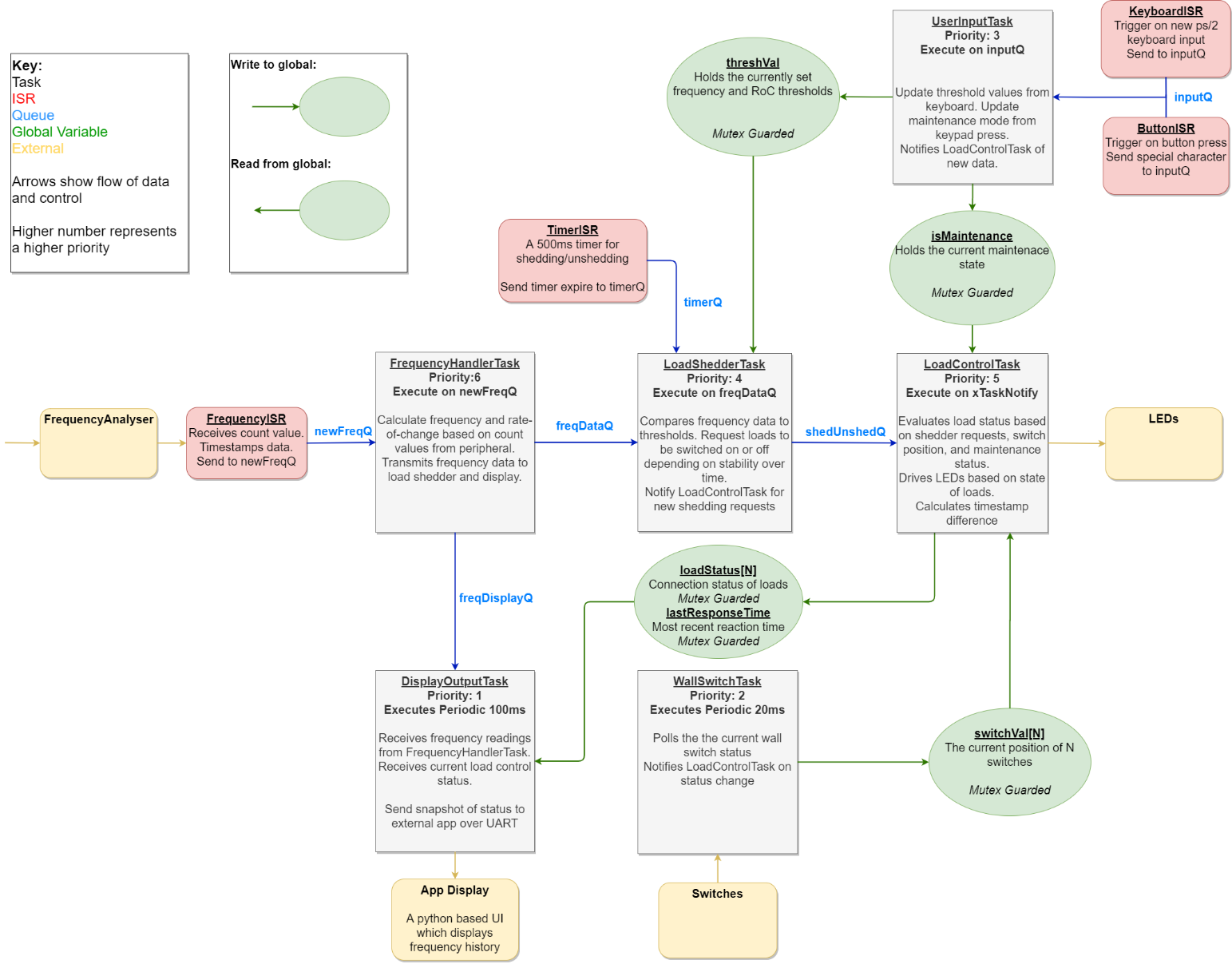
Page 1: Diagram PLACEHOLDER: THIS IS THE OLD DIAGRAM

Updates that need to be made to our paper design:

* Tasks Reflect Actual Names
* Additional shared vars and queues: isManaging, isIdle, ???



|  |  |
| --- | --- |
| 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**  A callback from a FreeRTOS timer. When the load shedder is managing (i.e. shedding or reconnecting), this timer is enabled at 500ms intervals which determines when the load shedder should act on a load. Reset when entering SHED or RECONNECT FSM state. Sends ‘1’ to timerQ on each overflow. |

**Tasks and ISRs:**

**Task Communication:**

|  |  |  |
| --- | --- | --- |
| newFreqQ | freqDisplayQ/freqDataQ | shedReconnectQ |
| <…>ToDisplayQ | **freqThresh/rocThresh** | **isMaintenance** |
| isManaging | **allConnected** | **switchVal** |
| userInputBuffer/newUserInputValue | **min/max/avgShedLatency** | **loadStatus** |

**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 off 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 Reflections:**

* Load shedder and load control highly coupled, would have been wise to combine
* Highly modular file structure allowed for better collaboration without conflicts
* Queues used for messages which were periodic, shared variables used for variables representing internal ‘state’