run_enable <= '1';
```

```
        -- Update the current values displayed on update pulse
        if (display_update_pulse = '1') then
            current_state <= run_update_display;
        end if;

        -- When the time is over sound the alarm
        if (run_time_remaining = x"0000") then
            current_state <= run_read_time;
            run_enable <= '0';
            run_stage <= run_stage + 1;
        end if;

        -- End run early if E is pressed.
        if pulse = '1' then
            case (button) is
                -- If 'e' is pressed then end run
                when x"e" =>
                    current_state <= h_both_ram_clear_setup;
                    run_enable <= '0';
                    run_stage <= x"0000";
                when others =>
                    null;
            end case;
        end if;

    -- Update the character display while in the run step
    when run_update_display =>
        run_sub_counter <= run_sub_counter + 1;
        case (run_sub_counter) is
            when 0 =>
                current_display_data_address <=
                    base_display_data_address;

            -- Write temperature
            when 1 =>
                dd_bin <= curr_temperature;
                bcd_index <= 4;
                run_leading <= '1';
            -- Write 'T'
            when 2 =>
                DisplayDataRAM_In.EN <= '1';
                DisplayDataRAM_In.WE <= '1';
                current_display_data_address <=
                    current_display_data_address+1;
                DisplayDataRAM_In.ADDR <= std_logic_vector(
                    to_unsigned(current_display_data_address,11));
                DisplayDataRAM_In.DI <= x"54";
            -- Write ':'
            when 3 =>
                current_display_data_address <=
                    current_display_data_address+1;
                DisplayDataRAM_In.ADDR <= std_logic_vector(
                    to_unsigned(current_display_data_address,11));
                DisplayDataRAM_In.DI <= x"3A";
```

```
when 4 =>
    bcd_index <= bcd_index - 1;
    current_display_data_address <=
        current_display_data_address+1;
    DisplayDataRAM_In.ADDR <= std_logic_vector(
        to_unsigned(current_display_data_address,11));
    if (run_leading = '1' and dd_bcd(bcd_index) = x"0"
        ) then
        DisplayDataRAM_In.DI <= x"20";
        current_display_data_address <=
            current_display_data_address;
    else
        run_leading <= '0';
        DisplayDataRAM_In.DI <= dd_bcd(bcd_index) + x"
            30";
    end if;
    if (bcd_index /= 0) then
        run_sub_counter <= run_sub_counter;
    end if;

-- Write space
when 5 =>
    current_display_data_address <=
        current_display_data_address+1;
    DisplayDataRAM_In.ADDR <= std_logic_vector(
        to_unsigned(current_display_data_address,11));
    DisplayDataRAM_In.DI <= x"20";
    if (current_display_data_address /=
        base_display_data_address + 8) then
        run_sub_counter <= 5;
    end if;
when 6 =>
    DisplayDataRAM_In.EN <= '0';
    DisplayDataRAM_In.WE <= '0';

-- Write Humidity
when 7 =>
    dd_bin <= curr_humidity;
    bcd_index <= 4;
    run_leading <= '1';
-- 'H'
when 8 =>
    DisplayDataRAM_In.EN <= '1';
    DisplayDataRAM_In.WE <= '1';
    current_display_data_address <=
        current_display_data_address+1;
    DisplayDataRAM_In.ADDR <= std_logic_vector(
        to_unsigned(current_display_data_address,11));
    DisplayDataRAM_In.DI <= x"48";
-- ':'
when 9 =>
    current_display_data_address <=
        current_display_data_address+1;
    DisplayDataRAM_In.ADDR <= std_logic_vector(
```

```
            to_unsigned(current_display_data_address,11));
        DisplayDataRAM_In.DI <= x"3A";
    when 10 =>
        bcd_index <= bcd_index - 1;
        current_display_data_address <=
            current_display_data_address+1;
        DisplayDataRAM_In.ADDR <= std_logic_vector(
            to_unsigned(current_display_data_address,11));
        if (run_leading = '1' and dd_bcd(bcd_index) = x"0"
            ) then
            DisplayDataRAM_In.DI <= x"20";
            current_display_data_address <=
                current_display_data_address;
        else
            run_leading <= '0';
            DisplayDataRAM_In.DI <= dd_bcd(bcd_index) + x"
                30";
        end if;
        if (bcd_index /= 0) then
            run_sub_counter <= run_sub_counter;
        end if;

    -- Write space
    when 11 =>
        current_display_data_address <=
            current_display_data_address+1;
        DisplayDataRAM_In.ADDR <= std_logic_vector(
            to_unsigned(current_display_data_address,11));
        DisplayDataRAM_In.DI <= x"20";
        if (current_display_data_address /=
            base_display_data_address + 15) then
            run_sub_counter <= 11;
        end if;
    when 12 =>
        DisplayDataRAM_In.EN <= '0';
        DisplayDataRAM_In.WE <= '0';

    -- Write stage
    when 13 =>
        run_leading <= '1';
        dd_bin <= run_stage;
        bcd_index <= 4;

    -- Write 'S'
    when 14 =>
        DisplayDataRAM_In.EN <= '1';
        DisplayDataRAM_In.WE <= '1';
        current_display_data_address <=
            current_display_data_address+1;
        DisplayDataRAM_In.ADDR <= std_logic_vector(
            to_unsigned(current_display_data_address,11));
        DisplayDataRAM_In.DI <= x"53";
    -- Write ':'
    when 15 =>
```

```
            current_display_data_address <=
                current_display_data_address+1;
            DisplayDataRAM_In.ADDR <= std_logic_vector(
                to_unsigned(current_display_data_address,11));
            DisplayDataRAM_In.DI <= x"3A";
        when 16 =>
            bcd_index <= bcd_index - 1;
            current_display_data_address <=
                current_display_data_address+1;
            DisplayDataRAM_In.ADDR <= std_logic_vector(
                to_unsigned(current_display_data_address,11));
            if (run_leading = '1' and dd_bcd(bcd_index) = x"0"
                ) then
                DisplayDataRAM_In.DI <= x"20";
                current_display_data_address <=
                    current_display_data_address;
            else
                run_leading <= '0';
                DisplayDataRAM_In.DI <= dd_bcd(bcd_index) + x"
                    30";
            end if;
            if (bcd_index /= 0) then
                run_sub_counter <= run_sub_counter;
            end if;

        -- Write space
        when 17 =>
            current_display_data_address <=
                current_display_data_address+1;
            DisplayDataRAM_In.ADDR <= std_logic_vector(
                to_unsigned(current_display_data_address,11));
            DisplayDataRAM_In.DI <= x"20";
            if (current_display_data_address /=
                base_display_data_address + (8+16)) then
                run_sub_counter <= 17;
            end if;
        when 18 =>
            DisplayDataRAM_In.EN <= '0';
            DisplayDataRAM_In.WE <= '0';
            run_leading <= '1';

        -- Write time remaining
        when 19 =>
            dd_bin <= run_time_remaining;
            bcd_index <= 4;
        -- 'R'
        when 20 =>
            DisplayDataRAM_In.EN <= '1';
            DisplayDataRAM_In.WE <= '1';
            current_display_data_address <=
                current_display_data_address+1;
            DisplayDataRAM_In.ADDR <= std_logic_vector(
                to_unsigned(current_display_data_address,11));
            DisplayDataRAM_In.DI <= x"52";
```

```
            -- ':'
            when 21 =>
                current_display_data_address <=
                    current_display_data_address+1;
                DisplayDataRAM_In.ADDR <= std_logic_vector(
                    to_unsigned(current_display_data_address,11));
                DisplayDataRAM_In.DI <= x"3A";
            when 22 =>
                bcd_index <= bcd_index - 1;
                current_display_data_address <=
                    current_display_data_address+1;
                DisplayDataRAM_In.ADDR <= std_logic_vector(
                    to_unsigned(current_display_data_address,11));
                if (run_leading = '1' and dd_bcd(bcd_index) = x"0"
                    ) then
                    DisplayDataRAM_In.DI <= x"20";
                    current_display_data_address <=
                        current_display_data_address;
                else
                    run_leading <= '0';
                    DisplayDataRAM_In.DI <= dd_bcd(bcd_index) + x"
                        30";
                end if;
                if (bcd_index /= 0) then
                    run_sub_counter <= run_sub_counter;
                end if;
             -- Write space
             when 23 =>
                current_display_data_address <=
                    current_display_data_address+1;
                DisplayDataRAM_In.ADDR <= std_logic_vector(
                    to_unsigned(current_display_data_address,11));
                DisplayDataRAM_In.DI <= x"20";
                if (current_display_data_address /=
                    base_display_data_address + 32) then
                    run_sub_counter <= 23;
                end if;
            -- Done
            -- Return to run
            when 24 =>
                DisplayDataRAM_In.EN <= '0';
                DisplayDataRAM_In.WE <= '0';
                current_state <= run;
                run_sub_counter <= 0;

            when others =>
                null;
        end case;

    -- Sound an alarm for the bread being ready
    when alert =>
        Static_Address_Base  <= Const_RAM.alert;
        Dynamic_Address_Base <= Proc_RAM.alert;
        if pulse = '1' then
```

```
            current_state <= h_both_ram_clear_setup;
        end if;

    when h_data_add_subprocess_setup =>
        current_display_data_address <= base_display_data_address;
        current_state <= h_data_add_subprocess;

    -- Subroutine used to copy keypresses into the character
       memory
    when h_data_add_subprocess =>
        DisplayDataRAM_In.EN <= '0';

        -- process keypad entry on pulse
        if pulse = '1' then
            case (button) is
                when x"0" =>
                    if (current_display_data_address <
                        base_display_data_address + 5) then
                        DisplayDataRAM_In.EN <= '1';
                        DisplayDataRAM_In.DI <= x"30";
                        DisplayDataRAM_In.WE <= '1';
                        DisplayDataRAM_In.ADDR <= std_logic_vector
                            (to_unsigned(
                            current_display_data_address, 11));
                        current_display_data_address <=
                            current_display_data_address + 1;
                    end if;
                when x"1" =>
                    if (current_display_data_address <
                        base_display_data_address + 5) then
                        DisplayDataRAM_In.EN <= '1';
                        DisplayDataRAM_In.DI <= x"31";
                        DisplayDataRAM_In.WE <= '1';
                        DisplayDataRAM_In.ADDR <= std_logic_vector
                            (to_unsigned(
                            current_display_data_address, 11));
                        current_display_data_address <=
                            current_display_data_address + 1;
                    end if;
                when x"2" =>
                    if (current_display_data_address <
                        base_display_data_address + 5) then
                        DisplayDataRAM_In.EN <= '1';
                        DisplayDataRAM_In.DI <= x"32";
                        DisplayDataRAM_In.WE <= '1';
                        DisplayDataRAM_In.ADDR <= std_logic_vector
                            (to_unsigned(
                            current_display_data_address, 11));
                        current_display_data_address <=
                            current_display_data_address + 1;
                    end if;
                when x"3" =>
                    if (current_display_data_address <
                        base_display_data_address + 5) then
```

```
            DisplayDataRAM_In.EN <= '1';
            DisplayDataRAM_In.DI <= x"33";
            DisplayDataRAM_In.WE <= '1';
            DisplayDataRAM_In.ADDR <= std_logic_vector
                (to_unsigned(
                current_display_data_address, 11));
            current_display_data_address <=
                current_display_data_address + 1;
        end if;
    when x"4" =>
        if (current_display_data_address <
            base_display_data_address + 5) then
            DisplayDataRAM_In.EN <= '1';
            DisplayDataRAM_In.DI <= x"34";
            DisplayDataRAM_In.WE <= '1';
            DisplayDataRAM_In.ADDR <= std_logic_vector
                (to_unsigned(
                current_display_data_address, 11));
            current_display_data_address <=
                current_display_data_address + 1;
        end if;
    when x"5" =>
        if (current_display_data_address <
            base_display_data_address + 5) then
            DisplayDataRAM_In.EN <= '1';
            DisplayDataRAM_In.DI <= x"35";
            DisplayDataRAM_In.WE <= '1';
            DisplayDataRAM_In.ADDR <= std_logic_vector
                (to_unsigned(
                current_display_data_address, 11));
            current_display_data_address <=
                current_display_data_address + 1;
        end if;
    when x"6" =>
        if (current_display_data_address <
            base_display_data_address + 5) then
            DisplayDataRAM_In.EN <= '1';
            DisplayDataRAM_In.DI <= x"36";
            DisplayDataRAM_In.WE <= '1';
            DisplayDataRAM_In.ADDR <= std_logic_vector
                (to_unsigned(
                current_display_data_address, 11));
            current_display_data_address <=
                current_display_data_address + 1;
        end if;
    when x"7" =>
        if (current_display_data_address <
            base_display_data_address + 5) then
            DisplayDataRAM_In.EN <= '1';
            DisplayDataRAM_In.DI <= x"37";
            DisplayDataRAM_In.WE <= '1';
            DisplayDataRAM_In.ADDR <= std_logic_vector
                (to_unsigned(
                current_display_data_address, 11));
```

```
                    current_display_data_address <=
                        current_display_data_address + 1;
                end if;
            when x"8" =>
                if (current_display_data_address <
                    base_display_data_address + 5) then
                    DisplayDataRAM_In.EN <= '1';
                    DisplayDataRAM_In.DI <= x"38";
                    DisplayDataRAM_In.WE <= '1';
                    DisplayDataRAM_In.ADDR <= std_logic_vector
                        (to_unsigned(
                        current_display_data_address, 11));
                    current_display_data_address <=
                        current_display_data_address + 1;
                end if;
            when x"9" =>
                if (current_display_data_address <
                    base_display_data_address + 5) then
                    DisplayDataRAM_In.EN <= '1';
                    DisplayDataRAM_In.DI <= x"39";
                    DisplayDataRAM_In.WE <= '1';
                    DisplayDataRAM_In.ADDR <= std_logic_vector
                        (to_unsigned(
                        current_display_data_address, 11));
                    current_display_data_address <=
                        current_display_data_address + 1;
                end if;
            -- Accept the number
            when x"a" =>
                current_state <=
                    h_run_data_add_subprocess_setup;
            -- Delete one character
            when x"b" =>
                if (current_display_data_address >
                    base_display_data_address) then
                    current_display_data_address <=
                        current_display_data_address - 1;
                    current_state <=
                        h_data_add_subprocess_clear;
                end if;
            when x"c" =>
                null;
            when x"d" =>
                null;
            when x"e" =>
                null;
            when x"f" =>
                null;
            when others =>
                null;
        end case;
    end if;

-- Setup for h_run_data_add_subprocess
```

```
when h_run_data_add_subprocess_setup =>
    -- read from the display ram into dcd
    DisplayDataRAM_In.EN <= '1';
    DisplayDataRAM_In.WE <= '0';
    DisplayDataRAM_In.ADDR <= std_logic_vector(to_unsigned(
        base_display_data_address, 11));
    current_display_data_address <= base_display_data_address;
    bcd_index <= 5;
    bcd_2_bin_bcd <= (others => (others => '0'));
    current_state <= h_run_data_add_subprocess_wait;

-- Read the numbers from the character display, and store it
    as a binary coded decimal
when h_run_data_add_subprocess =>
    -- read from the display ram into dcd
    DisplayDataRAM_In.ADDR <= std_logic_vector(to_unsigned(
        current_display_data_address+1, 11));
    current_display_data_address <=
        current_display_data_address + 1;
    bcd_index <= bcd_index - 1;

    -- Check for empty char and end of number
    if (DisplayDataRAM_Out.DO = x"00" or bcd_index = 0) then
        current_state <= h_run_data_add_subprocess_bcd2bin;
    else
        tmp := DisplayDataRAM_Out.DO - x"30";
        for i in 4 downto 1 loop
            bcd_2_bin_bcd(i) <= bcd_2_bin_bcd(i-1);
        end loop;
        bcd_2_bin_bcd(0) <= tmp(3 downto 0);
        current_state <= h_run_data_add_subprocess_wait;
    end if;

-- Extra cycle needed to give the display ram time to output
    the number
when h_run_data_add_subprocess_wait =>
    current_state <= h_run_data_add_subprocess;

-- Cycle for the bcd2bin to convert
when h_run_data_add_subprocess_bcd2bin =>
    current_state <= h_run_data_add_subprocess_write;
    current_run_data_address <= base_run_data_address;
    base_run_data_address <= base_run_data_address + 2;
    bin_index <= 0;

-- Write the binary coded decimal to the run ram
when h_run_data_add_subprocess_write =>
    if bin_index = 1 then
        current_state <= h_run_data_add_subprocess_post;
        RunDataRAM_In.WE <= '0';
        RunDataRAM_In.EN <= '0';
    else
        current_run_data_address <= current_run_data_address +
            1;
```

```
        end if;
         case (bin_index) is
             when 0 => RunDataRAM_In.DI <= bcd_2_bin_bin(7 downto
                 0);
             when 1 => RunDataRAM_In.DI <= bcd_2_bin_bin(15 downto
                  8);
             when others => RunDataRAM_In.DI <= x"00";
         end case;
        bin_index <= bin_index + 1;
        RunDataRAM_In.EN <= '1';
        RunDataRAM_In.WE <= '1';
        RunDataRAM_In.ADDR <= std_logic_vector(to_unsigned(
            current_run_data_address, 11));

    when h_run_data_add_subprocess_post =>
        current_state <= return_state;

    -- State used to clear a char from the display memory
    when h_data_add_subprocess_clear =>
        DisplayDataRAM_In.EN <= '1';
        DisplayDataRAM_In.DI <= x"00";
        DisplayDataRAM_In.WE <= '1';
        DisplayDataRAM_In.ADDR <= std_logic_vector(to_unsigned(
            current_display_data_address, 11));
        current_state <= h_data_add_subprocess;

    when h_display_ram_clear_setup =>
        current_display_data_address <= 0;
        current_state <= h_display_ram_clear;

    -- Reset the display RAM to all zeros
    when h_display_ram_clear =>
        DisplayDataRAM_In.EN <= '1';
        DisplayDataRAM_In.DI <= x"00";
        DisplayDataRAM_In.WE <= '1';
        DisplayDataRAM_In.ADDR <= std_logic_vector(to_unsigned(
            current_display_data_address, 11));
        current_display_data_address <=
            current_display_data_address + 1;
        if (current_display_data_address = 2047) then
            current_state <= h_display_ram_clear_post;
        end if;

    when h_display_ram_clear_post =>
        DisplayDataRAM_In.EN <= '0';
        current_state <= enter_time;

    when h_both_ram_clear_setup =>
        current_display_data_address <= 0;
        current_state <= h_both_ram_clear;

    -- Reset both rams to all zeros
    when h_both_ram_clear =>
        DisplayDataRAM_In.EN <= '1';
```

```
                    DisplayDataRAM_In.DI <= x"00";
                    DisplayDataRAM_In.WE <= '1';
                    DisplayDataRAM_In.ADDR <= std_logic_vector(to_unsigned(
                        current_display_data_address, 11));
                    RunDataRAM_In.EN <= '1';
                    RunDataRAM_In.DI <= x"00";
                    RunDataRAM_In.WE <= '1';
                    RunDataRAM_In.ADDR <= std_logic_vector(to_unsigned(
                        current_display_data_address, 11));
                    current_display_data_address <=
                        current_display_data_address + 1;
                    if (current_display_data_address = 2047) then
                        current_state <= h_both_ram_clear_post;
                    end if;

                when h_both_ram_clear_post =>
                    DisplayDataRAM_In.EN <= '0';
                    RunDataRAM_In.EN <= '0';
                    current_state <= main;

                when others =>
                    current_state <= main;
            end case;
        end if;
    end process;

    -- Set the constant bits of the RAM
    -- This design does not use parity bits
    DisplayDataRAM_In.DIP <= (others => '0');
    RunDataRAM_In.DIP <= (others => '0');

    -- Pulser logic is used to generate a pulse when the keypad is first
        pressed down
    pulser : process (clk) begin
        if (rising_edge(clk)) then
            if (previous_pressed = '0' and pressed = '1') then
                pulse <= '1';
            else
                pulse <= '0';
            end if;

            previous_pressed <= pressed;
        end if;
    end process;

end Behavioral;
```

A.3 Output Control Module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity output_control is
    Port ( clk : in STD_LOGIC;
           rst : in std_logic;
           curr_temperature : in STD_LOGIC_VECTOR (15 downto 0);
           curr_humidity : in STD_LOGIC_VECTOR (15 downto 0);
           set_temperature : in std_logic_vector(15 downto 0);
           set_humidity : in std_logic_vector(15 downto 0);
           enable : in std_logic;
           heat_control : out STD_LOGIC;
           humid_control : out STD_LOGIC);
end output_control;

architecture Behavioral of output_control is

    component CDiv
            GENERIC (
               TC : integer := 18 --Time Constant. 15 for ~1khz --
                   100000000/(TC^4) Hz
            );
            PORT (
               Cin   : IN  STD_LOGIC;
               Reset : IN STD_LOGIC;
               Cout : OUT STD_LOGIC);
    end component;

    signal humid_fast : std_logic;
    signal humid_slow : std_logic;
    signal temp_fast : std_logic;
    signal temp_slow : std_logic;

    signal average_sum_temp : integer range 0 to 50 := 25;
    signal average_sum_humid : integer range 0 to 50 := 25;
    constant average_sum_threshold : integer range 0 to 50 := 25;
begin

    tslow : CDiv
        generic map(
            TC => 196)
        port map(
            Cin => clk,
            Reset => '0',
            Cout => temp_slow);

    tfast : CDiv
        generic map(
            TC => 110)
        port map(
```

```
        Cin => clk,
        Reset => '0',
        Cout => temp_fast);

hslow : CDiv
    generic map(
        TC => 100)
    port map(
        Cin => clk,
        Reset => '0',
        Cout => humid_slow);

hfast : CDiv
    generic map(
        TC => 56)
    port map(
        Cin => clk,
        Reset => '0',
        Cout => humid_fast);

set_average : process(clk, rst)
begin
    if (rising_edge(clk)) then
        if(rst = '1') then
            average_sum_temp <= 25;
            average_sum_humid <= 25;
        else
            if(temp_fast = '1') then
                if(to_integer(unsigned(set_temperature)) <= to_integer(
                    unsigned(curr_temperature))) then
                    if(average_sum_temp > 0) then
                        average_sum_temp <= average_sum_temp - 1;
                    end if;
                elsif(to_integer(unsigned(set_temperature)) > to_integer(
                    unsigned(curr_temperature))) then
                    if(average_sum_temp < 50) then
                        average_sum_temp <= average_sum_temp + 1;
                    end if;
                end if;
            end if;

            if (humid_fast = '1') then
                if(to_integer(unsigned(set_humidity)) <= to_integer(
                    unsigned(curr_humidity))) then
                    if(average_sum_humid > 0) then
                        average_sum_humid <= average_sum_humid - 1;
                    end if;
                elsif(to_integer(unsigned(set_humidity)) > to_integer(
                    unsigned(curr_humidity))) then
                    if(average_sum_humid < 50) then
                        average_sum_humid <= average_sum_humid + 1;
                    end if;
                end if;
            end if;
```

```
            end if;
        end if;
    end process;

    set_output : process(clk, rst, enable)
    begin
        if (rising_edge(clk)) then
            if((rst = '1') or (enable = '0')) then
                heat_control <= '0';
                humid_control <= '0';
            elsif(enable = '1') then
                if(temp_slow = '1') then
                    if(average_sum_temp > average_sum_threshold) then
                        heat_control <= '1';
                    elsif(average_sum_temp <= average_sum_threshold) then
                        heat_control <= '0';
                    end if;
                end if;
                if (humid_slow = '1') then
                    if(average_sum_humid > average_sum_threshold) then
                        humid_control <= '1';
                    elsif(average_sum_humid <= average_sum_threshold) then
                        humid_control <= '0';
                    end if;
                end if;
            end if;
        end if;
    end process;



end Behavioral;
```

Appendix B Testbench VHDL Code

B.1 Proofing State Machine Test Bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library work;
use work.common.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Proofing_State_t is
--  Port ( );
end Proofing_State_t;

architecture Behavioral of Proofing_State_t is
    component Proofing_State
    Port (
          CLK,                                                    -- System
              clock
          reset_n : in std_logic;                                 -- Reset
          display_update_pulse : in std_logic;                    -- pulse to
              update run display
          clk_1hz : in std_logic;                                 -- One hertz
              clock used to update the timer
          button : in std_logic_vector (3 downto 0);              -- Button
              press from keypad
          pressed : in std_logic;                                 -- Pressed
              from keypad
          curr_humidity : in std_logic_vector(15 downto 0);       -- Current
              humidity from the sensor
          curr_temperature : in std_logic_vector(15 downto 0);    -- Current
              temperature from the sensor
          Mode : out  t_proofer_state;                            -- Current
              mode of the state machine
          Static_Address_Base : out integer range 0 to 2047;      -- Base
              address for static display
          Dynamic_Address_Base : out integer range 0 to 2047;     -- Base
              address for dynamic display
          DisplayDataRAM_In : out  T_RAM_Port_In;                 -- Display
              Data Ram
          DisplayDataRAM_Out : in  T_RAM_Port_Out;                -- Display
              Data Ram
```

```
            RunDataRAM_In : out  T_RAM_Port_In;                    -- Run Data
                Ram
            RunDataRAM_Out : in  T_RAM_Port_Out;                   -- Run Data
                Ram
            buzzer_enable : out std_logic;                         -- Enable the
                buzzer
            active_time : out std_logic_vector(15 downto 0);       -- Active Time
            active_time_remaining : out std_logic_vector(15 downto 0); --
                Remaining time for the run
            active_temp : out std_logic_vector(15 downto 0);       -- Active
                Temperature
            active_humid : out std_logic_vector(15 downto 0);      -- Active
                Humidity
            active_stage : out std_logic_vector(15 downto 0);      -- Active
                stage
            run_enable : out std_logic                             -- Is the
                device running?
        );
    end component;

    -- Inputs
    signal reset_n : std_logic := '1';
    signal button : std_logic_vector (3 downto 0) := "0000";
    signal pressed : std_logic := '0';
    signal DisplayDataRAM_Out : T_RAM_Port_Out;                -- Display Data
        Ram
    signal RunDataRAM_Out : T_RAM_Port_Out;                    -- Run Data Ram
    signal curr_humidity : std_logic_vector(15 downto 0);
    signal curr_temperature : std_logic_vector(15 downto 0);
    signal display_update_pulse : std_logic := '0';
    signal clk_1hz : std_logic := '0';

    -- Outputs
    signal Mode : t_proofer_state;
    signal Static_Address_Base : integer range 0 to 2047;
    signal Dynamic_Address_Base : integer range 0 to 2047;
    signal DisplayDataRAM_In :  T_RAM_Port_In;
    signal RunDataRAM_In :  T_RAM_Port_In;
    signal active_time : std_logic_vector(15 downto 0);
    signal active_time_remaining : std_logic_vector(15 downto 0);
    signal active_temp : std_logic_vector(15 downto 0);
    signal active_humid : std_logic_vector(15 downto 0);
    signal active_stage : std_logic_vector(15 downto 0);
    signal buzzer_enable : std_logic;
    signal run_enable : std_logic;

    -- Clock
    signal clk_100 : std_logic;
    constant clk_period : time := 10 ns;

begin

    curr_humidity <= x"0053";
    curr_temperature <= x"0021";
```

```
clock : process
begin
    clk_100 <= '0';
    wait for clk_period/2;
    clk_100 <= '1';
    wait for clk_period/2;
end process;

ram : process(clk_100)
begin
    if (rising_edge(clk_100))then
        case (DisplayDataRAM_In.Addr) is
            when std_logic_vector(to_unsigned(16#40#, 11)) =>
                DisplayDataRAM_Out.DO <= x"36";
            when std_logic_vector(to_unsigned(16#41#, 11)) =>
                DisplayDataRAM_Out.DO <= x"35";
            when std_logic_vector(to_unsigned(16#42#, 11)) =>
                DisplayDataRAM_Out.DO <= x"35";
            when std_logic_vector(to_unsigned(16#43#, 11)) =>
                DisplayDataRAM_Out.DO <= x"33";
            when std_logic_vector(to_unsigned(16#44#, 11)) =>
                DisplayDataRAM_Out.DO <= x"35";
            when others =>
                DisplayDataRAM_Out.DO <= x"00";
        end case;
    end if;
end process;

RunDataRAM_Out.DO <= x"01";

test : process
begin
    -- Reset board
    reset_n <= '0';
    wait for clk_period;
    reset_n <= '1';
    wait for 10*clk_period;

    -- Press key
    button <= x"1";
    pressed <= '1';
    wait for clk_period;
    pressed <= '0';
    wait for 10*clk_period;

    -- Press 1 for time
    button <= x"1";
    pressed <= '1';
    wait for clk_period;
    pressed <= '0';
    wait for 10*clk_period;

    -- Press 5 for time
```

```
button <= x"5";
pressed <= '1';
wait for clk_period;
pressed <= '0';
wait for 10*clk_period;

-- Press 0 for time
button <= x"0";
pressed <= '1';
wait for clk_period;
pressed <= '0';
wait for 10*clk_period;

-- Press A for time
button <= x"a";
pressed <= '1';
wait for clk_period;
pressed <= '0';
wait for 40*clk_period;

-- Press 2 for temp
button <= x"2";
pressed <= '1';
wait for clk_period;
pressed <= '0';
wait for 10*clk_period;

-- Press A for time
button <= x"a";
pressed <= '1';
wait for clk_period;
pressed <= '0';
wait for 40*clk_period;

-- Press 3 for humid
button <= x"3";
pressed <= '1';
wait for clk_period;
pressed <= '0';
wait for 10*clk_period;

-- Press A for humid
button <= x"a";
pressed <= '1';
wait for clk_period;
pressed <= '0';
wait for 40*clk_period;

-- Press B for stop adding
button <= x"b";
pressed <= '1';
wait for clk_period;
pressed <= '0';
wait for 40*clk_period;
```

```
        -- Signal Display Pulse
        display_update_pulse <= '1';
        wait for clk_period;
        display_update_pulse <= '0';
        wait for 60*clk_period;

        -- 1 Hz Pulse
        clk_1hz <= '1';
        wait for clk_period;
        clk_1hz <= '0';
        wait for 40*clk_period;
        wait;

    end process;

    UUT : Proofing_State
    Port map (
           CLK                   => CLK_100,
           reset_n               => reset_n,
           display_update_pulse  => display_update_pulse,
           clk_1hz               => clk_1hz,
           button                => button,
           pressed               => pressed,
           curr_humidity         => curr_humidity,
           curr_temperature      => curr_temperature,
           Mode                  => Mode,
           Static_Address_Base   => Static_Address_Base,
           Dynamic_Address_Base  => Dynamic_Address_Base,
           DisplayDataRAM_In     => DisplayDataRAM_In,
           DisplayDataRAM_Out    => DisplayDataRAM_Out,
           RunDataRAM_In         => RunDataRAM_In,
           RunDataRAM_Out        => RunDataRAM_Out,
           buzzer_enable         => buzzer_enable,
           active_time           => active_time,
           active_time_remaining => active_time_remaining,
           active_temp           => active_temp,
           active_humid          => active_humid,
           active_stage          => active_stage,
           run_enable            => run_enable
       );
end Behavioral;
```

B.2 Output Control Module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity output_control_tb is
--  Port ( );
end output_control_tb;

architecture tbench of output_control_tb is

    signal clk : std_logic;
    signal rst : std_logic;
    signal current_temperature : std_logic_vector(15 downto 0);
    signal current_humidity : std_logic_vector(15 downto 0);
    signal set_temperature_threshold : std_logic_vector(15 downto 0) := "
       0000000001010100"; --84*F
    signal set_humidity_threshold : std_logic_vector(15 downto 0) := "
       0000000000111000"; -- 56%
    signal enable_output : std_logic := '1';
    signal heat_control_out : std_logic;
    signal humidity_control_out : std_logic;

    component output_control is
        Port (
                clk : in STD_LOGIC;
                rst : in std_logic;
                curr_temperature : in STD_LOGIC_VECTOR (15 downto 0);
                curr_humidity : in STD_LOGIC_VECTOR (15 downto 0);
                set_temperature : in std_logic_vector(15 downto 0);
                set_humidity : in std_logic_vector(15 downto 0);
                enable : in std_logic;
                heat_control : out STD_LOGIC;
                humid_control : out STD_LOGIC
              );
    end component;

    -- Clock period declared and defined as a constant
    constant clk_period : time := 10 ns;

begin

    UUT : output_control
    Port map (
                clk => clk,
                rst => rst,
                curr_temperature => current_temperature,
                curr_humidity => current_humidity,
                set_temperature => set_temperature_threshold,
                set_humidity => set_humidity_threshold,
                enable => enable_output,
                heat_control => heat_control_out,
```

```
            humid_control => humidity_control_out
        );

    -- clock process definition with 50% duty cycle
    process
    begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    end process;

    -- stimulus process
    process
    begin
        rst <= '0';
        wait for clk_period;
        rst <= '1';
        wait for clk_period;
        rst <= '0';
        current_temperature <= "0000000001000110"; -- 70*F
        current_humidity <= "0000000000101101"; -- 45%
        wait for 1 sec;
        enable_output <='0';
        wait;
    end process;

end tbench;
```

Appendix C Web Interface Source Code

C.1 Web Interface Top-Level

```
from __future__ import division
import os
import database_functions as df
from flask import Flask, render_template, redirect, request, flash, Markup,
    session, jsonify
from flaskext.mysql import MySQL
import network_socket as ns
from time import time
from datetime import timedelta

################################
# Links app to FR database. #
################################
mysql = MySQL()
app = Flask(__name__)
app.config["MYSQL_DATABASE_USER"] = "root"
app.config["MYSQL_DATABASE_PASSWORD"] = "Brl9me8o"
app.config["MYSQL_DATABASE_DB"] = "bpb"
app.config["MYSQL_DATABASE_HOST"] = "localhost"
mysql.init_app(app)

temps = []
hums = []
times = []

################################
# Redirects to login screen. #
################################
@app.route("/")
def index():
    return redirect("/home")

def navbar():
    render_template("navbar.html")

#######################
# Loads Home screen. #
#######################
@app.route("/home")
def home():
    return render_template("home.html")

#################################################################
# Displays graph of data retrieved from current proof run. #
#################################################################
@app.route("/active")
def active():
    try:
        return render_template("active.html", at=ave_temp, max_t=max_temp,
```

```
                                    min_t=min_temp, ah=ave_hum, max_h=max_hum,
                                        min_h=min_hum, t=ts, dur=duration)
    except:
        return render_template("active.html", at="null", max_t="null", min_t="
            null",
                                    ah="null", max_h="null", min_h="null", t="null
                                        ", dur="null")

#########################################################
# Displays the previous runs for the current proof. #
#########################################################
@app.route("/stages")
def stages():
    if stages_done == "0":
        return render_template("stages.html", stages=int(stages_done))
    else:
        old_temps, old_hums, old_times = df.retrieve_stages(mysql, stages_done
            )
        ave_temp=[[] for x in xrange(len(old_temps))]
        max_temp=[[] for x in xrange(len(old_temps))]
        min_temp=[[] for x in xrange(len(old_temps))]
        ave_hum=[[] for x in xrange(len(old_hums))]
        max_hum=[[] for x in xrange(len(old_hums))]
        min_hum=[[] for x in xrange(len(old_hums))]
        dur=[[] for x in xrange(len(old_times))]
        for i in range(0, len(old_temps)):
            if (len(old_temps) != 0):
                ave_temp[i].append(sum(old_temps[i])/len(old_temps[i]))
                max_temp[i].append(max(old_temps[i]))
                min_temp[i].append(min(old_temps[i]))
                ave_hum[i].append(sum(old_hums[i])/len(old_hums[i]))
                max_hum[i].append(max(old_hums[i]))
                min_hum[i].append(min(old_hums[i]))
                dur[i].append(max(old_times[i]))
        if len(old_temps) == 3:
            return render_template("stages.html", stages=int(stages_done),
                temps_r1=old_temps[0], temps_r2=old_temps[1],
                                    temps_r3=old_temps[2], hums_r1=old_hums[0],
                                        hums_r2=old_hums[1],
                                    hums_r3=old_hums[2], times_r1=old_times[0],
                                        times_r2=old_times[1],
                                    times_r3=old_times[2], l3=len(old_temps[2]),
                                        l2=len(old_temps[1]), l1=len(old_temps[0])
                                        ,
                                    at=ave_temp, mt=max_temp, mint=min_temp, ah=
                                        ave_hum, mh=max_hum, minh=min_hum,
                                    dur=dur)
        elif len(old_temps) == 2:
            return render_template("stages.html", stages=int(stages_done),
                temps_r1=old_temps[0], temps_r2=old_temps[1],
                                     hums_r1=old_hums[0], hums_r2=old_hums[1],
                                         times_r1=old_times[0], times_r2=
                                         old_times[1],
                                     l1=len(old_temps[0]), l2=len(old_temps[1])
```

```
                                        , at=ave_temp, mt=max_temp, mint=
                                        min_temp,
                                    ah=ave_hum, mh=max_hum, minh=min_hum, dur=
                                        dur)
        elif len(old_temps) == 1:
            return render_template("stages.html", stages=int(stages_done),
                temps_r1=old_temps[0], hums_r1=old_hums[0],
                                    times_r1=old_times[0], l1=len(old_temps
                                        [0]), at=ave_temp, mt=max_temp, mint=
                                        min_temp,
                                    ah=ave_hum, mh=max_hum, minh=min_hum, dur=
                                        dur)


##############################################################################
############################ Not actual routes ################################
##############################################################################

##############################################################################
# Handles retrieving data form server. Constantly polling via Navbar.html #
##############################################################################
@app.route("/active/data", methods=['GET', 'POST'])
def active_data():
    global ave_temp
    global max_temp
    global min_temp
    global ave_hum
    global max_hum
    global min_hum
    global ts
    global duration
    global time
    global stage
    global stages_done
    navbar()
    try:
        print "Initializing connection to server.."
        socket = ns.init_socket()
        ns.connect(socket)
        print "Connection established.."
        socket.send("hey")
        data = socket.recv(32)
        if data:
            session["current"] = "1"
            if "current_stage" not in session:
                session["current_stage"] = "0"
            split_data = data.strip().split(",")
            print split_data
            stage = int(split_data[2])
            if stage == 0:
                if session["current_stage"] != "0":
                    times.append(times[-1]+1)
                    temps.append(int(split_data[0]))
                    hums.append(int(split_data[1]))
```

```
                    ave_temp = sum(temps)/len(temps)
                    max_temp = max(temps)
                    min_temp = min(temps)
                    ave_hum = sum(hums)/len(hums)
                    max_hum = max(hums)
                    min_hum = min(hums)
                    ts=duration
                    stages_done = str(int(stages_done)+1)
                return redirect("active/save")
            else:
                rem = int(split_data[3])
                duration = int(split_data[4])
                time = (duration-rem)
                print time
            stages_done = str(int(stage-1))
            if stage < int(session["current_stage"]):
                session["current_stage"] = str(stage)
            elif stage > int(session["current_stage"]):
                session["current_stage"] = str(stage)
                if session["current_stage"] != "1":
                    times.append(times[-1]+1)
                    temps.append(int(split_data[0]))
                    hums.append(int(split_data[1]))
                    ts=duration
                    return redirect("active/save")
            if time not in times:
                times.append(time)
                temps.append(int(split_data[0]))
                hums.append(int(split_data[1]))
            if (len(temps) != 0):
                ave_temp = sum(temps)/len(temps)
                max_temp = max(temps)
                min_temp = min(temps)
                ave_hum = sum(hums)/len(hums)
                max_hum = max(hums)
                min_hum = min(hums)
                ts = time
        else:
            session["current"] = "0"
        return jsonify(temp=temps, time=times, hum=hums, at=ave_temp, max_t=
            max_temp,
                       min_t=min_temp, ah=ave_hum, max_h=max_hum, min_h=
                           min_hum, t=ts,
                       dur=duration, stage=int(stages_done), current=int(
                           session["current"]))
    except:
        return ""


#########################################################
# Responsible for retrieving data from sidebar modals #
#########################################################
@app.route("/active/save", methods=['GET', 'POST'])
def active_save():
```

```
    if session["current_stage"] != "0":
        print "Saving stage..\n"
        temps_30 = []
        hums_30 = []
        times_30 = []
        if len(temps) <= 20:
            if times[0] not in times_30:
                times_30.append(int(times[0]))
                temps_30.append(temps[0])
                hums_30.append(hums[0])
            for j in xrange(0, len(temps)-1):
                times_30.append(int(times[j]))
                temps_30.append(temps[j])
                hums_30.append(hums[j])
            if times[len(times)-1] not in times_30:
                times_30.append(int(times[len(times)-1]))
                temps_30.append(temps[len(times)-1])
                hums_30.append(hums[len(times)-1])
        else:
            for j in xrange(1, len(temps)):
                if ((len(temps) > 20*j) & (len(temps) <= 20*(j+1))):
                    if times[0] not in times_30:
                        times_30.append(int(times[0]))
                        temps_30.append(temps[0])
                        hums_30.append(hums[0])
                    for i in xrange(0, len(temps), j+1):
                        times_30.append(int(times[i]))
                        temps_30.append(temps[i])
                        hums_30.append(hums[i])
                    if times[len(times)-1] not in times_30:
                        times_30.append(int(times[len(times)-1]))
                        temps_30.append(temps[len(times)-1])
                        hums_30.append(hums[len(times)-1])
        print temps_30, hums_30, times_30
        df.delete_stage(mysql, stages_done)
        df.create_stage(mysql, stages_done)
        df.save_stage(mysql, temps_30, hums_30, times_30, stages_done)
        if stage != 0:
            temps[:]=[]
            times[:]=[]
            hums[:]=[]
        print "Initiating new stage..\n"
    session["current_stage"] = str(stage)
    return jsonify(temp=temps, time=times, hum=hums, at=ave_temp, max_t=
        max_temp,
                   min_t=min_temp, ah=ave_hum, max_h=max_hum, min_h=min_hum, t
                       =ts,
                   dur=duration, stage=int(stages_done), current=int(session["
                       current"]))

####################################
# Main function where app is run. #
####################################
if __name__ == "__main__":
```

```
public = "10.131.41.63"
local = "127.0.0.1"
app.secret_key = "something"
app.run(debug=True, host=local)
```

C.2 Web Interface Database Code

```
import os
from flaskext.mysql import MySQL
from itertools import repeat
import string
from multiprocessing import Pool

##############################################
# Creates specified stage table in database. #
##############################################
def create_stage(mysql, stage):
    conn = mysql.connect()
    cursor = conn.cursor()
    query = "create table stage_" + stage + "(stageid varchar(3) not null,
        temps varchar(500) not null, hums varchar(500) not null, times varchar
        (500) not null, primary key(stageid));"
    cursor.execute(query)
    return conn.commit()

############################################################
# Queries database to return list of all project types. #
############################################################
def save_stage(mysql, temps, hums, times, stage):
    temps = ",".join(map(str, temps))
    hums = ",".join(map(str, hums))
    times = ",".join(map(str, times))
    conn = mysql.connect()
    cursor = conn.cursor()
    query = "insert into stage_" + stage + " values('" + stage + "', '" +
        temps + "', '" + hums + "', '" + times + "');"
    cursor.execute(query)
    return conn.commit()

##############################################
# Queries database to delete specified stage. #
##############################################
def delete_stage(mysql, stage):
    conn = mysql.connect()
    cursor = conn.cursor()
    query = "drop table if exists stage_" + stage + ";"
    cursor.execute(query)
    return conn.commit()

########################################
# Retreive specified temperature data. #
########################################
def retrieve_temps(stages, co, cu):
    temps = []
    for i in range(1, int(stages)+1):
        query = "select temps from stage_" + str(i) + ";"
        cu.execute(query)
        temps.append(cu.fetchall())
    return temps
```

```
#####################################
# Retrieve specified humidity data. #
#####################################
def retrieve_hums(stages, co, cu):
    hums = []
    for i in range(1, int(stages)+1):
        query = "select hums from stage_" + str(i) + ";"
        cu.execute(query)
        hums.append(cu.fetchall())
    return hums

#####################################
# Retrieve specified timing data. #
#####################################
def retrieve_times(stages, co, cu):
    times = []
    for i in range(1, int(stages)+1):
        query = "select times from stage_" + str(i) + ";"
        cu.execute(query)
        times.append(cu.fetchall())
    return times

##############################
# Convert strings ot floats #
##############################
def list_of_floats(a):
    try:
        a.remove('')
        return map(float, a)
    except:
        return map(float, a)

###########################################
# Retrieve current stage data in database. #
###########################################
def retrieve_stages(mysql, stages):
    new_temps = [[] for x in xrange(int(stages))]
    new_hums = [[] for x in xrange(int(stages))]
    new_times = [[] for x in xrange(int(stages))]
    conn = mysql.connect()
    cursor = conn.cursor()
    temps = retrieve_temps(stages, conn, cursor)
    hums = retrieve_hums(stages, conn, cursor)
    times = retrieve_times(stages, conn, cursor)
    table = string.maketrans( '', '', )
    for j in range(0, int(stages)):
        for i in range(0, len(str(list(temps[j])).strip().split(","))):
            new_temps[j].append(str(list(temps[j])).strip().split(",")[i].
                translate(table,"u()''[]"))
            new_hums[j].append(str(list(hums[j])).strip().split(",")[i].
                translate(table,"u()''[]"))
            new_times[j].append(str(list(times[j])).strip().split(",")[i].
                translate(table,"u()''[]"))
```

```
new_temps = Pool(4).map(list_of_floats, new_temps)
new_hums = Pool(4).map(list_of_floats, new_hums)
new_times = Pool(4).map(list_of_floats, new_times)
return new_temps, new_hums, new_times
```

Appendix D Recipe—Extra-Tangy Sourdough Bread

Hands-on time:	15 mins. to 20 mins.
Baking time:	30 mins.
Total time:	24 hrs.
Yield:	2 loaves

Ingredients:

- 1 cup "fed" sourdough starter
- 1 1/2 – 1 2/3 cups lukewarm water, enough to make a smooth dough
- 5 cups King Arthur Unbleached All-Purpose Flour
- 1 Tbsp sugar
- 2 1/4 tsp salt
- 1/2 – 5/8 tsp sour salt (citric acid), optional, for extra-sour bread

Method:

1. Combine the starter, water, and 3 cups of the flour. Beat vigorously for 1 minute.
2. Cover, and let rest at room temperature for 4 hours. Refrigerate overnight, for about 12 hours.
3. Add the remaining ingredients: 2 cups of flour, sugar, salt, and sour salt, if you're using it. Knead to form a smooth dough, about 10 minutes.
4. Allow the dough to rise in a covered bowl until it's relaxed, smoothed out, and risen. Depending on the vigor of your starter, it may become REALLY puffy, as pictured; or it may just rise a bit. This can take anywhere from 2 to 5 hours. Understand this: sourdough bread (especially sourdough without added yeast) is as much art as science; everyone's timetable will be different. So please allow yourself to go with the flow, and not treat this as an exact, to-the-minute process.
5. Gently divide the dough in half.
6. Gently shape the dough into two oval loaves, and place them on a lightly greased or parchment-lined baking sheet. Cover and let rise until very puffy, about 2 to 4 hours. Don't worry if the loaves spread more than they rise; they'll pick up once they hit the oven's heat. Towards the end of the rising time, preheat the oven to 425°F.
7. Spray the loaves with lukewarm water.
8. Make two fairly deep horizontal slashes in each; a serrated bread knife, wielded firmly, works well here.
9. Bake the bread for 25 to 30 minutes, until it's a very deep golden brown. Remove it from the oven, and cool on a rack.