COMPSYS 723 Assignment 2

Ahmad Umar
Department of Electrical and Computer Engineering
University of Auckland
Auckland, New Zealand
auma020@aucklanduni.ac.nz

Rajan Gupta
Department of Electrical and Computer Engineering
University of Auckland
Auckland, New Zealand
rgup198@aucklanduni.ac.nz

***Abstract*—Model-based approach is used to create an embedded software. In this report we have covered the design and specifications of a functional cruise control model implemented using Esterel V5.**

***Index Terms*—Cruise Control System, Concurrency, Esterel V5**

I. Introduction

This report is for COMPSYS 723 Assignment 2. The assignment required us to implement a Cruise Control system using Esterel V5. The brief details the behavior of the system which is described using a Finite State Machine (FSM). We implement the model using Esterel V5, a synchronous language. The assignment specifications are covered in the next heading, followed by details of our design. A readme file has been included which contains instructions to run the system.

II. Specifications

Using Esterel, the inputs from the environment to the Cruise Control System will be the inputs to the simulator GUI. The overall input-output behavior of the Cruise Control system is described using the context diagram as shown in Figure 1. The system receives nine inputs and these are summarized below:

1) On (pure): Enables the cruise control
2) Off (pure): Disables the cruise control
3) Resume (pure): Resumes the cruise control after braking
4) Set (pure): Set the current speed as the new cruise speed
5) QuickDecel (pure): Decrease the cruise speed
6) QuickAccel (pure): Increase the cruise speed
7) Accel (float): Accelerator pedal sensor
8) Brake (float): Brake pedal sensor
9) Speed (float): Car speed sensor

The system produces three outputs and these are summarized below:

1) CruiseSpeed (float): Cruise speed value
2) ThrottleCmd (float): Throttle command
3) CruiseState (enumeration): State of the cruise control. It can be one of: OFF, ON, STDBY or DISABLE.

The system starts in an Off state. This means the CruiseState output is set to OFF. During the OFF state, the system is immune to any input. The simulator GUI has six buttons for input to the system. These are: On, Off, Resume, Set, QuickAccel and QuickDecel. The cruise control system is turned ON or OFF by the On or Off input, respectively. The cruise system switches from OFF to ON state only when an On input is sensed; this sets CruiseState to ON. The system goes into OFF state when an Off input is set and this sets CruiseState to OFF.

Furthermore, the assignment briefs gives cruise control parameters, SpeedMin, SpeedMax and SpeedInc. The minimum value is *SpeedMin = 30 km/h* and top speed is *SpeedMax = 150 Km/h*. While the system is in ON state, the system maintains the Speed as long as it is within the set speed limits. An input of QuickAccel or QuickDecel, increases or decreases the speed,respectively, by a given amount. The step size is stored in the variable SpeedInc and for the purposes of this assignment, *SpeedInc = 2.5 Km/h*. When accelerating or decelerating from the inputs, if the value goes past SpeedMax or SpeedMin, the speed is saturated SpeedMax or SpeedMin, respectively.

The outputs from the Cruise Control system are: CruiseSpeed, ThrottleCmd and CruiseState. All these are displayed in the simlulator GUI (*Refer to Appendix I for screenshots of the GUI*). The output CruiseSpeed is calculated based on the inputs and is displayed to the user to show the speed being maintained by the system. The *ThrottleCmd* uses an external function written in C called *regulateThrottle*. This function uses factors, K_p and K_i, and calculates *ThrottleCmd* using a proportional and integral algorithm. The value of ThrottleCmd is saturated to a user defined value. In our case this is 45 percent and is stored in the variable ThrottleSatMax. This limits the car acceleration to ensure comfort for passengers. The output CruiseState has four possible values: 1(OFF), 2(ON), 3(STDBY) or 4(DISABLE).

The *CruiseState* is changed depending on the inputs. If an *Accel* input is sensed or the Speed is outside the speed limit, then the *CruiseState* is set to DISABLE. The state goes back to ON only when both the *Accel* input is removed and the Speed is within the defined limits. In this case, the previous set cruise speed value is used. The Set input changes the cruise speed value to the value of input Speed (can be changed in the GUI).

Regarding the *Brake* and *Accel* inputs, these are given in the GUI and can be passed any number. However, a *Brake* and *Accel* are only sensed as inputs when their values exceed a given threshold. In our case, the threshold is *3 percent* and is stored in the variable: *PedalsMin*. Given the threshold is exceeded for Brake, the cruise system is interrupted and CruiseState is set to STDBY state. A *Resume* input can put the system back into ON state given there is no *Accel* and the Speed value set is within the limits. In this both the conditions are met, the system uses the last set cruise speed.

III. Diagrams

Figure 1 below shows the top level context diagram for our Cruise Control system. It shows the inputs that will be input by the user via the simulator GUI and the outputs will be shown via the simulator GUI as well. *Refer to Appendix I for screenshots*.

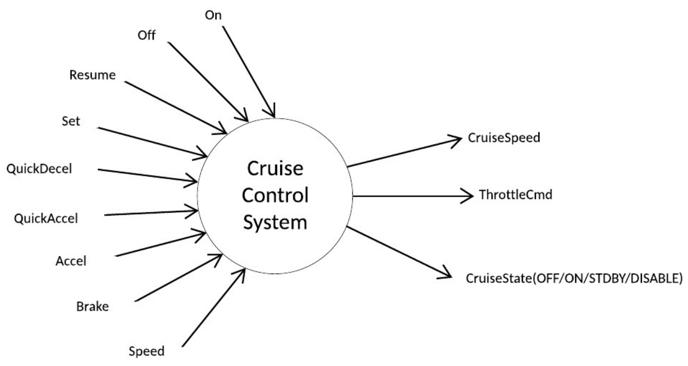


Fig. 1. Context Diagram of the Cruise Control System.

We then decided to break the top level context diagram to a low level one to show the additional six modules we have implemented to create a functional cruise control system. This low level context diagram is show in Figure 2.

Figure 2 shows all the inputs and outputs for each module. We have used Internal Signals to pass values between modules which are used to determine the outputs. The three main modules are: *CruiseStateController*, *CruiseSpeedController* and *ThrottleController*. The cruise control system runs these in parallel. The additional complimentary modules are used to check for certain conditions which carry out some calculations and use internal signals which are then passed to the main modules. *More detail on each module is given in the section IV*.

Figure 3 shows how we implemented the CruiseStateController as an FSM. We realised we have different states for the system and different data and information signals control the transition of each state. The FSM shows internal signals and inputs from GUI.

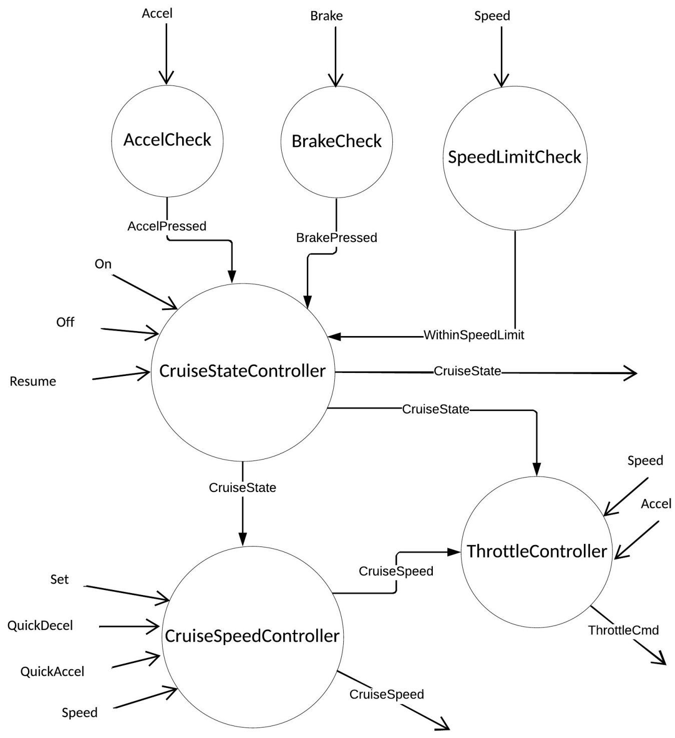


Fig. 2. Low Level Context Diagram

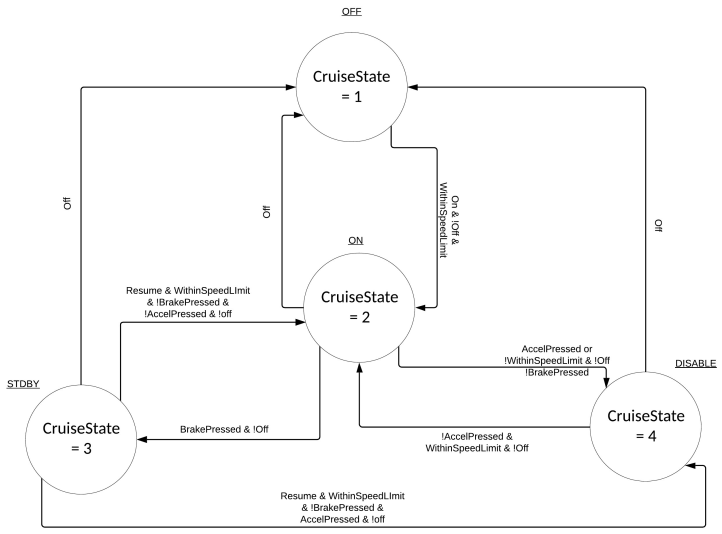

OFF
CruiseState = 1
Off
Off
On & !Off & WithinSpeedLimit
Off
ON
CruiseState = 2
Resume & WithinSpeedLImit & !BrakePressed & !AccelPressed & !off
AccelPressed or !WithinSpeedLimit & !Off !BrakePressed
DISABLE
STDBY
CruiseState = 3
BrakePressed & !Off
!AccelPressed & WithinSpeedLimit & !Off
CruiseState = 4
Resume & WithinSpeedLImit & !BrakePressed & AccelPressed & !off

Fig. 3. CruiseStateController FSM

IV. Design

Following the assignment requirements (described in section II), we have implemented an Esterel program. Our program has a top level Cruise Control module (as shown in Fig.1). The cruise control system functionality is achieved by creating an additional six modules which run in parallel. For the six additional modules, we have used Internal Signals for sharing values. The modules are explained below:

1) CruiseControl: Our top level module which runs

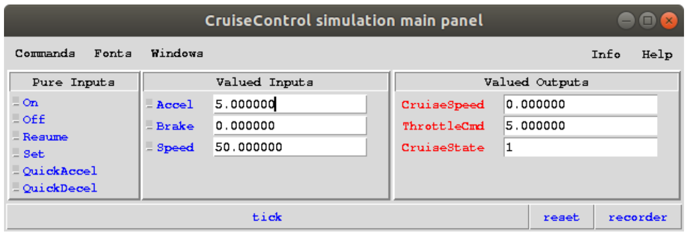
three modules in parallel, *CruiseStateController*, *CruiseSpeedController* and *ThrottleController*. The CruiseControl module receives the inputs from the simulator GUI and directs them to appropriate modules. The outputs are collected and *CruiseSpeed*, *ThrottleCmd* and *CruiseState* are produced as outputs by this module. The internal signals used between the modules running in parallel are *WithinSpeedLimit*, *AccelPressed* and *BrakePressed*.

2) SpeedLimitCheck: One of the inputs, *Speed*, is given to this module which checks if it lies within the allowed range by checking against cruise control parameters (*SpeedMin* and *SpeedMax*). If within the range, this module emits the *WithinSpeedLimit* signal.
3) CruiseStateController: This is one of the important modules in the system and is complex compared to other modules. This module outputs the state of the system based on the inputs from the simulator GUI (*On, Off and Resume*) and internal signals(*AccelPressed*, *BrakePressed* and *WithinSpeedLimit*). We have four states the system can be in, thus this module is implemented as an FSM.
4) BrakeCheck: This module takes in the *Brake* input from the simulator. It compares the Brake value against the constant *PedalsMin*, and if greater, an internal signal, *BrakePressed*, is emitted.
5) AccelCheck: This module takes in the *Accel* input from the simulator. It compares the Accel value against the constant *PedalsMin*, and if greater, an internal signal, *AccelPressed*, is emitted.
6) CruiseSpeedController: Several inputs from the simulator are directed to this modules. Inputs from the simulator include *QuickAccel*, *QuickDecel*, *Set* and *Speed* and input from an internal signal is *CruiseState*. Depending on the state, the module sets the speed when cruise system is enabled.Speed is increased or decreased in steps which is determined by a pre-set value, *SpeedInc*. If the new value is outside the range determined by *SpeedMin* or *SpeedMax* constants, the speed is set to *SpeedMin* or *SpeedMax*, respectively.
7) ThrottleController: This module uses the function, *regulateThrottle*, which is given to us as part of the assignment brief. The inputs from the simulator are *Accel* and *Speed*; internal signal is *CruiseState* and this module also takes in *CruiseSpeed*, the output of the module *CruiseSpeedController*. The regulateThrottle takes in a boolean *true* if *CruiseControl* goes from OFF to ON state, else boolean *false* if already in ON state. Using the given K_p and K_i values, the function sets the *ThrottleCmd* using a proportional and integral algorithm.

V. Conclusion

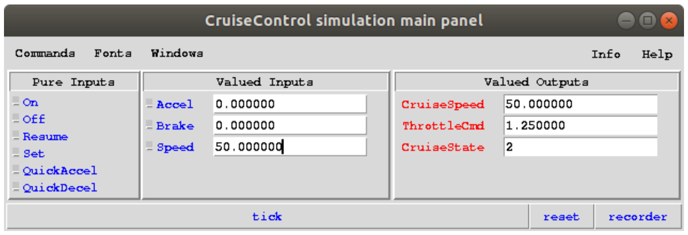
In conclusion, we have designed a cruise control system based on the requirements given in the assignment brief. We carried out several tests and the results matched the expectations. The system has been implemented using Esterel V5. Initially, the new language seemed to be a challenge, however as a group we were able to break the problem down into smaller parts and by joining dots between the smaller modules we were able to implement a fucntional cruise control system. Help from the lecture slides and TAs helped immensely with our understanding of Esterel.

VI. Appendix I



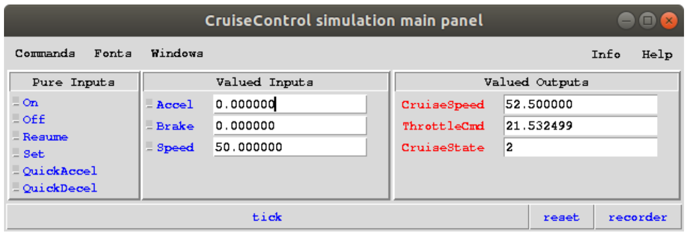
Valued Inputs: Accel 5.000000; Brake 0.000000; Speed 50.000000
Valued Outputs: CruiseSpeed 0.000000; ThrottleCmd 5.000000; CruiseState 1

Fig. 4. Cruise System in OFF state



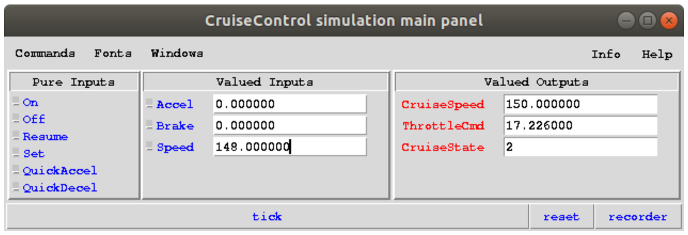
Valued Inputs: Accel 0.000000; Brake 0.000000; Speed 50.000000
Valued Outputs: CruiseSpeed 50.000000; ThrottleCmd 1.250000; CruiseState 2

Fig. 5. CruiseSpeed matches speed when in ON state



Valued Inputs: Accel 0.000000; Brake 0.000000; Speed 50.000000
Valued Outputs: CruiseSpeed 52.500000; ThrottleCmd 21.532499; CruiseState 2

Fig. 6. CruiseSpeed increases with QuickAccel input



Valued Inputs: Accel 0.000000; Brake 0.000000; Speed 148.000000
Valued Outputs: CruiseSpeed 150.000000; ThrottleCmd 17.226000; CruiseState 2

Fig. 7. The CruiseSpeed maxes out at *SpeedMax(150)* if limit is being breached