Group 10 Assignment 1 Documentation

Joshua Gudsell (jgud007) and Sakayan Sitsabesan (ssit662)

### Instructions for running the assignment on a DE2-115 board

To run the project on an Altera DE2-115 development board follow the steps provided below:

1. Program the device using the Quartus Programmer tool and the provided .SOF file.
2. Open Nios build tools for Eclipse.
3. Create a new Nios II application and BSP from Template using the provided nios2.sopcinfo and the Hello World template.
4. Copy the contents of the main.c file provided and use them to replace the contents of hello\_world.c
5. Copy the provided FreeRTOS folder into the aforementioned Nios II application.
6. Build the application, then right click on the application folder and select Run As, 3. Nios II Hardware.

To use the project any of the switches from 0-7 can be used to control a load. Any of the push buttons excluding KEY0 can be used to enter maintenance mode. To configure the thresholds simply use the number row on the keyboard. Six values must be entered (three for frequency threshold and three for rate of change threshold) before pressing enter to transfer the temporary new thresholds to the actual threshold values. If an incorrect value is entered while updating the thresholds esc can be pressed to start again.

### Solution

o All the tasks, what they do

o Which tasks are periodic? Non-periodic? Why?

o A final diagram of your design (highlighting any changes from your initial paper design)

### Design decisions

Split of functionality

We decided to split the functionality so that there was one main controller, which was responsible for the logic of the system and was supported by other tasks and ISRs. The MainController task recieves data about the switch position through a shared variable that is populated by the SwitchPoll task. The MainController then performs logic and populates shared variables used in other tasks. The LEDs representing the loads are controlled by the LEDController task using shared variables provided from the MainController. The VGAController is a task that displays the data provided from MainController to the VGA display. The tasks are supported by three ISRs, the KeyboardISR, the PushButtonISR and the FrequencyRelayISR.

Protecting shared variables

Shared variables are protected by only writing to them in a single task or ISR. This ensures there will be no errors due to multiple tasks/ISRs writing to a shared variable at the same.

Communication mechanism

The main communication mechanisms used were global variables and FreeRTOS queues. Global variables were chosen for data that could be conveyed between tasks as a single variable such as the system status. Queues were used for data that may need to be stored multiple times before it is read (frequency values). FreeRTOS Queues are a very simple to use communication method which provide a thread safe first in first out buffer which is perfect for transferring the frequency data between tasks. Queues also provide the ability to queue data from ISR allowing for easy and safe communication between ISRs and tasks.

Task priorities

The tasks priorities were assigned based on their importance to the systems functionality and requirements. The MainController task was given the highest priority as this is where most of the data is generated that other tasks consume. The MainController also provided most of the system functionality and thus has the highest priority. The next lowest priority is shared by the SwitchPoll task and the LEDController task. These tasks were considered of equal importance to the system as they both relate to reading/writing the loads status. Since managing the loads is the main functionality of the system it is important that the MainController task has the most up to date information on the loads status and that it can write changes to the loads quickly when it calculates a new load status. Finally the lowest priority task is the VGAController as this task does not perform any system critical functions and is only required to display data to the user.

### FreeRTOS

Features of FreeRTOS used

The features of FreeRTOS used in assignment one were Tasks, Queues, a Semaphore and a Timer. Tasks were obviously required in order to allow the multitasking of processes to occur under the control of the FreeRTOS scheduler. We used four tasks, three of these were periodic and used the vTaskDelay() function to operate at every certain number of milliseconds once the scheduler is available. Queues were used to transfer data between the ISR’s and tasks, along with being used to transfer calculated frequency values between the mainController task and the VGAController task. A binary semaphore was used to control entry to the main controller as long as new frequency data had been received or a 500ms timer had timed out. We used a FreeRTOS software timer to ensure we complied with the 500ms of stability/instability required before reconnecting/disconnecting another load. We also used the FreeRTOS xTaskGetTickCount to calculate the system uptime and time between events occurring.

Task interaction

The communication between tasks occurred using global variables along with queues. The SwitchPoll task ran every 10ms and wrote the status of the load switches (SW0-SW7) to a global variable. The LEDController task also ran every 10ms and wrote the status of the loads to the LEDs from a global variable. The MainContrller task was where the main logic of the system occurs. The MainController reads the status of the load switches and writes the status of the load LEDs. The main controller is entered any time it can take a semaphore from the FreqRelayISR which is responsible for queuing up the frequency data coming in or 500ms timeoutCallback responsible for timing 500ms of stability/instability. The MainController computes the frequency values to display and queues them. The VGAController task receives frequency data from a queue populated by the main controller. VGAController also receives data surrounding key presses on the keyboard from a queue populated by the KeyboardISR and uses this to update the configurable thresholds which can then be written into the global threshold variables to be read by the MainController. The VGA controller reads from the currentState global variable to display the current system status.

Potential problems

We have overcome the problem of multiple tasks/ISRs writing to a shared variable by only writing to all shared variables in a single place throughout our solution. Any other FreeRTOS problems we have avoided?

### Limitations

|  |  |  |
| --- | --- | --- |
| Work Item | Worked on by | Time Taken (hours) |
| VGA output | Josh | 5 |
| Keyboard input | Josh | 5 |
| Configurable thresholds | Josh | 2 |
| Debugging | Josh | 5 |
| Documentation | Josh | 3 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |