COMPSYS 723 ASSIGNMENT 1: LOW-COST FREQUENCY RELAY

Louis Olsen-Stahl

Department of Electrical and Computer Engineering
University of Auckland, Auckland, New Zealand

Abstract

Power systems are designed to function at a certain frequency, though it can vary due to the supply and load in the network. Over-frequency happens when the power supply is in excess, while under-frequency occurs when there is an overload. A frequency relay is designed to detect these irregular power frequencies and react to them by either shedding or restoring loads to maintain a stable network. For households, loads could be controlled individually in order to make energy delivery more selective and efficient. A Low-Cost Frequency Relay (LCFR) can be used for this situation. It will measure the real-time frequency and shed or restore loads on the fly, keeping the overall power system stable.

1. Introduction

The aim of this project is to implement the software aspect of the LCFR on the DE2-115 development board for use in households. This frequency relay will control the load shedding and restoring based on the internal frequency of the board which will act as the household frequency of the power network. The embedded system is written in C, with concurrent tasks being implemented for multitasking on the real-time FreeRTOS operating system.

The shedding and restoring of loads will be set depending on the minimum current frequency and the maximum rate of change of frequency. If the system exceeds these limits, it will become unstable, and the system will shed a load within 200ms. If the system remains stable or unstable for 500ms, a load will either be restored or shed respectively. The system will return to the normal state once there are no more loads to restore.

The DE2 board peripherals such as the switches, buttons and the seven segment display are utilised in this project. In addition to this, external peripherals will also be used. These include the PS/2 keyboard for user input and the VGA out for displaying information visually on a monitor.

2. System Overview

The system utilises a finite state machine (FSM), comprised of three states: Normal, Load Managing and Maintenance.

The Normal state allows the user to manually toggle loads with the switches, while also allowing the system to take control and switch to the Load Managing state in order to handle loads if the system becomes unstable.

The Load Managing state handles whether loads need to either be shed or restored. After the initial load is shed within the 200ms requirement, the system will restore or shed a load if the system remains stable or unstable for a full 500ms. When in this state, loads can be manually turned off by the user using the switches, but not turned back on.

The Maintenance state is entered and exited when the push button key3 is pressed, while in the Normal state. While in this state, the loads can only be turned on and off using the switches. The system will not take control and manage loads

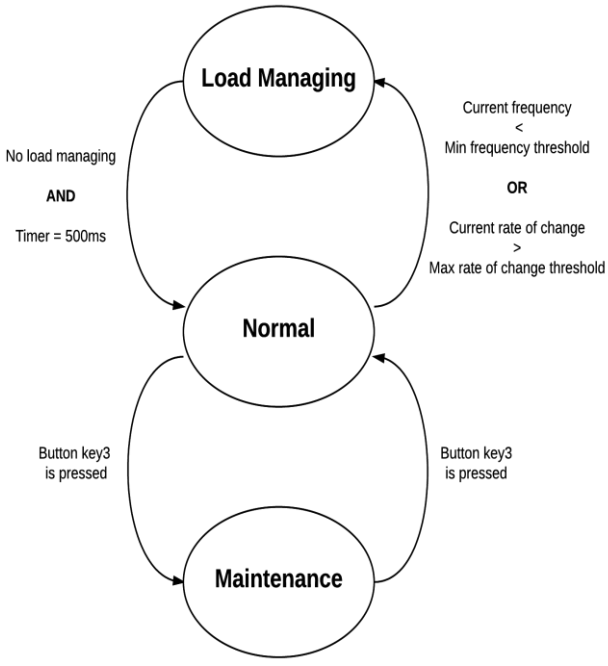

Figure 1:The finite state machine implemented

3. Tasks

3.1. Main Task

The main task contains the FSM for the system. When the main task is entered, it resets the 500ms timer start flag. It then checks the current state and reacts accordingly.

If in either the Normal or Maintenance state, the 500ms timer flag will be cleared to ensure the system enters the Load Managing state when required.

When the current state is Load Managing, the system will check if the 500ms timer is not running. If this is true, it will shed a load and capture the number of ticks (each tick is 1ms) at that moment and compared to the ticks when the system first became unstable. The time taken to shed the load is then calculated and stored in the time array to display on screen. The stability is saved and the 500ms timer is started. The current system stability will also be checked against the previous stability once timer is running. If the system stability has changed, the 500ms timer is reset as the stability switched within the 500ms limit. This will not happen if a load has just been shed. The current system stability is then finally stored for comparison. A vTaskDelay of 5 ticks is run. Because a task delay is being used, the main task is not periodic.

3.2. Switches Task

The first eight switches (SW0 - SW7) are set to be used as loads. A mutex semaphore for the current load and controlled load is then created. It is then checked to see if has been created and is available to be taken. When the semaphore is taken, the current state will be checked. If Load Managing, the current load will be updated by comparing the current loads against the switches and displaying that result on the red LEDs. When in either the Normal or Maintenance state, the loads that are currently on will be displayed through the red LEDs. After writing to the LEDs, the semaphore is released, and the value of the switches is displayed on the seven segment display. A vTaskDelay of 10 ticks is run. Because a task delay is being used, the switches task is not periodic.

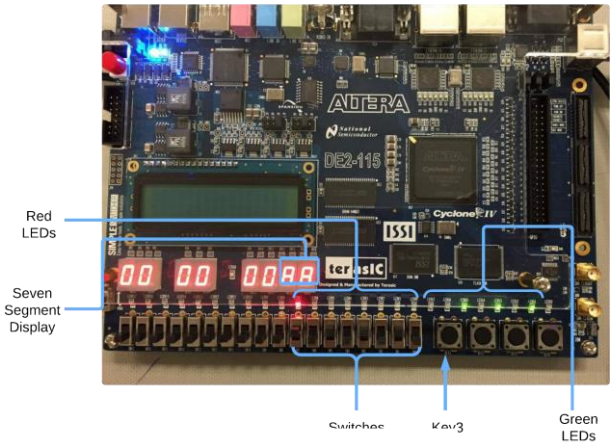

Figure2: Annotation of DE2 board identifying peripherals

3.3. Keyboard Task

The keyboard input array to be displayed on screen is first initialised to all zeros. The queue for the keyboard input is then checked to see if there are any messages, which are then read in and decoded to obtain the current number entered.

R key (0x2D): Changes the keyboard input to be for the maximum rate of change of frequency threshold.

F key (0x2B): Changes the keyboard input to be for the minimum frequency threshold.

Enter key (0x5A): Converts the keyboard input string to a double and saves it. The keyboard input is then cleared and the converted number is displayed on screen and the specified threshold value updated.

Esc key (0x76): Clears the keyboard input. The screen will be updated to 0.000.

If any key is pressed, it will first be converted using a lookup table into a readable character. A check is then made on that character to see if it is a digit or decimal point. If it is, it will be added to the keyboard input string, with the current string then being converted to a double for display on screen.
A vTaskDelay of 10 ticks is run. Because a task delay is being used, the keyboard task is not periodic.

3.4. VGA Draw Task

The pixel and character buffers are first checked to see if they are not empty. Based on the information being updated, the strings displayed on screen are also refreshed.

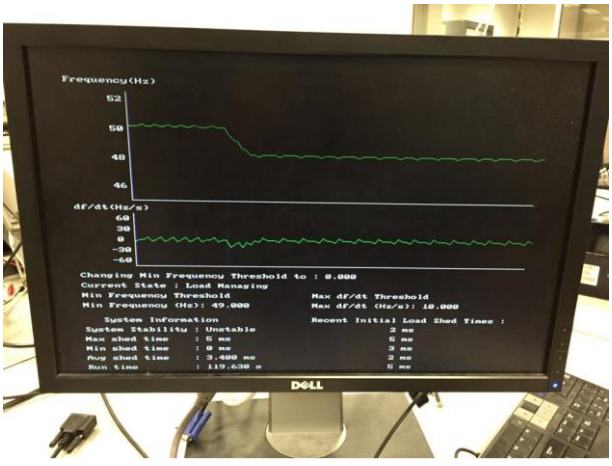

Figure 3: VGA displaying graphs and information

The information being displayed is as follows:

- A graph of the frequency
- A graph of the rate of change
- Current keyboard input
- Current minimum frequency threshold
- Current maximum rate of change of frequency threshold
- The current system stability
- The maximum, minimum and average load shed times
- The latest five load shed times
- The total run time of the system

A vTaskDelay of 10 ticks is run. Because a task delay is being used, the VGA draw task is not periodic.

4. Interrupt Service Routines (ISR)

4.1. PS2 Keyboard

The keyboard input is read in as an scan code that is specific to the PS/2 keyboard. When a key is pressed, it will return in the form: input byte-F0-input byte. In order to avoid registering F0 and the second input byte, an invalid key Boolean was implemented. The keys for miscellaneous buttons such as the arrow keys, home and page up were ignored as they mapped to other buttons that were used. The first byte read for those type of keys started with the byte E0.

When the key was pressed, first byte (the one needed) was saved to the keyboard input queue. The second time the ISR was entered, the F0 key would be the input byte. This was ignored with an invalid flag set. The ISR would return with no message saved to the queue. The third time entering the ISR would be for the repeated input byte. The invalid flag would then be checked, returning once again without saving anything into the queue.

Whenever the PS/2 ISR is entered, the byte was checked to see if it was within the lookup table created to convert the byte codes into readable characters. This would be used later in the keyboard task to input and display information.

4.2. Frequency Relay

This interrupt service routine updates the frequency data inputs and determines the system stability. The sampling frequency of the system is defined as 16000Hz. From that value, the signal frequency is then calculated. With the new signal frequency, the current rate of change is found with the equation:

$$\text{Rate of change} = \frac{(\text{frequency}_{new} - \text{frequency}_{old}).\text{Sampling frequency}}{\text{Number of samples between the two frequency readings}}$$

The absolute value of the rate of change is determined and this is sent to the frequency data queue to be displayed on screen.

System stability is also checked to see if a state change is required. If the current signal frequency is below the minimum frequency threshold or the current rate of change is greater than the maximum rate of change threshold, the system stability is set to false. When this happens in the Normal state, it will switch to Load Managing and count the number of ticks (milliseconds) at that point. This is then used when the initial load shed time is calculated in the main task.

4.3. Button

The interrupt service routine for the button is implemented in order to toggle between the Normal and Maintenance state. When a button is pressed, the edge cap register is cleared, and a mask is set to only look for a button push from key3.

When the button is pressed and the state is in either Normal or Maintenance, the state will switch from on to the other. The state will not change if the system is load managing.

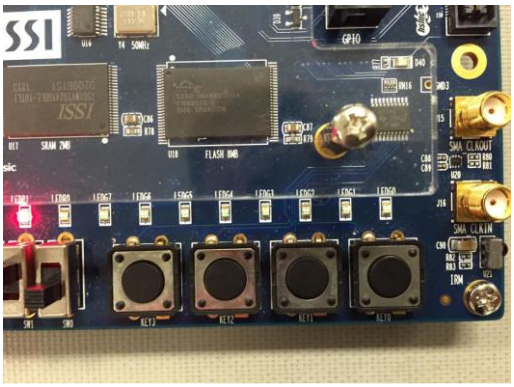

Figure 4: Frequency relay push buttons. Key3 on the far left

5. Design Decisions

5.1. Functionality Distribution

In splitting the required functionality of this system between various tasks and functions, the aim was to keep the functions and tasks in the most modular form. Modular functions and tasks with minimal coupling and maximum cohesion were aimed for to improve the portability, reusability and maintainability of the code.

The requirements of the system, displaying a VGA output with system performance metrics such as minimum and maximum load shed times, keyboard input to change the minimum frequency or maximum rate of change of frequency thresholds and load managing functionality were then analysed in such a way so as to achieve these goals. This resulted in four main tasks, the Main, Keyboard, PRVGA and Switches tasks, along with three ISRs and two main functions involved with the load managing.

The Main task dealt with the running of the main state machine and thus the Load managing, calling the shed and restore load functions. As a result, additional function calls can be easily added to each state, Normal, Load managing or Maintenance, within the Main task in order to expand the functionality of each state. Additional states could also be added to give additional functionality to the system as a whole. The shed and restore load functionalities were created as functions in order to improve maintainability and cohesion.

The Keyboard task deals with all keyboard input codes, decoding the keys entered to determine which threshold the user wishes to change along with the number they are currently inputting. This information is processed within this task with only useful and necessary information being relayed to other tasks as they are required. An example of this would be the final number entered as a double instead of a series of characters to set the new threshold value.

The Switches task only deals with the input switch values indicating which loads the user has turned on, and uses this information to update the current loads value indicating the loads currently turned on. The switches task also displays the current switch value, which is not necessarily the current loads value if the system is in the load managing state, on the seven segment display, to which this task has exclusive access. This provides a modular, reusable and maintainable piece of code due to the low coupling and high cohesion present.

As only the VGA task requires the system metrics, these values are calculated within this task. As a result the VGA task contains all the necessary functionality to draw and display the required data to a connected monitor, resulting in a modular and easily modifiable piece of code.

Several ISRs were also used in this system to deal with the event driven intake of data, such as keyboard input, button presses or the new frequency data. As these data are event driven they need to be captured quickly and not lost, ISRs were set up and used to pass data to the required task, such as the keyboard or VGA display, effectively.

5.2. Communication Mechanisms

A number of different communication mechanisms were used to pass data between the different tasks running in this system. Some global flags, such as the system stability flag systemStable, were used to pass simple information such as the current system stability between tasks. The system stability was only written to in the Frequency relay interrupt service routine when there was a change in system stability, with other tasks and functions checking this flag to determine the systems current stability state and derive the corresponding actions required, such as resetting the 500ms load shed timer.

Other global variables were used for some data that was written to more frequently and from a number of different tasks. This included the controlled and current loads variables indicating what share of the loads were switched on or controlled at a given time. Due to the multiple locations where this data could be written to, mutex semaphores were used to ensure mutually exclusive access to these variables and prevent erroneous operations with these variables. The procedure followed to acquire one of these mutex semaphores was to first create and check the validity of the semaphore before requesting it. This request also featured a 100ms timeout to reduce the possibility of deadlock and improve the robustness of the system.

Finally, after acquiring a mutex semaphore and performing the necessary actions on the shared variable, the mutex was then released. Further information concerning the mutex semaphores chosen to protect the access to shared variables in this system can be found under Mutex semaphore in the Shared Variable Protection section.

Queues were also utilised to facilitate data transfers in which a FIFO buffer would be effective, such as transferring the keyboard input byte codes to the keyboard task for decoding or passing the new frequency values from the relay to the VGA to graph when it was updated. Queues were effective in passing this data as the one communicating task only received data, sent from its corresponding ISR, which was used to obtain the recent data for processing. The queues are initialised in such a manner so as to reduce the likelihood of a queue overflowing, as a known number of frequency updates will be queued and the keyboard only registers six key input numbers.

Queues allowed the effective and efficient transfer of this data through the use of FreeRTOS's queue API functionality. The xQueueSendToBackFromISR function is designed specifically for safely adding messages to FIFO queues from ISRs. This was used to pass the data from the keyboard and frequency relay ISRs to the respective tasks effectively and safely. The corresponding tasks then read from the queues until no messages remained using the uxQueueMessagesWaiting and uxQueueReceive functions.

5.3. Shared Variable Protection

5.3.1. Mutex Semaphore

Mutexes are similar to the more standard binary semaphore, however they include a priority inheritance mechanism not present in the FreeRTOS binary semaphore. This priority inheritance mechanism allows a task which has taken the mutex to raise its priority to that of the highest priority task which could also attempt to take the mutex. As a result a mutex semaphore is more effective in ensuring mutually exclusive access to a shared resource.

In this frequency relay, the Switches task, shed load and restore load functions all make use of a mutex semaphore to make changes to the variables containing the data concerning the number of controlled and currently switched on loads. Both of the restore and shed loads functions are called from the finite state machine in the main task. Therefore if the Switches task takes the mutex its priority will be raised to that of the main task.

This priority inheritance resolves the occurrence of priority inversion, which occurs when a low priority task has access to a mutex but is interrupted by a higher priority task also wanting access to the mutex. Once either the Switches task, restore loads or shed loads functions have finished working with the mutex variables, the mutex is released and if it was held by the switches task, the switches task would lose the temporary priority increase it gained and return back to its own priority level.

5.4. Task Priority Assignments

The chief purpose of this system was to effectively manage loads switched on by a user to help balance and maintain the power supply frequency at a desired level. As a result, as soon as the system becomes unstable the first load was required to be shed within 200ms. Consequently in designing this frequency relay the response time to this change in stability was taken to be of the highest priority, resulting in the below task priority assignments.

Table 1: Task priorities

Task	Priority
Main	14
Keyboard	13
Switches	12
VGA Draw	Idle Priority + 1

The Main task, which runs the finite state machine containing the Load Managing state and its subsequent calls to the load shed and load restore functions, is assigned the highest priority due to its direct effects upon the system response times. Therefore the high priority assigned to this task ensured the scheduler recognises its importance and schedules the task accordingly.

Of the three remaining tasks, the Switches task was the most directly related to the loads managed by a system, as it updated the loads the user currently desired to be powered on, and was therefore assigned the second highest priority of the four tasks.

The remaining two tasks were not as crucial to system functionality, with the keyboard task only updating the current minimum frequency and maximum frequency rate of change thresholds, which would not be changing rapidly and did not require as much response as the previously mentioned Main or Switch tasks. The keyboard task was however deemed to require a higher priority than the VGA display task, as updating the VGA display takes a large time to run in comparison and is less crucial to the systems performance than updating the frequency thresholds. Due to the computational demand of the VGA task it was given a priority only slightly higher than idle so that it affected the performance of the other tasks as little as possible.

6. Discussion

6.1. Performance

The main performance requirement of this system was to switch a currently active load off within 200ms of the system becoming unstable. This requirement was easily achieved by the frequency relay designed in this project, with the relay achieving a typical maximum load shed time of five milliseconds and a minimum of one millisecond. This performance time also appears to reach the maximum as a result of the task delay given to the Main task, which is responsible for enacting the finite state machine that sheding the initial load. As a result further decreasing of this task delay could reduce the typical maximum response time further if stricter timing requirements were enacted. Further analysis into the effects of a pre-emptive or other alternative scheduler types could also provide insight into ways of optimising the system for additional functionalities or tighter timing constraints.

6.2. VGA display

The VGA display of this system could be developed further improve the resolution, refresh rate or user interface. Although these improvements would likely not have a significant effect on the actual performance of the system, they would improve the end user's experience. Thus the ability of the system to interact effectively with its consumers would be improved and a better overall product would be provided to customers.

7. Conclusions

The functionality of this system along with the priorities assigned to the given components has been proven to be an effective and efficient design. This success was signified by the rapid response time of the system in correlation to the give n 200ms load shed requirement.

The additional functionality such as VGA display, keyboards input and 500ms timing between shedding and restoring a load has also been achieved to a high standard.

8. References

FreeRTOS. (n.d.). Retrieved April 26, 2015, from http://www.freertos.org/

Cheng, M. (2007, October 23). A Simple RTOS. Retrieved April 1, 2015, from http://jwtanner.com/rtos/doxygen/html/index.html