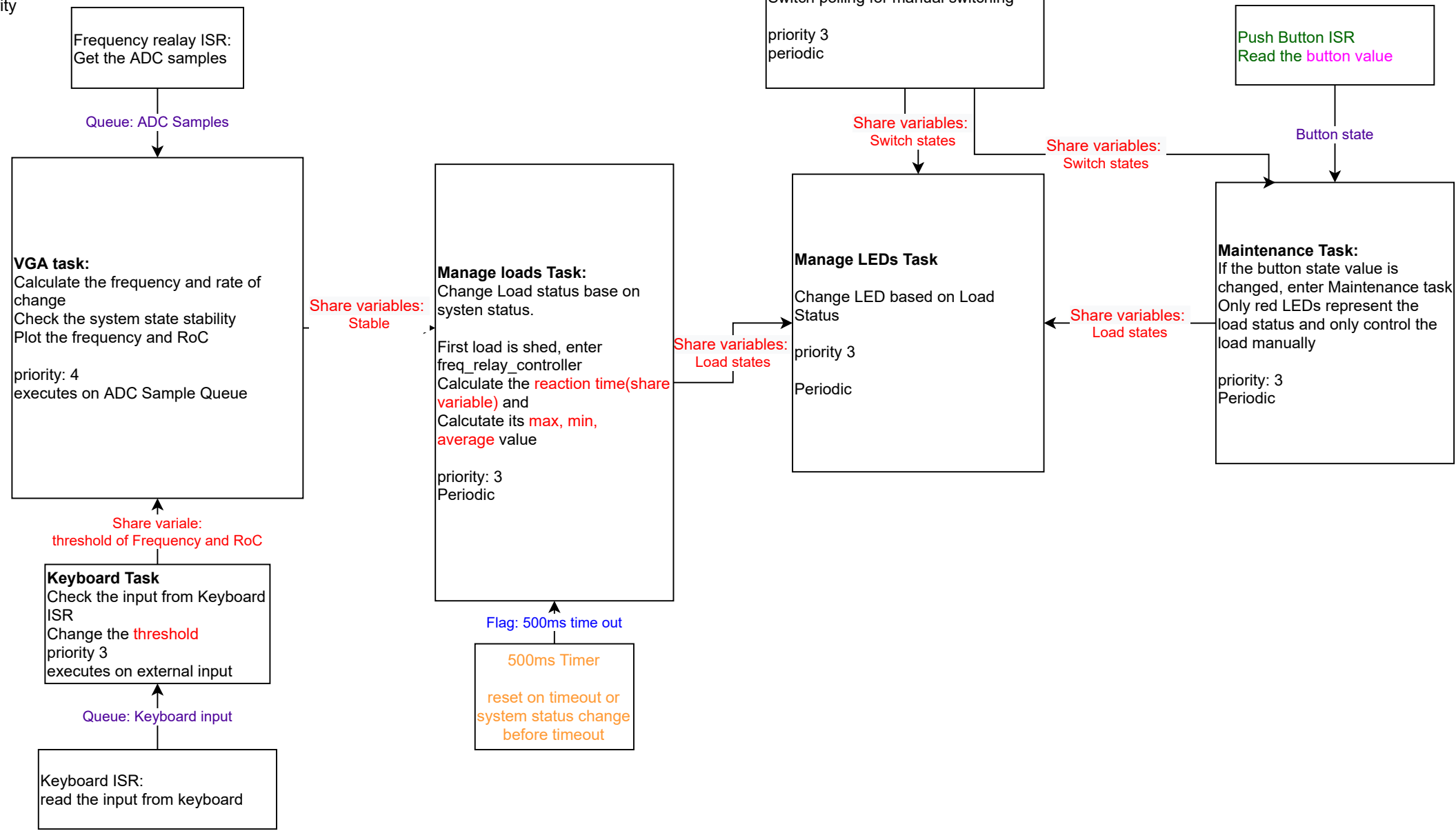
Final conceptual design

Larger number indicates higher priority



COMPSYS 723 Assignment 1 Group 19 Documentation

Instructions for running your assignment on a DE2 - 115 board
To run the project with the Altera DE2-115 board, please follow the instructions below

1. Connect the board to the computer, and connect the VGA screen, connect keyboard to the DE2-115
2. Program the board in Quartus II 13.0, with Quartus Programmer using provided freq_relay_controller.sof file
3. Open Nios II IDE, import test_freetos and test_freetos_bsp to workspace
   Build the Project
4. Create a new hardware run configuration and make sure the target connection is USB blaster
5. Select Run As Nios II Hardware

A description of our entire solution

We have six tasks in the system, LED Controller task, Switch polling task, PRVGA draw task, keyboard task, maintenance task and Load Management task.

In **PRVGA Draw task**, it would receive the ADC samples from frequency relay ISR, and calculate the frequency and rate of change, save them in the arrays. Also it would check the stability of the load network, if current frequency drops below the frequency threshold, or the rate of change is larger than the RoC threshold, the stability flag will be off. After the calculation, it would draw the frequency and rate of change plot on the VGA screen, show the state of the load network, total system run time, its threshold values of frequency and rate of change, the recent five reaction time measurements, the minimum, maximum, average and total reaction time.

In the **LED controller** task, it would modify the LED base on system status and load status. In the maintenance status, all green leds off. In the normal status, if load shed by freq_relay controller, red led turn off, green led turn on; if load shed by switch, only red led turn off.

Switch polling task which is checking the status of switch, read the switch value, because if a switch value is changed which means a load is being shed manually.

Keyboard task reads the input keys from the keyboard ISR, if 1 is pressed in the keyboard, the threshold value of frequency will increase by 1, if 2 is pressed then the threshold value of frequency will decrease by 1, 9 for increasing the threshold value of Rate of Change by 1 and 0 for decreasing the RoC threshold value by 1.

In **maintenance task**, if button 1/2/3 is pressed, the system will enter Maintenance mode, in Maintenance mode, only the red LEDs will be on to show the states of Loads, and there is no load shedding automatically, the load can be controlled by switching manually. If button 0 is pressed the system will be reset and need to run again.

In **Load Management** task, we check the status of the system, if the stability flag is 0, and the first load hasn't shed. The first load will shed as soon as possible. We start measuring the reaction time for shedding the first load between the freq_relay isr and the fist load is actually shed, then save it into the reaction time array and get the minimum, maximum and average value. After the first load is shed, the system will be controlled by the frequency relay. The load will shed or reconnect if the system remains unstable/stable for 500ms. If the system status changes within the 500ms, timer reset.

In these tasks, the tasks that need to receive the data from the queue are non-periodic, which are PRVGA task and Keyboard task, because both of them are triggered by ISR. And LED controller, Load management, Maintenance tasks are periodic.

Justification of our design decisions

Different tasks are doing different jobs, we decided to split the functionalities so that there is one main controller, load management receives flags as stability flag, frequency relay flag, the timer time-out flag. It changes an array of global load status variables, if the condition is met then the status of the load would be changed to 1 or 0, 0 means the load is shed. The LED controller, which is following the logic of the system and is supported by the other tasks and ISRs. The LED represents the load are controlled by the LED controller task using shared variables as load status and all on flag from the load management task. The VGA task is receiving the data from frequency relay ISR and data from the load management task. The tasks are supported by three ISRs, Button ISR, Keyboard ISR and frequency relay ISR. The measurement of the reaction is calculated based on the system's counter, we get the tick count as soon as the frequency relay ISR is triggered, and when the LED of the load is turned off, get another Tick count and calculate the difference between two ticks. In the default setting of the system, 1 tick is counted for 1 ms and it can be changed as the configTick_RATE_HZ(1000) in the FreeRtosConfig.h file.

Protect shared variables

We create semaphores with MUTEX. We use semaphoreTake and semaphoreGiveOnly, one task or ISR will be able to write to the share variables. Other tasks can only read from the share variable at the same time.

Communication mechanism

Global variables and FreeRTOS queues were the primary communication structures employed. For data that could be transmitted between tasks as a single variable, such as the device status, global variables were chosen. Data that would need to be stored several times before being read was stored in queues (frequency values). FreeRTOS Queue is an easy-to-use communication tool that provides a thread-safe first-in-first-out buffer that's ideal for transmitting frequency data between tasks. Queues also allow ISR data to be queued, allowing for simple and secure communication between ISRs and tasks.

Task Priorities

Prioritization was provided to the tasks based on their significance to the system's functionality and specifications. The Load Management task was given priority as 3 because it generates the majority of the data that other tasks use. The Load Management also provided the majority of the system's features, making it the most important component. The SwitchPoll and LEDController tasks are tied for the same priority. These activities were given equal weight in the method since they both include reading and writing the load status. Since controlling the loads is the system's primary feature, it's critical that the MainController role has the most up-to-date load status information and that it can easily write adjustments to the loads when it calculates a new load status. Finally, the VGA task is the highest priority role since it performs no system-critical functions, calculates the frequency and RoC, checks its stability and is needed to show data to the user.

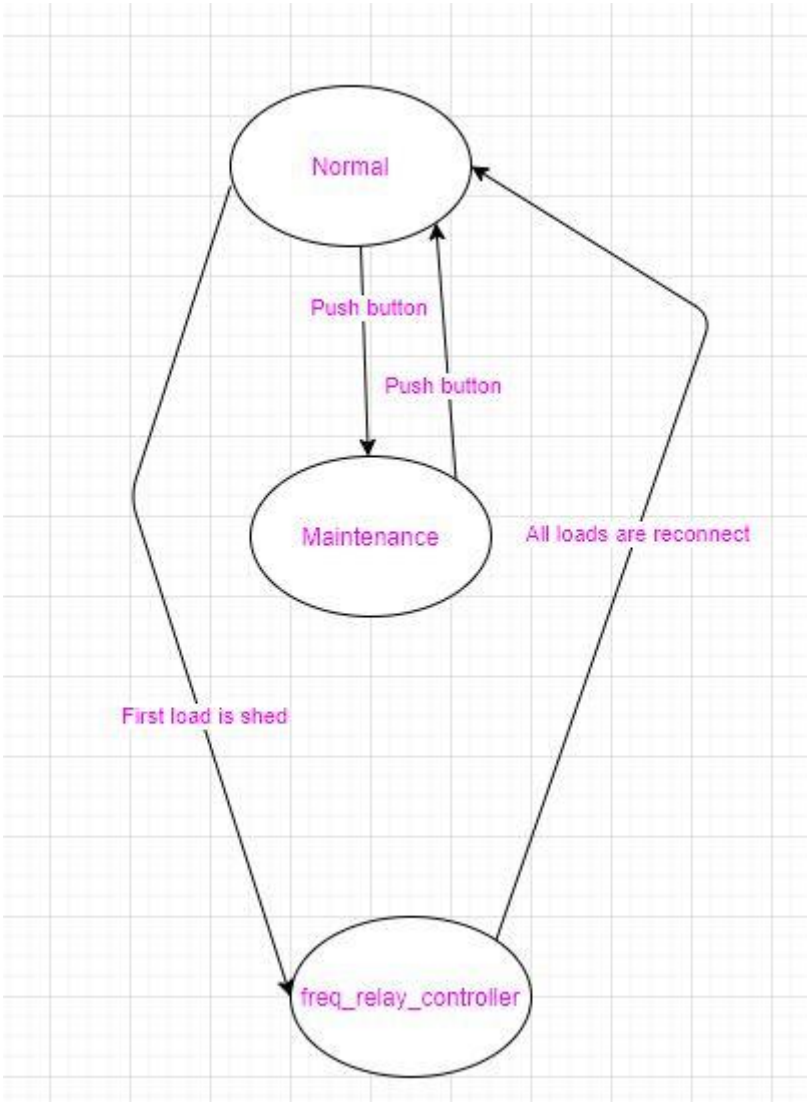


Fig: system mode

Evidence that you're engaging with FreeRTOS

Features of FreeRTOS we used:

Tasks, Queues, Semaphore, and a Timer were among the FreeRTOS features used in assignment one. Tasks were obviously needed in order for the FreeRTOS scheduler to be able to control multitasking of processes. We used six tasks, four of them were periodic, and we used the vTaskDelay() function to run them at a set interval until the scheduler was usable. Queues were used to pass data between ISRs and tasks, as well as determined frequency values between the mainController and VGAController tasks. As long as new frequency data was received or a 500ms timer had timed out, a binary semaphore was used to monitor access to the main controller. Before reconnecting/disconnecting another load, we used a FreeRTOS software timer to insure we met the 500ms of stability/instability requirement. We also used the FreeRTOS xTaskGetTickCount to figure out how long the device was up and how long it took for events to occur.

Task interaction:

Global variables, as well as queues, were used to communicate between tasks. The SwitchPoll task polled the load switches every 5ms and saved the results to a global

variable. The LEDController task also ran every 5ms and used a global variable to write the status of the loads to the LEDs. The Load management task was where the system's main logic took place. The load management reads and writes the status of the load switches and load LEDs. Load management task will take a queue from the FreqRelay ISR, which is in charge of queuing up the frequency data coming in, or a 500ms timeoutCallback, which is in charge of timing 500ms of stability/instability. The VGA receives data from the queue and calculates the frequency and rate of change. While the keyboard task receives data from the keyboard ISR, confirm the input from the PS/2 keyboard then change the threshold values.

Problems we overcome:
We achieve the requirements in the assignment brief.
Our system responds quickly, always less than 200ms .
The priorities of the task, because if the priority of the task is unsuitable, the task will be blocked by other tasks and make mistakes.

Limitation:

- The system design is based on FreeRTOS

Tasks assignment

Work items	Work assigned	Time taken
ISRs	Together	3 hours
VGA task	Together	8 hours
LED controller	Huiyi Huang	5 hours
Load management	Huiyi Huang	5 hours
Maintenance task	Huiyi Huang	4 hours
Keyboard task	Ninger Gong	2 hours
Switch Polling	Ninger Gong	3 hours
Debugging	Ninger Gong	8 hours
Documentation	Together	3 hours