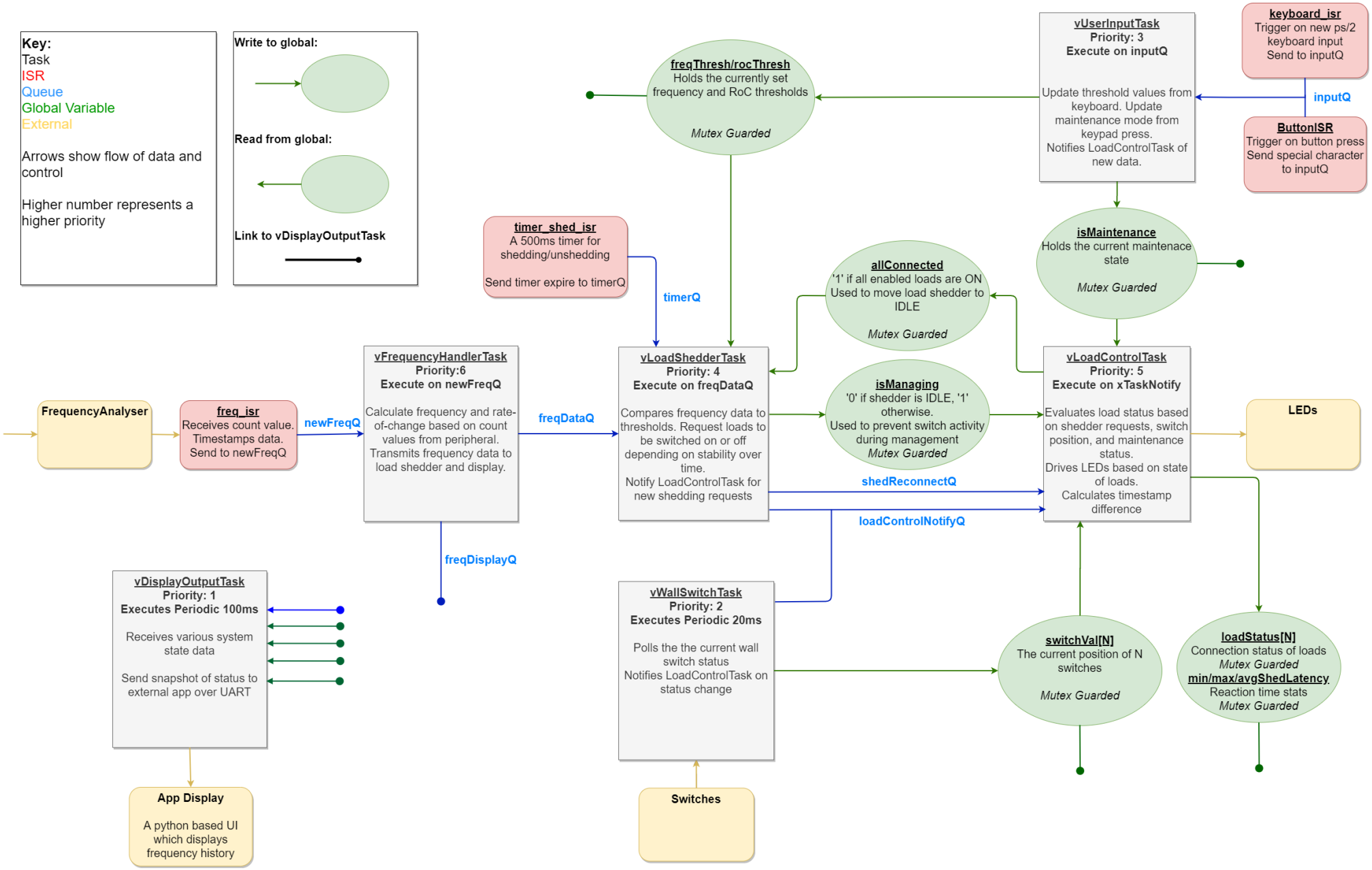
Key:
Task
ISR
Queue
Global Variable
External
Arrows show flow of data and control
Higher number represents a higher priority
Write to global:
Read from global:
Link to vDisplayOutputTask
freqThresh/rocThresh
Holds the currently set frequency and RoC thresholds
Mutex Guarded
vUserInputTask
Priority: 3
Execute on inputQ
Update threshold values from keyboard. Update maintenance mode from keypad press.
Notifies LoadControlTask of new data.
keyboard_isr
Trigger on new ps/2 keyboard input
Send to inputQ
inputQ
ButtonISR
Trigger on button press
Send special character to inputQ
timer_shed_isr
A 500ms timer for shedding/unshedding
Send timer expire to timerQ
timerQ
isMaintenance
Holds the current maintenace state
Mutex Guarded
allConnected
'1' if all enabled loads are ON
Used to move load shedder to IDLE
Mutex Guarded
FrequencyAnalyser
freq_isr
Receives count value.
Timestamps data.
Send to newFreqQ
newFreqQ
vFrequencyHandlerTask
Priority:6
Execute on newFreqQ
Calculate frequency and rate-of-change based on count values from peripheral.
Transmits frequency data to load shedder and display.
freqDataQ
vLoadShedderTask
Priority: 4
Execute on freqDataQ
Compares frequency data to thresholds. Request loads to be switched on or off depending on stability over time.
Notify LoadControlTask for new shedding requests
isManaging
'0' if shedder is IDLE, '1' otherwise.
Used to prevent switch activity during management
Mutex Guarded
vLoadControlTask
Priority: 5
Execute on xTaskNotify
Evaluates load status based on shedder requests, switch position, and maintenance status.
Drives LEDs based on state of loads.
Calculates timestamp difference
LEDs
shedReconnectQ
loadControlNotifyQ
freqDisplayQ
vDisplayOutputTask
Priority: 1
Executes Periodic 100ms
Receives various system state data
Send snapshot of status to external app over UART
App Display
A python based UI which displays frequency history
vWallSwitchTask
Priority: 2
Executes Periodic 20ms
Polls the the current wall switch status
Notifies LoadControlTask on status change
Switches
switchVal[N]
The current position of N switches
Mutex Guarded
loadStatus[N]
Connection status of loads
Mutex Guarded
min/max/avgShedLatency
Reaction time stats
Mutex Guarded

Table 1 – Description of Tasks and ISRs

vFrequencyHandlerTask This task is responsible for interpreting the count from the external frequency analyser as frequencies (Hz) and rates-of-change. The task is blocked until a new reading is transmitted over newFreqQ, and then transmits the frequency data to both the load shedder and display output. Because this is the first step in the 'hot path', it is afforded the highest priority.	vUserInputTask Processes input values from the keyboard and the keypad, received from the input queue from respective ISRs. Resolves keyboard input into float thresholds for rate-of-change and frequency, and assigns these for the load shedder. Resolves keypad 3 as maintenance mode toggling. Maintains input buffer with keyboard state shared with vDisplayOutputTask.
vWallSwitchTask Periodically checks wall switches for updates. Bit-encoded switch values are decoded to an array for easier processing. If the wall switch state has changed, sends a mailbox alert to awaken the loadControl task to read the new switch state.	vDisplayOutputTask Responsible for communicating the entire system state to the app display over JTAG UART. Periodically sends update packets. Also responsible for driving the LCD display with the current keyboard input buffer shared from vUserInputTask.
vLoadControlTask Combines the output of the load shedder and the switches to drive the final load state. Switches are read directly, managed loads are calculated based on received shed/unshed messages. Controls both the RED load LEDs and the GREEN shedder LEDs. Calculates shed latency from timestamp difference since the first unstable reading.	vLoadShedderTask An FSM describing when loads are automatically managed by the system. IDLE until a threshold violation is detected, will then move to SHED state where a shed request is sent every 500ms. When again stable, moves to RECONNECT state where an unshed request is sent every 500ms. When vLoadControlTask reports all loads reconnected, returns to IDLE.
freq_isr External frequency analyser peripheral invokes freq_isr when a new reading becomes available. The reading is a counter value representing a signal frequency. This reading is timestamped for latency calculation, and forwarded to vFrequencyHandlerTask by queue.	keyboard_isr External keyboard peripheral invokes keyboard_isr when any activity is detected. One keystroke sends three bytes (and three ISRs), so a flag is used to make sure the data is only processed once. Keyboard code is converted to ASCII and sent by queue to vUserInputTask.
button_isr Invoked by external keypad peripheral when a button is pressed. Sends button press to vUserInputTask using the same input queue as the keyboard, using a special sentinel value. Only keypad 3 is masked to invoke this ISR.	timer_shed_isr FreeRTOS timer callback. When the load shedder is managing (shedding or reconnecting), this timer triggers at 500ms intervals to determine when a load should be acted upon. Resets when enters SHED or RECONNECT FSM state. Sends '1' to timerQ on each overflow.

Table 2 – Description of Task Communication Mechanisms

inputQ Queue communication of ASCII keypress values from keyboard_isr to vUserInputTask. Also contains keypad press represented as sentinel value.	timerQ Queue of '1's from timer_isr to vLoadShedderTasks indicating when a load should be shed or reconnected during management. Runs continuously during management.	loadControlNotifyQ A notification queue used for awakening vLoadControlTask. The value indicates the notification source, which is either a wall switch update or an automatic load shedder update.
newFreqQ Queue communication of struct containing count and timestamp from freqISR to vFrequencyHandlerTask	freqDataQ Queue communicating struct containing frequency, rate-of-change, and timestamp to the load shedder.	shedReconnectQ Queue communication of struct containing request from vLoadShedderTask to vLoadControlTask to shed a load.
<…>ToDisplayQ Most tasks communicate data to vDisplayOutputTask for display on the app. They either send the data through a queue directly such as freqDisplayQ, or use a queue as a notification for vDisplayOutputTask to read a shared variable. Display notify queues are omitted on the above diagram.	freqThresh/rocThresh Mutex-guarded float variables which describe the threshold for frequency or rate-of-change before the system is considered 'unstable'. Written to by vUserInputTask based on keyboard input, and read by vLoadShedderTask.	isMaintenance Mutex-guarded boolean variable describing whether the system is in maintenance mode. When '1' no automatic load management can occur. Can only be toggled by button if the LoadShedder is in 'IDLE' state, otherwise ignored. Written by vUserInput and read by vLoadShedderTask.
isManaging Boolean signal from vLoadShedderTask to vLoadControlTask that the automatic load management is running. vLoadControlTask uses this to prevent manual control during management.	allConnected Boolean signal from vLoadControlTask to vLoadShedderTask to indicate that all enabled loads are connected. vLoadShedderTask uses this to return to IDLE from RECONNECT.	switchVal The current state of the DE2 switches, represented as an array. '1' means the switch is in the ON/UP position, '0' in the OFF. Assigned by vUserInputTask, read by vLoadControlTask to manually control loads.
min/max/avgShedLatency Measurements of the latency between when a threshold is breached and the response. Calculated by vLoadControlTask, and sent to the app by vDisplayOutputTask.	loadStatus The final output showing whether each load is on or off, represented as a Boolean array. Created by vLoadControlTask and sent to the app by vDisplayOutputTask.	

Simulator:

The same FreeRTOS program can be run on either the NIOS II board or on a Windows-based machine. This was achieved by extending the provided FreeRTOS simulator to include mocked peripherals and Altera NIOS library calls. This greatly enhanced the remote development where only one project member had a DE2-115 board for the NIOS target. Any contributor can seamlessly add and test functionality from the same codebase. Output is reflected in the app. Usage instructions in the readme.

App:

To monitor the behaviour of the relay, we implemented a python app. The python app reads protocol messages from STDIN, to which the STDOUT of the JTAG UART is bash piped (or normal STDOUT in the case of the simulator). For usage details see the README. The app itself is implemented using PySimpleGUI and Matplotlib, and displays the following:

- Graphs of frequency and rate-of-change over a time window
- Threshold values
- Minimum, maximum, average, and recent latencies
- Switch status
- Load status

Design Decisions:

We aimed for a highly modular design of a larger number of smaller interacting components. Tasks were split between individual files. The motivation for this was to enhance remote collaboration by preventing merge conflicts and technical dependencies. This low coupling also made it easier for us to mock out Altera functions for the simulator. Shared variables were only visible to the files that required them, providing some degree of encapsulation. Overall, we found this approach largely beneficial for remote development. One drawback was that not all modules were suitable for being separated. We developed automatic load management and manual load management as two separate tasks, however realised that their behaviour was highly dependent on each other. To resolve this, they each have some feedback through shared variables. However, they may be more appropriate as one task.

We use queues for periodic messaging of small data (e.g. values). Many periodic messages must be guaranteed to be read, such as shed requests from the load shedder. Queues provide this sequence for free, as well as allowing tasks to ‘awaken’ on new data only allowing for better scheduling. In some cases, we awaken a task at a specific time using a notification queue as a synchronisation mechanism. We use this when there are multiple triggers for a task, such as when xLoadControlTask needs to be updated on either a switch change or an automatic shed request.

Shared memory is used for data which only the current state matters, as well as for shared arrays. For example, the current threshold values only need to be checked against new incoming frequency readings; the task does not need to be reactive to an update of the threshold itself. Transmitting values from a C array is also less intuitive than sharing the array. All shared variables are mutexed for safe access between tasks.