COMPSYS 723 Assignment 1 – Frequency Relay

1. User Interface
2. **DE2 Board**

* Toggle switches 4 to 0: the wall switches for loads 4 to 0 respectively.
* Red LEDs 4 to 0: the current status of connectivity for loads 4 to 0 respectively.
* Green LEDs 4 to 0: the current shed status for loads 4 to 0 respectively.
* Button 3: toggles *Maintenance Mode*.
* Red LED 17: whether *Maintenance Mode* is on or off.
* Red LED 16: whether *Precise Incremen*t is on or off.
* Button 2: toggles *Precise Increment* - for setting thresholds.

1. **Keyboard**

* UP/DOWN: Increment/decrement *frequency threshold* by 0.5 or 0.1 Hz.
* LEFT/RIGHT: Increment/decrement *rate of change* (RoC) *threshold* by 1 or 0.2 Hz/s.

Amount dependents on *Precise Increment* flag.

1. **VGA Display**

* Graph of Frequency and RoC - updated right to left
* Total run time of the system.
* Frequency and RoC thresholds.
* Reaction times - updated right to left (round table).
* Maximum, minimum, average reaction times.

1. ISRs
2. **Frequency Relay**

Triggered when a new peak data has been received, calculate instantaneous frequency and push to *frequency data* queue. It also records the current tick stamp and pushes that to *time stamp* queue.

1. **Button ISR**

Reads the edge capture register, toggle *maintenance mode* or *precise increment* flag then clear edge capture register.

1. **Keyboard ISR**

Retrieve PS2 keyboard data, decode and push relevant data (ASCII\_MAKE\_CODE) to *keyboard data* queue.

1. **Timer ISR**

Set *timer expired* flag, indicating that 500 ms has expired and sends a notification to the *Load Manager* to execute.

1. Tasks
2. **Frequency Update Task**

Priority: 5 Execution condition: when *frequency data* queue is not empty.

Retrieve the head of the *frequency data* queue and *time stamp* queue. Take frequency mutex and calculate RoC and store them into the *frequency* and *time stamp* arrays at *freq index,* which gets incremented. Upon completion sends a notification to load manager to execute.

1. **Keyboard Update Task**

Priority: 4 Execution condition: when *keyboard data* queue is not empty.

Get key press and increment/decrement instantaneous frequency and rate of change (RoC) thresholds, ensures that the values are positive.

Pops the *keyboard data* queue, checks if it is one of the arrow keys and it is press down instead of release by toggling a *key pressed* flag every time an arrow key has been popped. If it is a press down it will take the *threshold mutex semaphore* protecting the thresholds, increment or decrement by a certain amount then give the semaphore back. The amount that is incremented or decremented depends on *precise increment* flag, toggled by button ISR.

A minor problem with this approach is that if two or more arrows keys are pressed, the thresholds change may update twice or not at all. This can be simply fixed by having individual key pressed flag for each arrow key.

This task has higher priority than *Load Manager* Task as the manager requires the threshold values to determine whether the system is stable.

1. **Load Manager Task**

Priority: 3 Execution condition: Task Notification from *Frequency Update* Task or *Timer ISR*

Core functionality of this Frequency Relay, controlled by a FSM (Fig. 2). When a new frequency data has been received, it checks the current state, apply state output and decide on the next state.

In *maintenance* *mode,* the *Load Manager* task is placed on IDLE and a semaphore is given to *Load Controller* to notify that system is stable.

If unstable condition is met, it will push a request to shed to *load requests* queue and go to UNSTABLE state, where it starts a 500 ms timer, if the timer expires, it will send another shed request and restart timer. If the system becomes stable before timer expires, it will go to STABLE state and restart the timer.

If the timer expires in the STABLE state then it will send a reconnect request and restart the timer, or if the system becomes unstable, it will go back to the UNSTABLE state. This repeats until it is stable for long enough that all the disconnected loads have been reconnected. The load manager task will then resume its idle state.

1. **Load Control Task**

Priority: 2 Execution condition: periodic, 5 ms.

Read toggle switches, if *maintenance mode* is on, set load LEDs according to toggle switches and reaction time history will be reset. Only the toggle switches 4 to 0 are used. If the Load Manager is in IDLE state, i.e. stable, it clears the *managed LEDs* (green) and set load LEDs according to toggle switches.

When the Load Manager is active (not IDLE), Load Controller will attempt to pop the *load request* queue and disconnect the lowest priority load or reconnect highest priority load. If it’s a Disconnect request, it will also get the current system tick count, calculate the reaction time and update min/max/avg reaction times.

Regardless of the Load Manager’s state, if any toggle switches have been turned off, the respective load will be immediately turned off, but will not reset the 500 ms timer that Load Manager uses for stable/unstable state transitions.

Then it will check whether the toggle switch values against the current loads using xor, and all connected flag will be set/cleared.

1. **VGA Task**

Priority: 1 Execution condition: periodic, 33 ms (~30 Hz).

Reads the frequency and RoC arrays and plots them onto graphs with red lines indicating thresholds. Also prints the total run time, current frequency and RoC thresholds, along with 5 most recent reaction times and min/max/avg reaction times.

This task has the lowest priority, and the reading of frequency data arrays are not mutex protected, as it was decided that it would not affect functionality as frames would be almost instantaneously corrected from the point of the user if the read every produced an error.

1. Shared Variables

A total of sixteen global variables are included in this design and are considered as four different types;

* User-interface – data obtained through user interface controls
* Internal system control – flags indicating current system conditions
* Frequency related data – frequency and RoC
* Reaction time information – information about reaction time

Mutual exclusion protection is required for shared variables that should not be altered while a task is using that data in its operations. Mutex semaphore was used to properly protect some shared variables over other solutions such as binary semaphores.

User interface and internal system control generally do not need mutual exclusion. Any data that is altered within these shared variables can simply be considered the next time the relevant task is executed without affecting functionality. The only exception is the shared variables which contain thresholds in which a read and write at the exact same time on a floating point container may be dangerous. For this reason the containers holding the threshold data is semaphore protected.

Shared variables containing frequency related data require semaphore protection. This data is critical in determining system functionality. Semaphore protection is used to ensure that the load manager task has time to process the frequency data before it is updated.

Shared variables containing reaction time information do not require semaphore protection. The main functionality of the system is not dependent on this and alterations on the data itself while another task is using it is acceptable.

1. Other Design Decisions
2. **Load Manager FSM**

Originally the *Load Manager* was designed as a Moore machine consisting of five states. However this was changed to a Mealy machine with three states, so that requests can be sent to the *Load Control* task immediately to reduce reaction time.

1. **Execution conditions**

The *Load Manager* task is triggered by FreeRTOS’ task notification features, notified by the *timer ISR* and the *Frequency Update* task so that it is aware that there is new data that should be processed. The task notification feature is much more efficient and faster than other solutions such as binary semaphore in this scenario.

1. **Communication**

Queues were used to communicate data between some ISR-to-task and task-to-task. They are thread-safe and ensure that data packets are not missed. This reassurance cannot be achieved using global variables.

1. **Periodicity**

Periodicity of tasks are achieved using FreeRTOS’ Task Delay Until to ensure correct delays for periodic tasks. Non-periodic tasks are triggered by notification or non-empty queues.

1. **Priorities**

Higher priorities were assigned to tasks in charge of retrieving data. The *Load Manager* and *Load Controller* task were given lower priorities because it is most important that the system has the most up to date information before trying to process it. *VGA* task was given the lowest task priority because it can be considered a background task, even if it fails to meet deadlines or print out a wrong value for one frame, the system still retains complete functionality.

1. **Shared Variable Protection**

See Section IV.

1. **Task Functionality Partitioning**

Functionality is partitioned to divide the application into input, control and the output. Data gathering tasks are divided into keyboard data retrieval and frequency data retrieval as they require different execution conditions. The majority of control of the system is done through the *Load Manager* task. The output is further divided into a *VGA* task and a *Load Control* task with each requiring different execution times with different priorities.

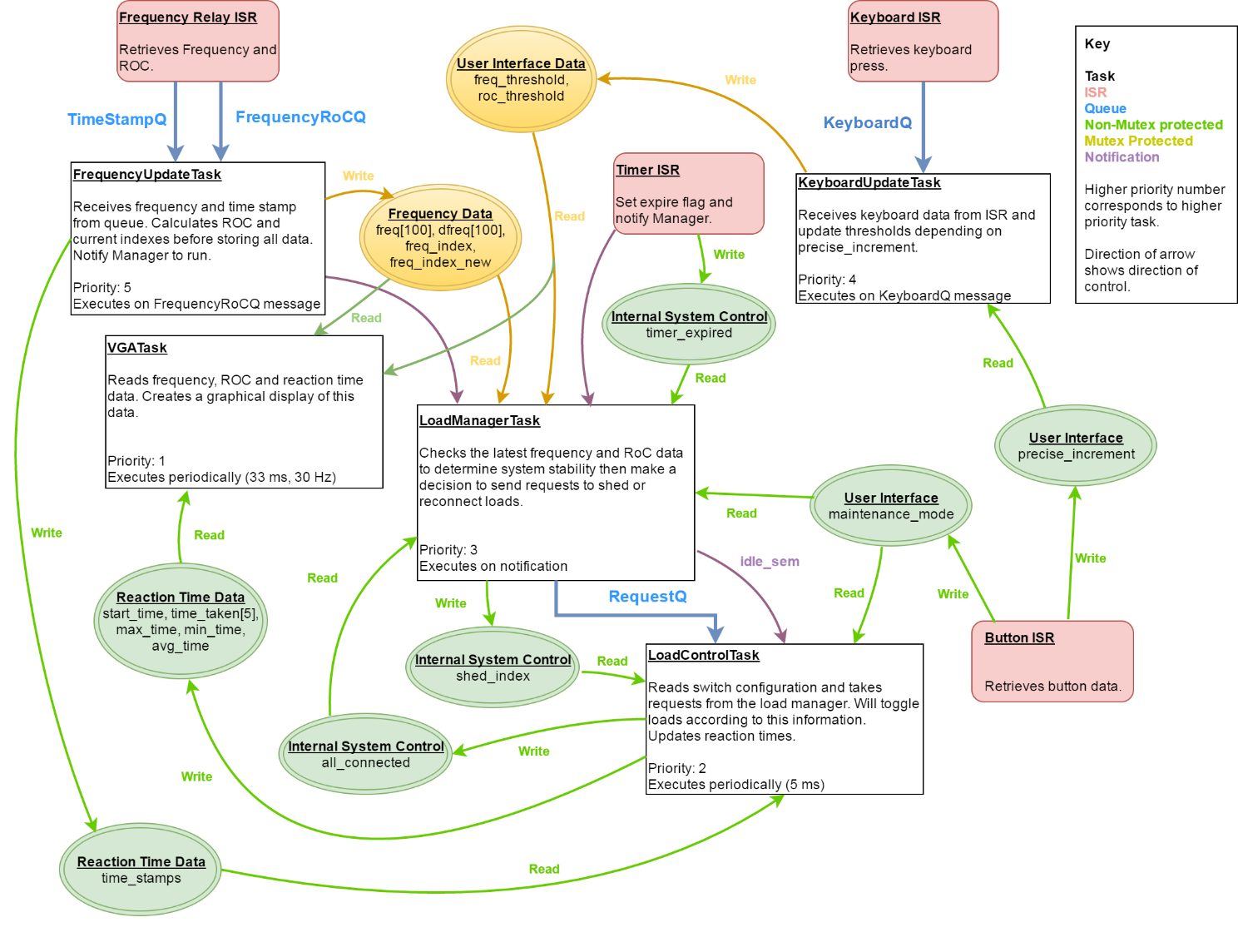
1. Limitations/Future Development

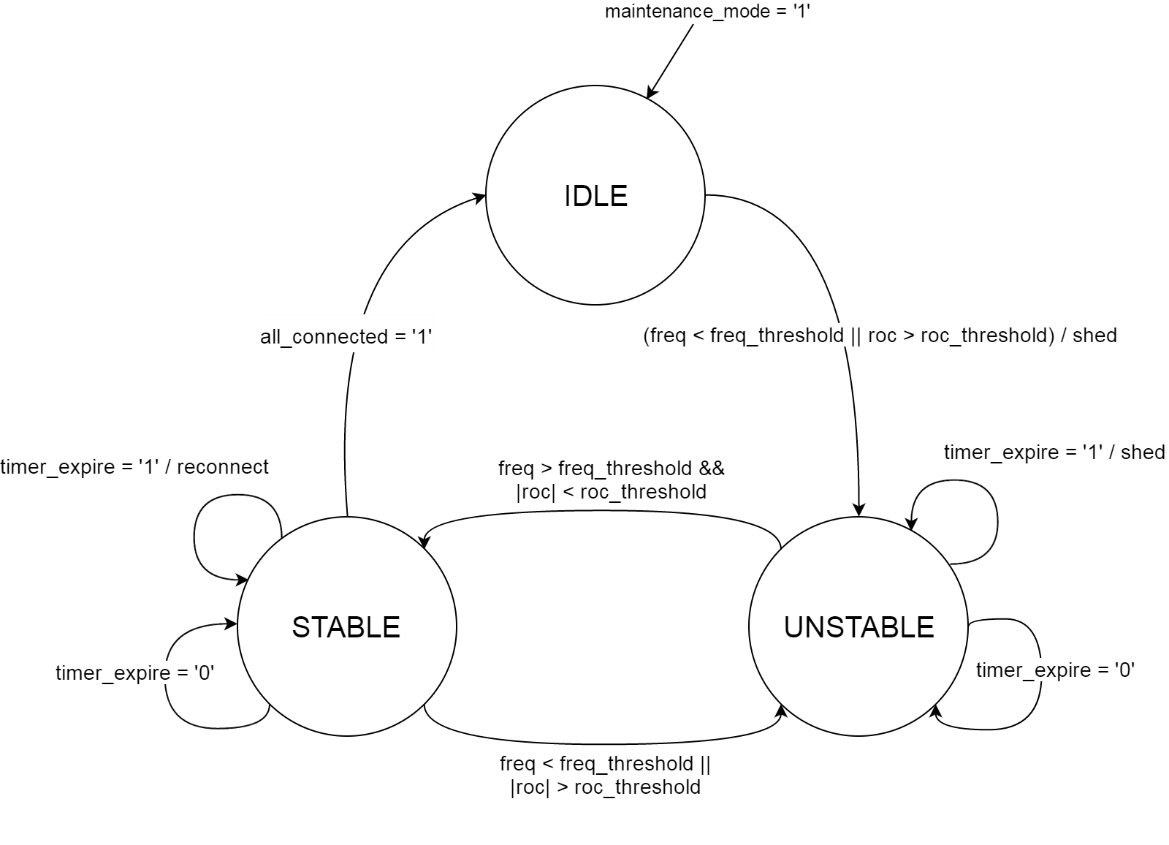
**The task tick counter is 32 bits in milliseconds:**

As of now, the hour counter will not overflow after 23, and will restart to 0 after 49 days. This does not impact the core functionality of the Frequency Delay whatsoever, but it can be fixed in future development.

**Reaction times** to shed a load are typically 5 to 20 ms which meets the real time requirements of the assignment (less than 200 ms). Other than the actual decision logic and writing to the load register (red LEDs), there is one thing adding additional delay that could be reduced: Periodicity of *Load Control* task (up to 5 ms), since it is the task that actually shed the loads. The execution condition of *Load Control* task could be changed from periodic to a completely notification-based trigger. This could further improve reaction times and reduce unnecessary operations.

**Extending number of loads supported** – only 5 loads are support as of now.

1. Instructions to Run Assignment
2. Programme the board with NIOS .sof file.
3. Set Eclipse workspace to 723a1g10/
4. Right click on g10a1, click clean all.
5. Right click on g10a1, click build all.
6. Run as NIOII Hardware - .elf should be already set up.
7. Appendix



*Figure 2. Manager Task FSM*

*Figure 1. Final Paper Design*