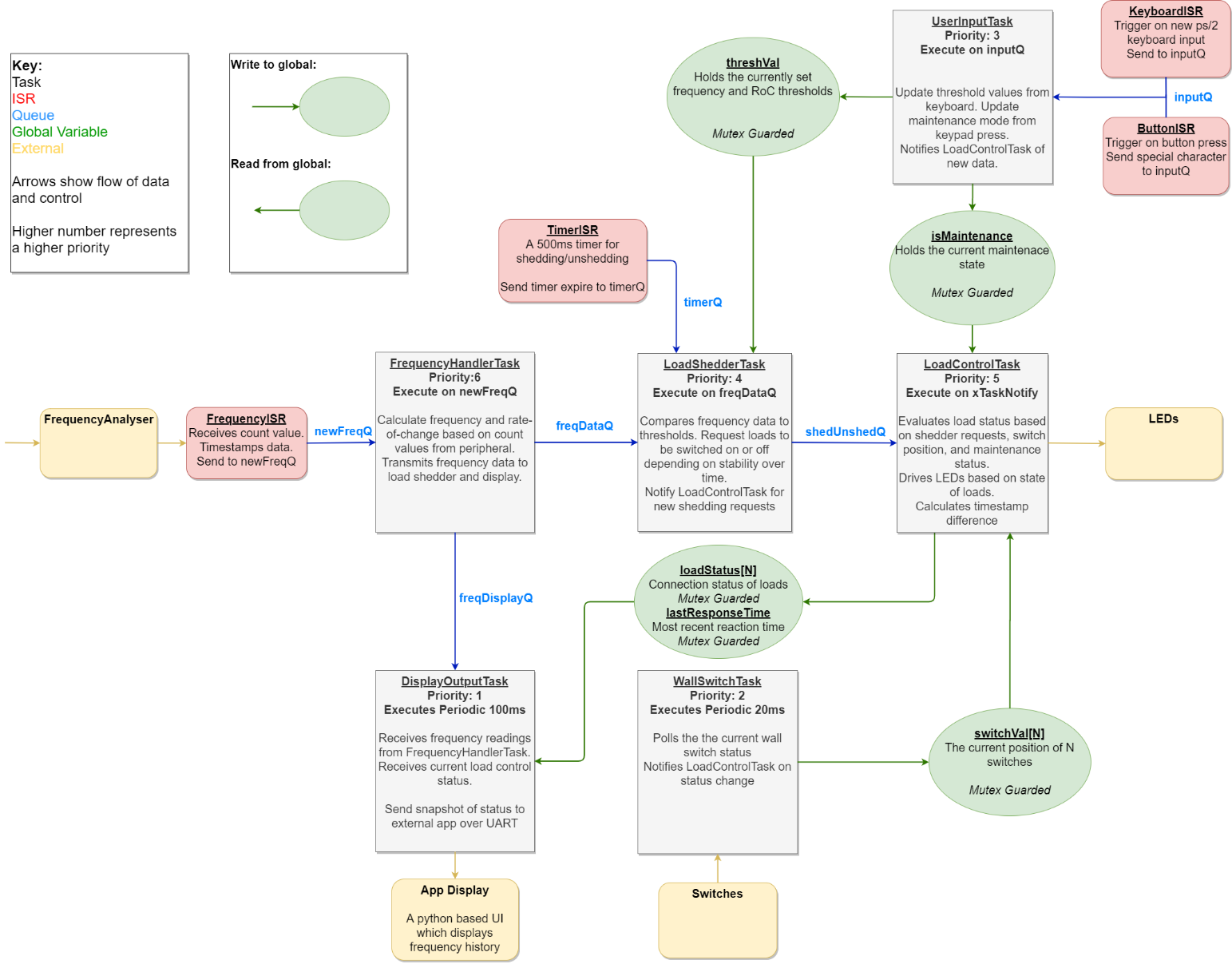
Page 1: Diagram PLACEHOLDER: THIS IS THE OLD DIAGRAM

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** |
| **vWallSwitchTask** | **vDisplayOutputTask** |
| **vLoadControlTask** | **vLoadShedderTask** |
| **freq\_isr** | **keyboard\_isr** |
| **button\_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