COMPUTER SYSTEMS 723 CRUISE CONTROLLER

William Chao
Leighton Jonker

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

Abstract

When it comes to the creation of embedded software, there are multiple ways to approach the design process. One such process is the model-based approach. The model-based approach involves the development of functional specification models which are then implemented using a synchronous language. For this assignment we have been tasked with the designing of a functional cruise control model which will then be implemented using Esterel.

1. Introduction

In Computer Systems 723 we have been tasked with the creation of a cruise control system, so we can gain hands-on experience in designing embedded software using a model-based approach. We will first start the assignment by creating a functional model that adheres to the specifications we have been given as per the assignment brief. Next, we will implement our model using the synchronous language Esterel. Our goal is to have created an embedded system which is executable, reactive and adheres to all the given specifications.

2. Specifications

The Cruise control system will take the following inputs:

- ON (Binary)
- OFF (Binary)
- Resume (Binary)
- Set (Binary)
- QuickAccel (Binary)
- QuickDecel (Binary)
- Accel (Float)
- Brake (Float)
- Speed (Float)

And produce the following outputs:

- CruiseSpeed (Float)
- ThrottleCMD (Float)
- CruiseState (Enum: OFF, ON, STDBY, DISABLE)

These inputs and outputs will allow the cruise control system to effectively control the speed of the car within the speed range of 30 and 150 kilometers per hour, with increasing or decreasing increments of 2.5 kilometers per hour.

Initially the cruise control system will be set to OFF when the car is first turned on, ignoring other inputs.
The system will then go ON when the ON input is sensed, or OFF when the OFF input is sensed, if the system is currently ON.
While the system is ON and the car speed is within the limit with the accelerator not being pressed, the cruise control system will regulate the speed of the car.
When the accelerator is pressed or the speed of the car exceeds the speed limits, the system will go into the DISABLE state, and will thus be disabled.
The way to get out of the DISABLE state is by having the accelerator not being pressed and the speed of the car being within the limits.
If the brake is pressed then the control system will go into the STDBY state and wait to be re-enabled again through the Resume input, additionally depending on if the accelerator is being pressed and the current speed of the car.
Additionally, it is assumed that the acceleration and brake pedals will never be pressed together.
When the state of the cruise control system is set to OFF, the accelerator pedal will dictate the speed of the car.
The cruise control system will regulate the speed of the car using a proportional and integral algorithm with factors Kp = 8.113 and Ki = 0.5 while in the ON state. The integral part of the regulation will be reset when cruise control transitions to on, and frozen when the throttle output is saturated. The throttle output (ThrottleCMD) will be saturated at ThrottleSatMax (45%) so that car acceleration is limited, for the comfort of the passengers.
The speed at which the car is regulated at will be set to the current speed of the car when the system is turned ON or at any time the SET input is signaled.
Activating the QuickAccel or QuickDecel inputs will increment or decrement the cruise speed by 2.5 kilometers per hour for each time pressed.

3. Diagrams

First, we had to create functional specification diagrams. The diagrams would help us get a better understanding and handle of the system that we were tasked to create as well as help us when it came to the creation of our system. Below are images of our model during development.

First, we started with a simple model of a Cruise control system which was inputted simple inputs such as ON, OFF, Accel, Brake etc. with outputs such as Cruise State and Cruise Speed.

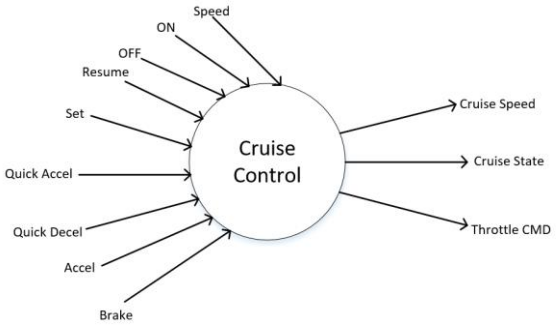

Figure 1: Model Phase 1

A model such as this was incomplete and simple in our eyes and we felt we needed to break up the signals into more specific sub-modules so that implementation and understanding would be easier. This prompted us to split the previous Cruise Control module into two separate modules called State Resolver and Control Unit. With this refinement we could now easily distinguish between the two main areas of this system; the states and control.

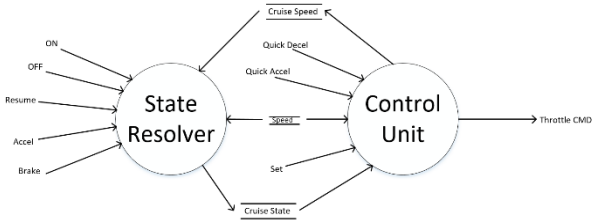

Figure 2: Model Phase 2

Next, we decided that we could even further separate the Control Unit module into the Cruise Speed Setter and the Throttle Controller for good object-oriented design.

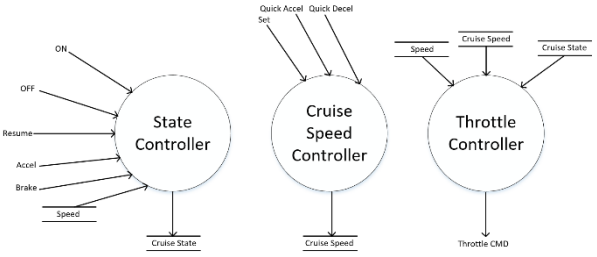

Figure 3: Model Phase 3

Lastly, we now needed to include monitors to monitor certain inputs from either the user or other modules in our design which could then pass the correct signals to the appropriate modules so they can continue correct functionality. We added an Accel Monitor, Brake Monitor and Speed Monitor. The final version of our specification model is shown below:

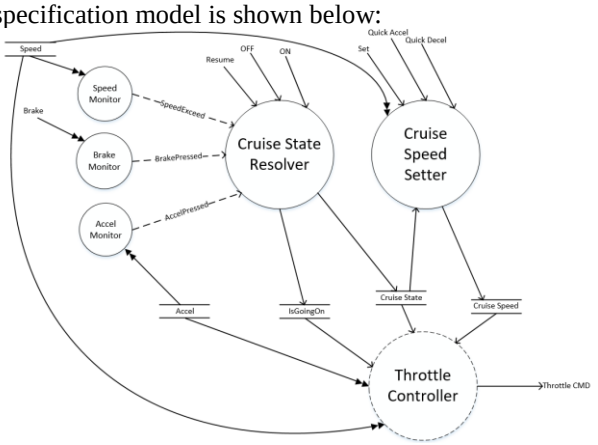

Figure 4: Model Phase 4

After verifying that our current model fit all the requirements that were stated in the project brief we then started on the creation of the Finite State Machines for each module.

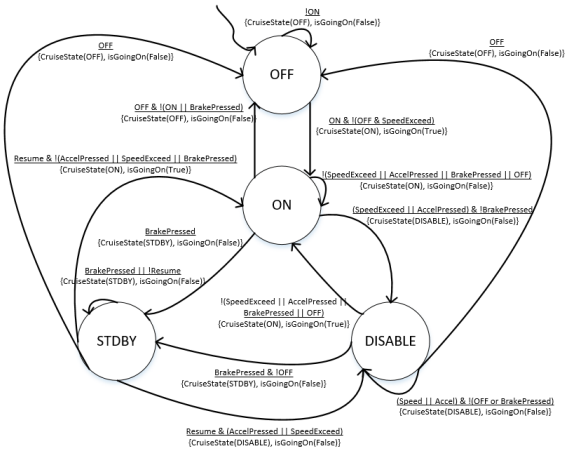

Figure 5: Cruise State Resolver FSM

The Cruise State Resolver is one of the most complex modules in the system, and decides what state the cruise controller is currently in. It takes multiple inputs ranging from the current car speeds to binary inputs like OFF or ON and emits the current state the car should be in.

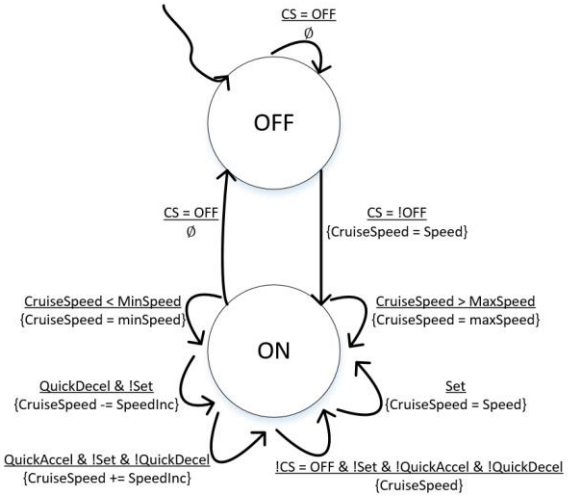

Figure 6: Cruise Speed Setter FSM

The Cruise Speed setter module is designed to set the speed at which the cruise control system will run the car at when the cruise controller is active. If the speed exceeds the minimum or maximum threshold at which this assignment has provided for us, it will emit a signal that will return the cruise speed to the appropriate and correct levels. All self-transitions in the ON state have an implicit !CS=OFF condition so when CS=OFF it will change to the OFF state immediately and will never have a causal issue such as when the cruise state is OFF and SET is pressed.

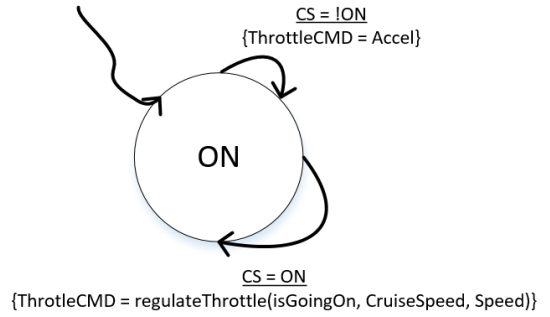

Figure 7: Throttle Controller FSM

The Throttle Controller considers the current cruise state, cruise speed and speed to calculate what to regulate the throttle of the car at. The signal isGoingOn is used to reset the integral of the controller if the cruise control resets. RegulateThrottle is a function written in C which has been provided to us. It simply calculates what the throttle output percentage should be to adjust its speed compared to the cruise speed.

The Finite State Machines for each of the monitors are much simpler than the other Finite State Machines as they mainly focus on one input and act accordingly as opposed to the other modules which are inputted multiple signals and sometimes also output a considerable number of signals.

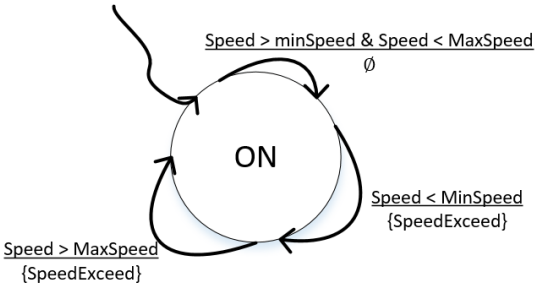

Figure 8: Speed Monitor FSM

The Speed Monitor is given the current speed of the car and calculates whether the speed exceeds the given min or max limits for car speed, emitting the signal SpeedExceed if it does.

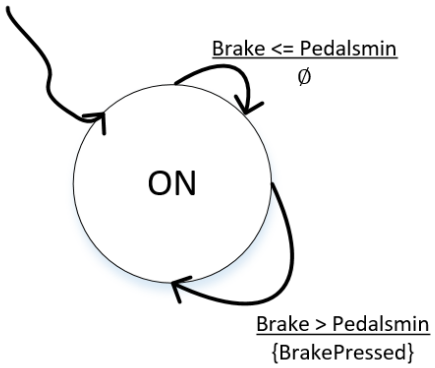

Figure 9: Brake Monitor FSM

The Brake Monitor takes the brake input and emits an output called BrakePressed if the brake input signal goes over the Pedalsmin threshold to signify if the car's brake has been pressed.

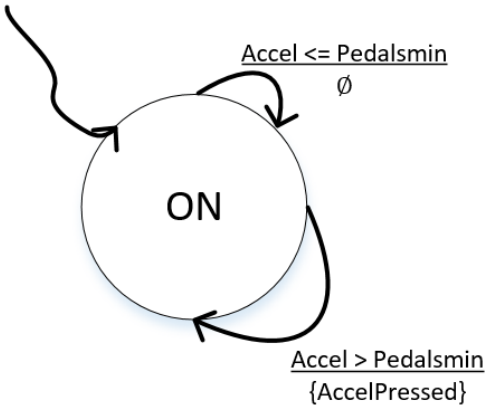

Figure 10: Accel Monitor FSM

Finally, the Accel Monitor takes the accelerator input and emits an output called AccelPressed if the accelerator input signal goes over the Pedalsmin threshold to signify if the car's accelerator has been pressed.

4. Implementation

Once we had finalized our designs it was then time to start the implementation. Although we had never used Esterel for programming before, we understood how the language worked and decided that the main issue we would face would be the initial creation of the diagram. Using the given specifications and the Esterel syntax, we found that some of the specifications given to us were difficult for us to implement as we didn't know how the syntax for those specifications worked. Thankfully with the help of the lecture slides, TAs and provided reading, we understood the language more, allowing us to be able to successfully implement our design in Esterel. We started with the conversion of the given specifications of the cruise control system being converted individually into pseudocode and eventually Esterel modules following the creation of the functional model and Finite State Machines. Once the individual modules had been created and finalized, they needed to be integrated with one-another. The integration was the most difficult part, as occasional errors would start to appear, even though all modules worked perfectly by themselves, the integration would cause new conflicts in between modules. Once the integration was complete the system would then need to be verified to ensure it met all the provided specifications in the brief.

5. Validation

Once we had integrated and implemented our system it was time to test and ensure that it worked. We were provided with input and output vectors with the assignment that were meant to be used to test our system and ensure that it passed all the specifications that were presented to us. The testing was originally meant to be run using the "Recorder" feature in the Esterel simulator. However, the vector files that were meant to be input into the simulator were unable to be used as the recorder did not support the vector files. As a result, instead of automatically testing our system, we manually tested all the scenarios that were provided in the input vector file along with some other scenarios that the input file may have missed. Once we tested the scenarios and made sure that they were correct we were confident that our implementation of the cruise control system was up to standard.

6. Conclusions

In conclusion, our implementation of the cruise control system met the requirements that were set upon us. We have been able to successfully create a system that runs on Esterel, which is able to take certain inputs, and depending on those inputs, manage a cruise control system designed to keep a car at a set speed. It also had additional features such as going on standby and disabling itself when certain inputs are signaled (Accelerator, Brake etc.). The use of Esterel for this assignment proved slightly difficult at first, but as we got more and more used to using the language, as well as seeing the similarities between it and VHDL – a language we were familiar with, we were able to understand the language much more easily.