Task	Description	Priority
auditFrequencyTask	Reads the frequency coming in, calculates stability, and gives commands to *updateConnections*. This task was created to concurrently read input data and manage tasks through *updateConnections*. Highest priority as it is responsible for calculating system stability and keep up with new input values.	4
updateLoadConnections Task	Reads commands from *auditFrequencyTask* and sheds or connect loads accordingly. Highest priority as we want this task to manage loads as fast as possible.	4
updateSystemStateTask	Manages the other tasks by reading the current state. We chose to have this task as we needed a reliable way of turning off tasks depending on state. For example, we did not want *auditFreq* or *updateLoad* to run while in the maintenance state. It is also responsible for resetting any queues.	3
updateThresholdTask	Responsible for reading keyboard data and updating the stability thresholds. *updateSystem* makes sure that only this task is on during the maintenance state.	2
VGAOutputTask	Outputs the display, showing frequency, rate of change, current status, etc. This task is the lowest priority as it does not need to redraw the screen faster than 50Hz (20ms), meaning a less frequent deadline than any other tasks.	1

ISR	Description
mt_bt_isr	Responsible for detecting when the maintenance button is pressed.
freq_relay	Reads incoming data on peak signal interrupt and pushes it into a queue.
ps2_isr	Responsible for reading keyboard inputs during maintenance state.

Mutex

Semaphore mutexes were used to protect groups of variables. For example, *freqCalc* protected all the variables to do with frequency calculations. By grouping variables, we lowered the complexity of the program and guarantee mutual exclusion.

Queue

Queues were used to safely transfer a stream of data between tasks. For example, frequency values from *freq_relay* were sent to *auditFrequency* for processing.

Mailbox

A mailbox was implemented using a queue with a size of one. For example, *auditFreq* would send a message whenever it detected a change in stability. This provided us with an easy way to update the “Stability” part of the UI safely.

FSM

The FSM functionality works with a 2D state transition table. By having one axis represent states and the other events, we can create a simple Moore FSM that easily maintainable.

Block Diagram (Full System Description)