COMPSYS723 Assignment 1: Final design documentation

Prabhnandan Singh (psin546)

Rodger Chen (rche647)

Application Instructions

1. Plug the power cable in the power port, the PS/2 keyboard in the PS/2 port, the VGA cable in the VGA port for the Altera DE2-115 board.
2. Connect the USB-Blaster cable to the USB-Blaster port.
3. Open Quartus Programmer and select USB-Blaster as the currently selected hardware in the Hardware setup window
4. Click on Add File button and select the given "freq_relay_controller.sof" file, and click Start.
5. Open Nios II Software Build Tools for Eclipse and select "../freq_reay_27" as the workspace.
6. If the project does not appear on the workspace, select file> Import > General > Existing Projects into Workspace and select "../freq_relay_27" as the root directory and click Finish.
7. Build the project and right-click the frequencyrelay folder, and select Run As > Nios II Hardware. If "Unable to validate connection settings" error shows up, go to Run > Run configurations and select the Target Connection tab. Click on Refresh Connections and Run.

System Controls:

- Use the Push button "Key1" to enter Maintenance mode.
- Use the switches SW0 to SW4 to turn loads on or off.
- Use the keyboard's number keys to enter the value for the thresholds or Backspace to delete a character. (Only in maintenance mode)
- Press "F" to set the entered value as the frequency threshold or "R" to set as the rate of change threshold.
- Only ten characters are allowed as the inputs (including the decimal).

Tasks:

FrequencyCalculator

FrequencyCalculator is responsible for calculating the frequency and the rate of change of frequency from the ADC values received from the Frequency ISR. The calculated values are then sent to StabilityAnalyser and VGADisplay task through queues.

This task is non-periodic and executes upon receiving a message from the FreqDataQ. This is to ensure that new frequency values are calculated as soon as possible and sent over to other tasks, which makes the system's reaction time better.

StabilityAnalyser

StabilityAnalyser is responsible for checking for any violations of the stability of the system. This task receives data from the FreqDataQ queue and then compares this value to a set threshold value. A global stableBoolean value is then changed depending on the outcome of this comparison. If stability conditions are violated, this task also sends a signal to the LoadManager task through a ShedQ queue for the first load shedding. It starts a timer by sending a signal to the TimerReset task using the TimerQ queue. The timer callback function uses the stableBoolean variable to determine the stability of the recently received values.

This task is non-periodic as it executes upon receiving data from the FreqDataQ queue. This ensures that the stability of the received values can be analysed immediately and appropriate signals are sent.

TimerReset

TimerReset is responsible for stopping, resetting, and starting the 500ms timer depending on the status of the system and the value of the global stability Boolean.

This task is non-periodic, only executing upon receiving data from the TimerQ queue. This is done so that the timer is controlled appropriately immediately on a change of stability.

TimerCB

TimerCB is a callback function for the 500ms timer. This task sends a signal to the LoadManager task through the ShedQ queue, telling the system to shed or not depending on if the system has been stable or unstable for 500ms.

This task is non-period executing whenever the 500ms timer reaches 500ms.

MaintenanceISR

MaintenanceISR is responsible for switching the system into Maintenance mode when the push button "Key1" is pressed. This is done by simply changing the global modeStatus variable. The system is only switched into Maintenance mode when the system is stable. If the system is not stable, a maintenanceFlag is raised so that the mode can be changed to maintenance mode whenever the system becomes stable next.

LoadManager

LoadManager is responsible for shedding or reconnecting loads depending on the values received from the ShedQ queue and the WallSwitchQ queue. The LED values are calculated in this task and sent over to the DisplayLed task through LEDQ. This task also handles changing the mode to maintenance mode if the MaintenanceISR did not previously change it due to the system being unstable.

This task also handles calculations for the first load shed timings. It writes them in global variables, which are read by the VGADisplay task.

This is a periodic task, running every 1ms. The loads can be shed/reconnected or turned on/off using the values received from both ShedQ and WallSwitchQ. Therefore, it would not have been viable to make this task execute on receiving a message from one of them or both of them, as it could lead the task to be suspended whenever the queues were empty, which could affect the reaction time of the system (LEDs not turned on or off quickly). Therefore, this task was made periodic. The period is set to 1 ms to ensure the loads are shed/reconnected as fast as possible by sending the LED values.

WallSwitch

WallSwitch is responsible for checking the status of the switches on the DE2-115 board and relaying this information to the LoadManager task.

This is a periodic task that runs every 50ms (as interrupts were not used for the switches), reading the switch values and sending this information over to the WallSwitchQ queue.

DisplayLed

DisplayLed is responsible for displaying currently load information on the DE2-115 board using the green and red LEDs. This is done by writing the appropriate LED status information to the green and red LEDs bases.

This task is non-period as it relies on receiving information from the LoadManager task through the LEDQ. Having this information dependent upon receiving data from a queue means that the load shed and reconnection information is as accurate and fast as possible to the real-time operations of the system.

KeyboardISR

KeyboardISR is responsible for reading the keyboard's inputs when in maintenance mode and sending this information to the KeyboardQ task. This ISR sends both the ASCII value or the key value to the ThresholdManager task through KeyboardQ.

ThresholdManager

ThresholdManager is responsible for updating the threshold values of the system through the keyboard input received through the KeyboardQ queue. Upon receiving data from KeyboardQ, ThresholdManager checks that the characters received are valid inputs filling up the buffer if they are. Upon receiving the character "F" or "R", this task sets the buffer as either the new frequency threshold or rate of change threshold, respectively.

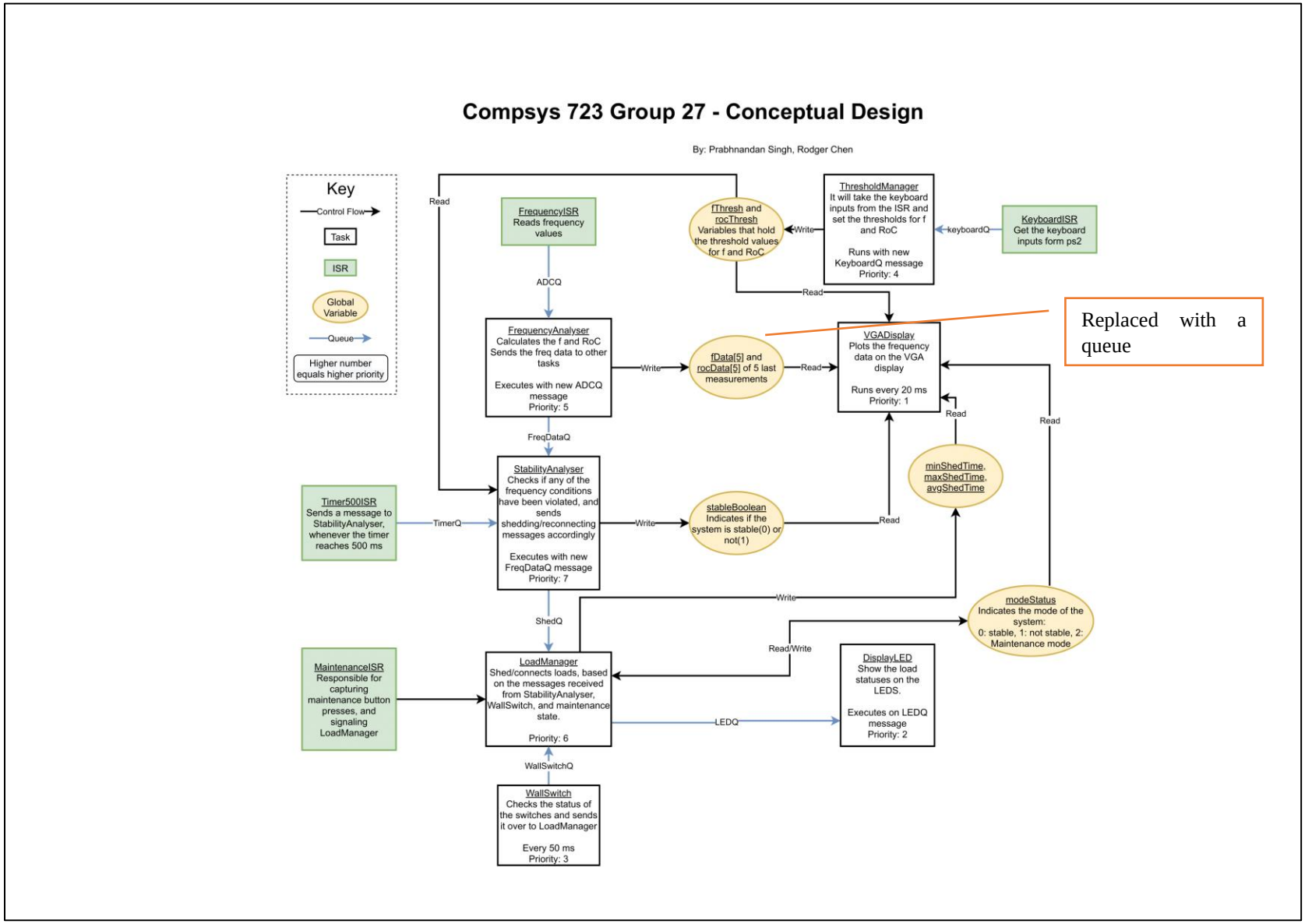
This task is non-periodic, executing upon receiving data through the KeyboardQ queue to ensure there is no lag between the key being pressed and processing the input.

VGADisplay

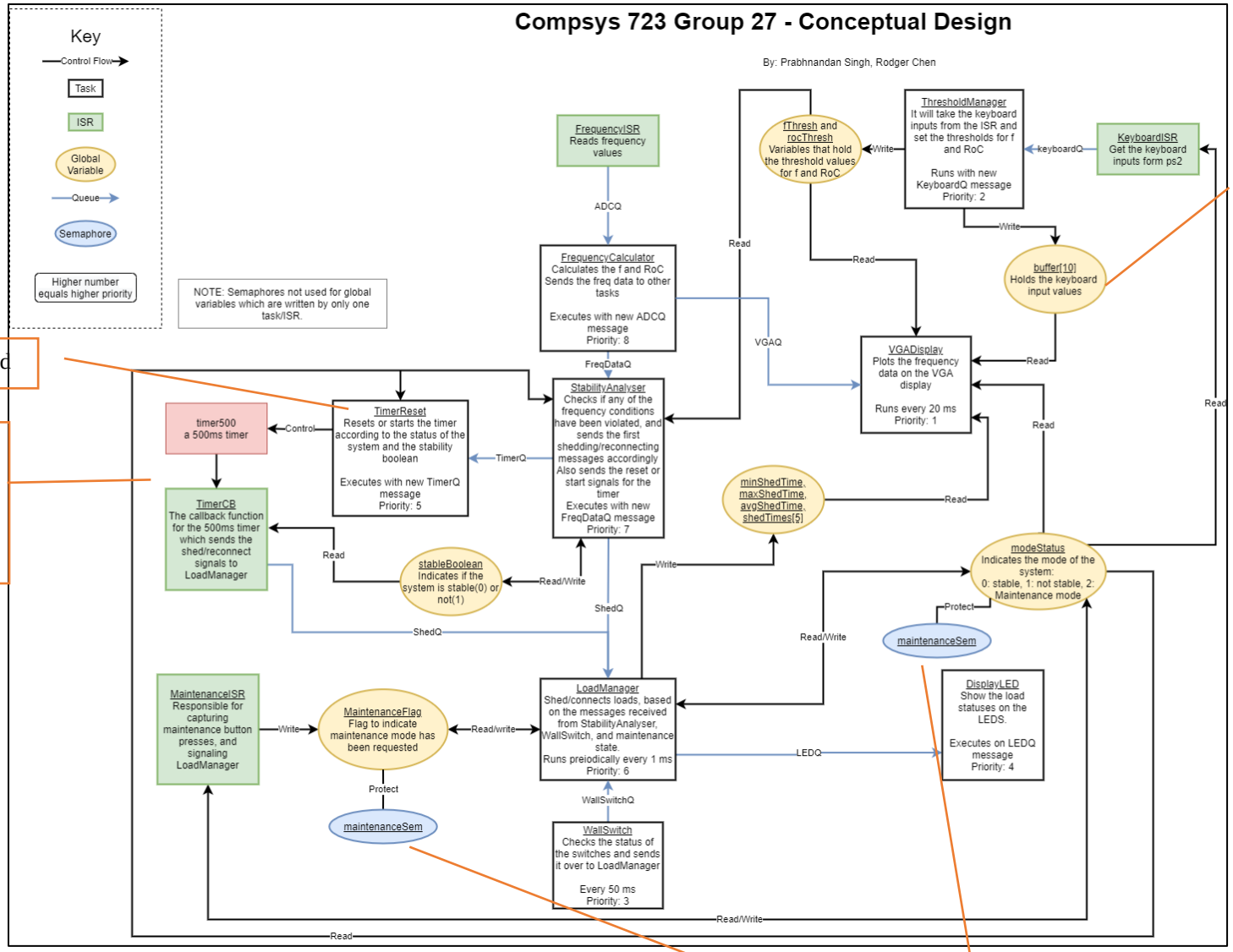
VGADisplay is responsible for displaying frequency relay information to a VGA display.

This task is periodic, updating the display every 20ms. This is possible because the system's function and display of system information can be run independently without affecting each other. This 20ms poll rate also approximately lines with the 60hz refresh rate of the VGA monitor.

Old Design:


Compsys 723 Group 27 - Conceptual Design
By: Prabhnandan Singh, Rodger Chen
Key
Control Flow
Task
ISR
Global Variable
Queue
Higher number equals higher priority
FrequencyISR
Reads frequency values
ADCQ
FrequencyAnalyser
Calculates the f and RoC
Sends the freq data to other tasks
Executes with new ADCQ message
Priority: 5
FreqDataQ
fThresh and rocThresh
Variables that hold the threshold values for f and RoC
ThresholdManager
It will take the keyboard inputs from the ISR and set the thresholds for f and RoC
Runs with new KeyboardQ message
Priority: 4
KeyboardISR
Get the keyboard inputs form ps2
keyboardQ
fData[5] and rocData[5] of 5 last measurements
VGADisplay
Plots the frequency data on the VGA display
Runs every 20 ms
Priority: 1
Replaced with a queue
Timer500ISR
Sends a message to StabilityAnalyser, whenever the timer reaches 500 ms
TimerQ
StabilityAnalyser
Checks if any of the frequency conditions have been violated, and sends shedding/reconnecting messages accordingly
Executes with new FreqDataQ message
Priority: 7
stableBoolean
Indicates if the system is stable(0) or not(1)
minShedTime, maxShedTime, avgShedTime
modeStatus
Indicates the mode of the system:
0: stable, 1: not stable, 2: Maintenance mode
ShedQ
MaintenanceISR
Responsible for capturing maintenance button presses, and signaling LoadManager
LoadManager
Shed/connects loads, based on the messages received from StabilityAnalyser, WallSwitch, and maintenance state.
Priority: 6
LEDQ
DisplayLED
Show the load statuses on the LEDS.
Executes on LEDQ message
Priority: 2
WallSwitchQ
WallSwitch
Checks the status of the switches and sends it over to LoadManager
Every 50 ms
Priority: 3
Read
Write
Read/Write

New Design:



New task added

Explicitly show the timer functionality

New global variable added, to show the keyboard input on

Added additional control flow lines according to the implementation

Show the semaphores required

Design decisions

Changes made to conceptual design

Our initial conceptual design mainly covered the tasks/ISRs which we thought were needed for the implementation. It also covered how the control flows between different tasks and what variables were written/read by different tasks/ISRs. However, when we started the implementation, we realized some changes were necessary, highlighted in the designs provided earlier. One extra task was added to the design, the TimerReset task, which controls the 500ms timer according to the state of the system and the incoming stability signals from the StabilityAnalyser task. Other changes include showing the timers and the callback function more explicitly (as suggested by the feedback received) and adding more control flow lines, as while making the initial conceptual design, we were not sure which tasks would read/write global variables. Some more changes were made as required to achieve the desired functionality.

Split of functionality

The functionality of the system was split using different tasks/ISRs, each having its responsibilities. One responsibility criterion was the interaction of the software with the hardware. Different task and ISRs were responsible for interacting with different hardware and processing the data as required. For example, FrequencyISR interacts with the frequency analyser, WallSwitch interacts with the switches, KeyboardISR and ThresholdManager interact with the PS2 keyboard, DisplayLED interact with the LEDs, TimerReset interacts with the timer, and the VGADisplay interacts with the VGA port. Based on the data from these tasks/ISRs, calculations and decisions had to be made. The split for this functionality was based on what calculation or decision making was needed and which other tasks needed the outputs. The FrequencyCalculator task does the frequency calculations required for both displaying on VGA and analysing the stability. The StabilityAnalyser checks for any violations of thresholds required for resetting the timer or managing the loads. Finally, LoadManager has the responsibility to manage the loads. If one task were responsible for all of the calculations and decision making, it would have affected the system's performance.

Shared variables protection mechanism

The design was made such that most of the globally shared variable was only written by one task/ISR but could be read by multiple tasks/ISRs, therefore not requiring any protection from overwriting data. However, MaintenanceFlag and modeStatus could be written by both LoadManager and MaintenanceISR, and we used a semaphore to prevent data corruption.

Communication mechanism

We mainly used queues as the communication mechanisms, as it allows sending multiple messages to a task. We used a standard queue size of 30 which means it is improbable that the task/ISR writing to a queue will block due to the queue being full. This also means that all the tasks can work at their speeds, but the data will still be synchronized (received in the sequence sent (FIFO)). Additionally, the tasks automatically block when sending to a full queue or receiving from an empty queue; therefore, tasks do not use the processor when not needed.

Task Priority

The priorities assigned to the tasks are:

1. VGADisplay
2. ThresholdManager
3. WallSwitch
4. DisplayLed
5. TimerReset
6. LoadManager
7. StabilityAnalyser
8. FrequencyCalculator

In this list, the higher value of priority, the more time-critical the task is. For example, FrequencyCalulator would have the highest priority while VGADisplay would be the lowest priority.

VGADisplay was chosen to be the lowest priority task because the only purpose of this task was to display data through a VGA monitor to the user and does not have any system-critical functions. Furthermore, it executes every 20ms, therefore we did not want this task to keep the processor busy while real-time critical tasks are blocked.

ThresholdManager is second on the list because updating threshold values on the system is not time-critical as well. It only runs when keyboard input is received in maintenance mode and has very fast execution.

WallSwitch is third on the list because the task only runs every 50ms and is only responsible for sending the switch values to the LoadManager task. However, getting the switch values is not very time-critical, and slight delays due to being blocked by the scheduler are acceptable given the reaction time for a human is only around 250ms, and the task runs at a much higher speed of 50ms.

DisplayLed is fourth on the list because the execution of this task depends only on the higher priority task. It is more time-critical than the lower priority tasks but still less time-critical than the other higher priority tasks.

TimerReset is fifth on the list because it is responsible for controlling the 500ms timer, which makes it quite more critical than the lower priority tasks, however to control the timers, the appropriate values need to be calculated and analysed faster than this task, making it less time-critical than the higher priority tasks.

LoadManager is sixth on the list to make sure that the system's current status can be displayed as fast as possible, making it more time-critical than the lower priority tasks. However, it needs some information from the higher priority tasks too.

StabilityAnalyser is seventh on the list as this task determines if the recently received frequency values would affect the system's stability, making it very time-critical. It only depends on FrequencyCalculator to receive the recent frequency values.

FrequencyCalculator was chosen as the highest priority task as the other tasks depend on the values calculated by this task. The faster the values are calculated, the faster the rest of the system will react to the changes in the stability of the frequency values.

FreeRTOS features

The list of FreeRTOS features we used in our implementation is given below:

- Task: Tasks were used to allow multitasking. Tasks were created with the xTaskCreate call. The scheduler is started with the vTaskStartScheduler function call. xTaskGetTickCount command is used to get the number of ticks a task has been running for. This is used to calculate the first load shed times and the system's runtime.
- Timer: A 500 ms timer is used to analyse the system's stability and send shedding/reconnecting signals appropriately. xTimerCreate call is used to create the timer. xTimerIsTimerActive call is used to check if the timer is running or not. xTimerReset, xTimerReset and xTimerStart calls are used to control the 500 ms timer.
- Counting semaphores: A semaphore is used to avoid data corruption when a global variable can be written by multiple tasks/ISRs. xSemaphoreCreateCounting is used to initialize the semaphore. xSemaphoreTake and xSemaphoreGive calls are used to get and release semaphores.
- Queues: Queues were used as a communication mechanism between the tasks. The queues were created using the xQueueCreate call. Other functions were used to send or receive messages and check if messages were already waiting in a queue.

Tasks interaction

FrequencyCalculator calculates the frequency data from the ADC values received from the FrequencyISR. These values are then sent through queues to StabilityAnalyser and VGADisplay. StabilityAnalyser compares the frequency data with the thresholds (default or set by ThresholdManager) and sends the appropriate shedding signal to

LoadManager and the timer signals to the TimerReset. TimerReset depending on the stability boolean received from StabilityAnalyser starts, stops or resets the 500ms timer. The LoadManager depending on the shed signal received from the timer callback function or the first load shed from the StabilityAnalyser, along with the switch values from the WallSwitchQ, determines if loads need to be turned on/off or shed/reconnected by sending LED values to DisplayLed. DisplayLed displays the current loads status on the LEDs on the FPGA board. The ThresholdManager sets the threshold values from the keyboard inputs received. VGADisplay displays all the required frequency relay information from the frequency values from FrequencyCalculator and the global variables set by other tasks.

Potential problems prevented

One of the main potential problems with multitasking is the possibility of data being corrupted due to a lack of mutual exclusion. To prevent this problem, we have only let a single task or ISR write to a global variable, with one exception where we used a semaphore to ensure mutual exclusion.

Limitations in the design

The number of data points that are plotted on the graph can be set by just changing the DATA_SIZE macro. However, other values related to the graph are calculated using this value, limiting the data size that can be plotted on the graph. With the current design, only factors of 25 can be used as the value for DATA_SIZE. Another limitation of the design is that value for the data size can only be changed before compiling the code, not during runtime.

Contribution table

Task	Contributed By	Approximate time taken (hours)
Conceptual Design	Prabh, Rodger	3
Final Conceptual Design	Prabh, Rodger	3
ISRs	Prabh, Rodger	1-2
FrequencyCalculator	Rodger	< 1
StabilityAnalyser	Prabh, Rodger	< 1
TimerReset	Prabh, Rodger	1
LoadManager	Prabh	2-3
WallSwitch	Rodger	< 1
DisplayLed	Rodger	< 1
ThresholdManager	Prabh, Rodger	1-2
VGADisplay	Prabh	2-3
Documentation Report	Prabh, Rodger	5-6
Debugging and code comments	Prabh, Rodger	2-3