

Introduction to LabView

Chapter 3 part ii: Power and Emergency Controls using Finite State Machine Implementation

This note and the code can be downloaded from the course web pages as a ZIP file.

This chapter is provided in the form of a worked exercise. The objective is to implement the basic start-up and emergency controls for a typical mechatronic control device operating in a safety critical environment.

I will discuss the requirements and technical background in separate notes. However, some information is essential to understand the necessity for this particular approach.

Turning a machine on and off requires more than just a simple on-off switch. Large and powerful machines can inflict severe damage and injuries to personnel if they move at the wrong time.

Australian and International (ISO) standards specify the types of safety controls that machines should be fitted with. For example, refer to AS 4024.1 1996 Safeguarding of Machinery (See part 1: General Principles).

The machine will be protected with an emergency stop circuit. This circuit is normally unbroken. It can be interrupted by any of the following:

- One or more emergency stop switches -- large red pushbuttons in prominent locations
- Switches that detect when an operator enters an enclosure or opens a protective guard
- A sensor that detect overheating or a fire
- A device that detects failure of the control computer

Normally this circuit is wired to the main power on off relay or circuit breaker: as soon as the circuit is interrupted power to the machine will be cut off and brakes automatically applied. It is very important that the machine is not restarted automatically if the emergency circuit is restored.

Turning on the power for a large machine is often not instantaneous: it can take several seconds for the power to reach a sufficient level for reliable machine operation. In the example we are about to see it takes two seconds for the power to reach a safe operating level. If the power does not reach the required level, it is likely there is some fault with the power system and the machine must be reset before being restarted.

We assume the machine has two basic operating notes: manual and automatic. To keep the example simple, the automatic mode generates sinusoidal motion. In the manual mode a knob can be used to adjust the position.

In the example, the emergency stop circuit is simulated with an emergency button. Turning on the power is also simulated: a random variable ensures that power faults will be generated every now and again.

This control arrangement is typical of many mechatronic machines such as robots, machine tools, and process plant. A finite state machine provides an ideal basis for the design that starts with a state diagram.

The handwritten notes that follow provide a draft specification followed by implementation notes that I wrote as I prepared the worked example code. I noted the times as I worked and all mistakes I made. In several places I had to make changes to my draft specification and I reworked design details.

The LabView code combines several of the earlier examples. The finite state machine is built in the same way as the examples in chapter 3. The master VI implements two separate real-time threads: one operates

the state machine and the power supply simulation and the other operates the sine wave generator when required. An uninitialised shift register module (USR) is used to store all state and system variables so they can be accessed safely by the different threads.

You will need to explore the code for yourself and follow the notes to gain a full understanding.

Exercise

The design is incomplete. With deliberate oversight I omitted a method for turning off the power without using the emergency stop button. Your task is to change the basic state diagram and the code to implement this. You can choose whether to use the reset button or the power button.

I hope you will find that this modification is quite simple to implement. You will need to understand the example code and this takes some time. However, in doing this exercise you will begin to understand why finite state machine designed code provides a model that is easy to understand and maintain. It is not the only way to design code for discrete control systems like this. However, it has proved to be a model that provides cost-effective results and low maintenance costs.

The handwritten notes provide a model of the approach to software development that I would encourage you to follow. Start on paper: define a draft specification. Perform your coding work one piece at a time, writing implementation notes as you proceed. When you find the design is deficient, STOP. Return to paper work and re-define the specification before making changes to the code. Remember: *if you do not know enough to write the specification in detail, you do not know enough to write the code.*

I expect you to write specifications, design notes, implementation notes etc. to support your design work. In competitive projects in particular, these notes and records of testing will be part of the evaluation process: you will be expected to produce them for inspection. It will be obvious if they are written up after the coding is finished: write them as you go. While neatness is an issue, do not attempt to bury your mistakes. Use your mistakes to learn not to repeat them.

Read the LabView application note “LabView Development Guidelines (see “View Printed Manuals”)” under the Help menu. Chapters 1, 3 and 5 are the most valuable at this stage.

VI List

FSM-Power-Master.vi:	Top level VI (see panel and diagram below)
Power-state.vi:	Sub-VI that implements the finite state machine
Power-sim.vi:	Sub-VI that simulated the power rise after switching on, and fall after switching off. Expose front panel to see meter that will show power level while the system is operating.
Power-state USR.vi:	Sub-VI implementing USR to store state information
FSM-Power.vi:	partially completed implementation
Power-sim-test.vi:	Stand-alone VI for testing: used to find and correct design errors shown in implementation notes
Power-USR-test.vi:	Stand alone VI for testing USR module: coding error found as described in implementation notes
Power-state.ctl:	Strict type-def defining system state information cluster



