COMPSYS 723 Assignment #2

Group 29

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

1 . Introduction

Our implementation of a cruise control system as an executable esterel program; the system is intended to implement simple IO to allow the user to set up a desired speed and then enable the system to keep a car travelling at this speed automatically using a PI regulation algorithm.
The structure of this report is as follows. In section 2 we discuss the provided behavioural specification of the system and the functional requirements and present our context diagrams. In section 3 we discuss the refinement of these context diagrams into a final system control data-flow design. In section 4 we develop finite-state-machine (FSM) descriptions for the modules present in the final diagram and discuss the conversion process. In section 5 we discuss the mapping of these FSMs into esterel and section 6 sums up the design process.

2. Specification

User inputs are taken from buttons; On, Off, Resume, Set, QuickAccel, QuickDecel, these are used to enable or disable the system and setup the desired cruising speed. Sensor inputs Accel, Brake, Speed which indicate the position of the accelerator, brake and the current speed of the car. Using these inputs the system is to output: CruiseSpeed, ThrottleCmd and CruiseState; these signals indicate the current set cruising speed, the current throttlecmd that is to be applied to the accelerator to automatically reach the cruising speed and the current state of the system (including ON, OFF, STDBY, DISABLE).
The specific behavioural and corresponding functional requirments of the system are as follows:

- Buttons On/Off: enable or disable the system, these are the highest priority inputs where pressing the On button will immediately enable the cruise control system and vice-versa for Off entering respective states ON/OFF.
- Buttons QuickAccel/QuickDecel: adjust the desired cruising speed by a $\pm$2.5km/h, the crusing speed is incremented or decremented by 2.5km/h but only within the specified speed range where a value outside of this range would result in the cruising speed being set to the value limit.
- Buttons Set: set the cruising speed to the current speed of the car, the output cruising speed is set to the current car speed as long as it is within the specified speed range.
- Button Resume: used to reenable the cruise-control after the car has performed braking, after the brake pedal is pressed the user can push this button to reenable the cruise-control where the system will start regulating the speed as long as the car is within the specified speed range and the accelerator is not pressed.
- Sensors Accel, Brake, Speed: indicate the current position of the accelerator/brake pedal as well as the current speed of the car. Accel/Brake > 3.0 indicates a press of the pedal where a press of the accelerator will cause the system to enter the DISABLE state and the brake pedal the STDBY state. The speed is used as the output of the system for feedback into the PI algorithm (as well as for the set button).
- Outputs CruiseSpeed, ThrottleCmd, CruiseState: cruisespeed indicates the current set cruising speed for the car, the speed is only regulated towards this in the ON state but CruiseSpeed can be adjust in either ON, STDBY or DISABLE states. ThrottleCmd is the accelerator pedal position fed into the car, this is automatically adjusted by the system in the ON state but is otherwise set to the input accelerator position. CruiseState simply indicates the current state of the system to the user.

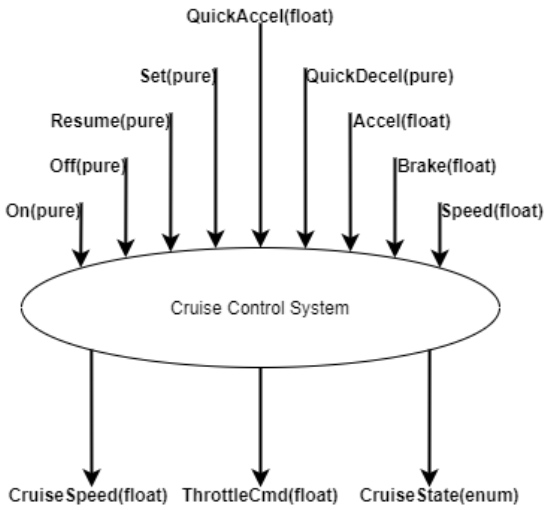

Fig. 1. Top-Level Context Diagram of the Cruise Controller

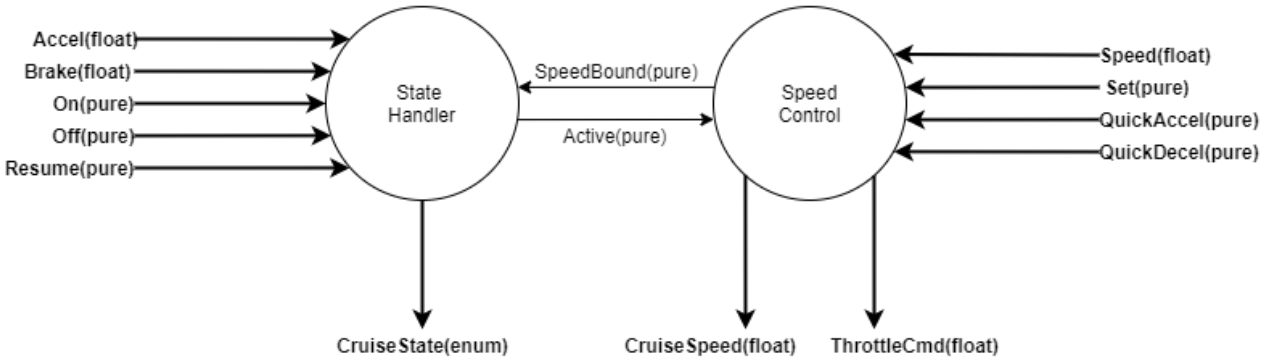

Fig. 2. First Refinement

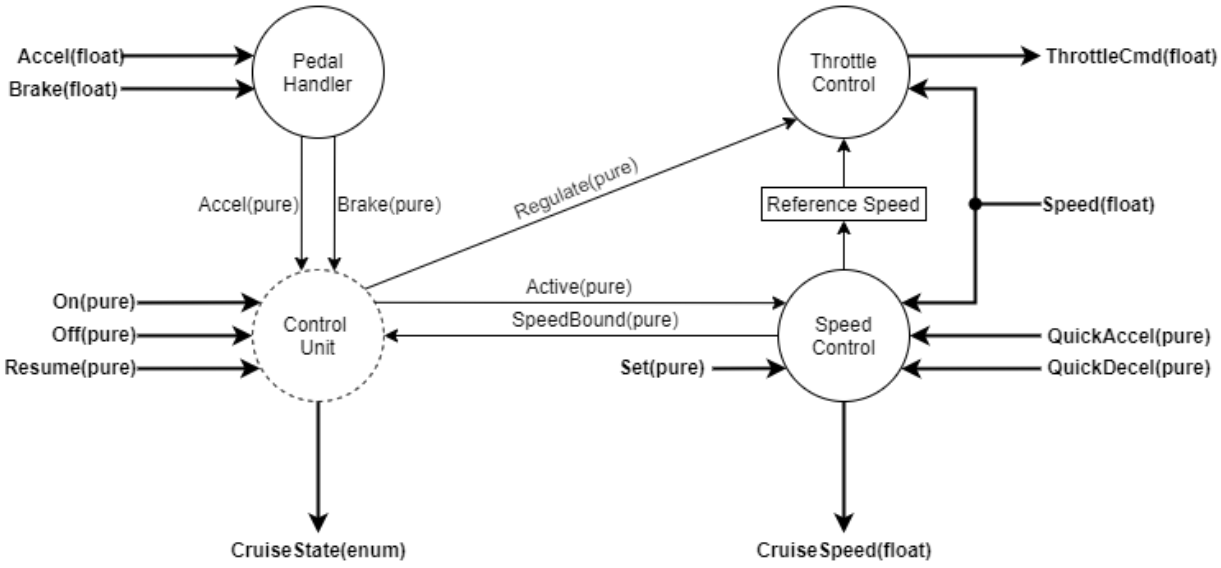

Fig. 3. Final Refinement

3. Design Refinement

The input-output interface of the system is shown in figure 1, we then refine this design by splitting up system functionality into smaller less complex modules and implement internal communcation signals. Figure 2 shows the first refinement where the overall system has been split into two modules; State Handler that handles the system state and most user IO and Speed Control which controls the cruising speed and output throttleCmd. This split simplifies the speed management that needs to be performed where the speed can be set in speed control based purely on a single signal active which indicates the speed should be regulated by the system (emitted in ON) rather than the cruise speed being set with multiple commands across several states. Additonally the speed control module can simply check for a speed outside the given range and emit SpeedBound removing the need for speed value comparisons across multiple states.
Figure 3 shows the further refinement of figure 2 into a final system context-diagram, state handler is split into two modules; pedal handler which simply converts accel and brake into pure signals indicating a push of the pedals (value>3.0) and a control unit which controls the state and outputs active which indicates the cruise speed can currently be configured, and regulate which indicates the throttle speed should be actively managed by the system. This split moves all the control to the control unit and allows it to operate on pure signals only, simplifying the design; comparisons of Brake and Accel across states not needed. Speed control is also split into two modules a smaller speed control that only handles configuring the current cruise speed and throttle control which simply implements the PI algorithm for setting the throttleCmd. This split only introduces one additional control signal regulate and one data signal reference speed but removes the throttle managing aspect from speed control where it is now only required to set the cruisespeed based on the three input signals; set, QuickAccel and QuickDecel.

4. FSM Design

The next series of figures show the design of the individual modules in the final context-diagram represented as FSMs, they all operate concurrently and communicate using internal signals, the transitions and state outputs are based on the functional requirements of the system.

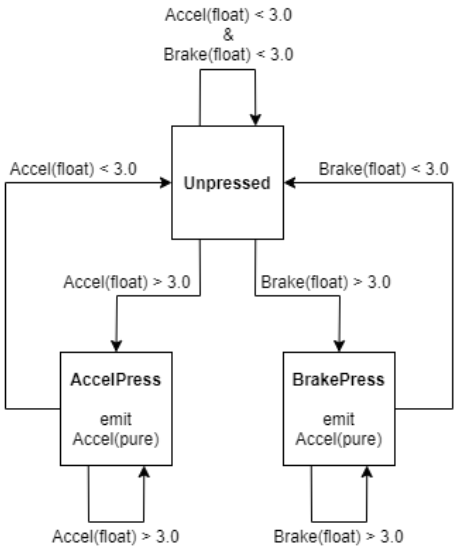

Fig. 4. Pedal Handler FSM

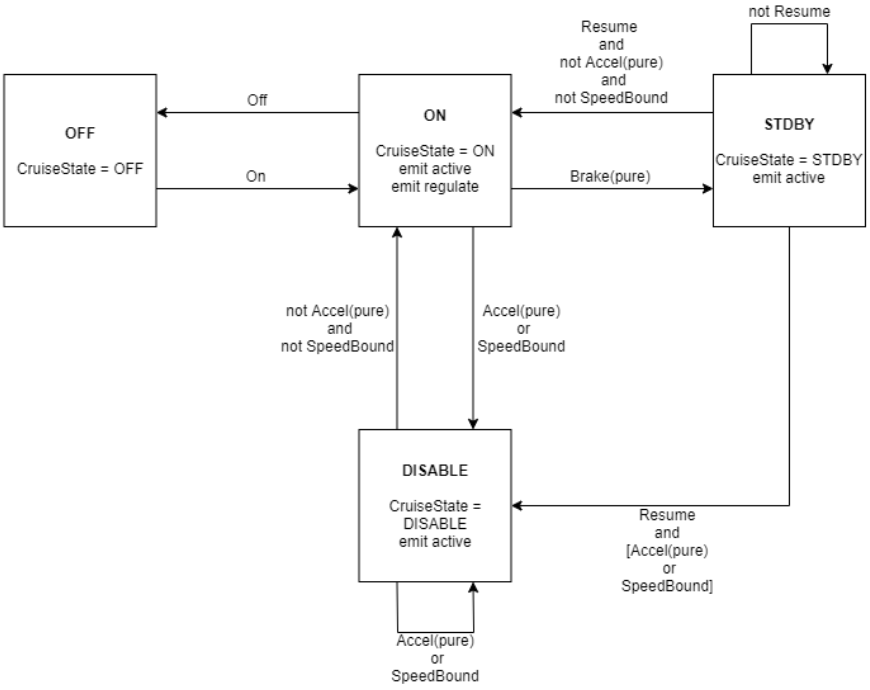

Fig. 5. Control Unit FSM

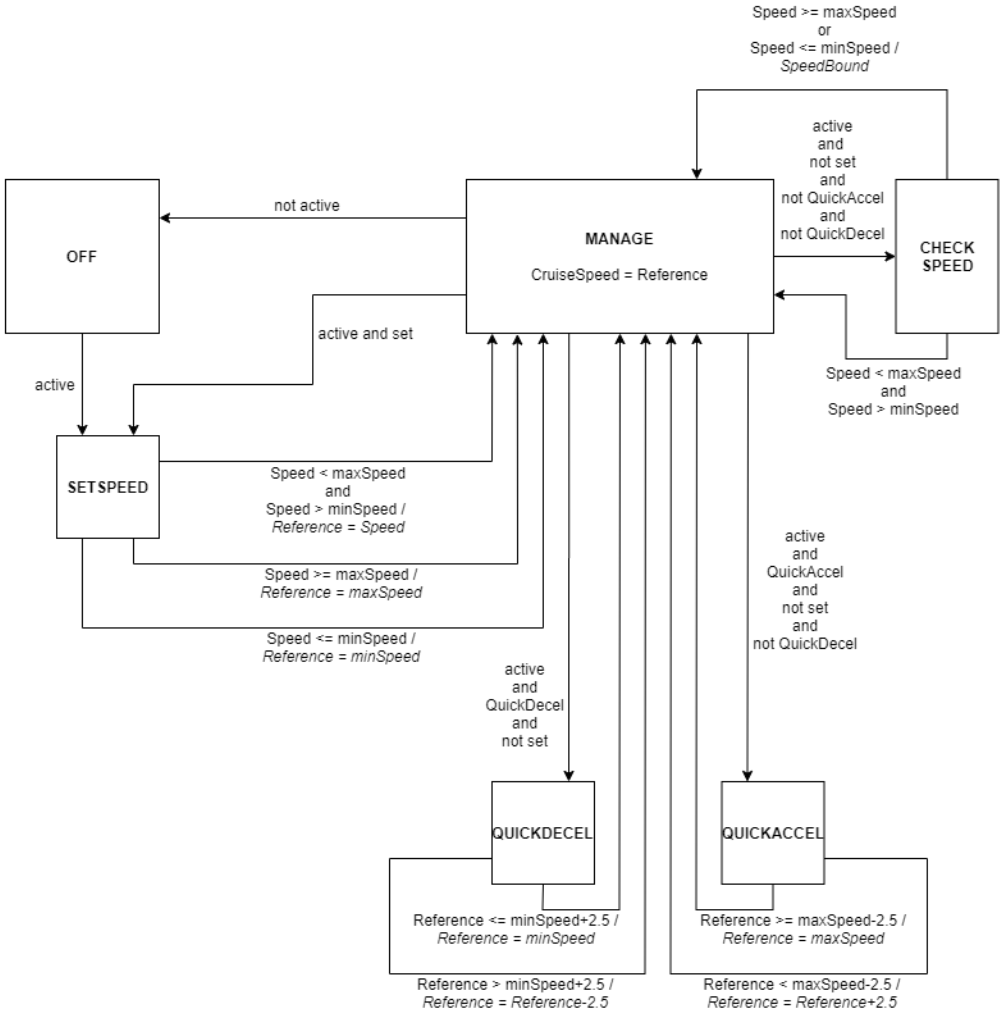

Fig. 6. Speed Control FSM

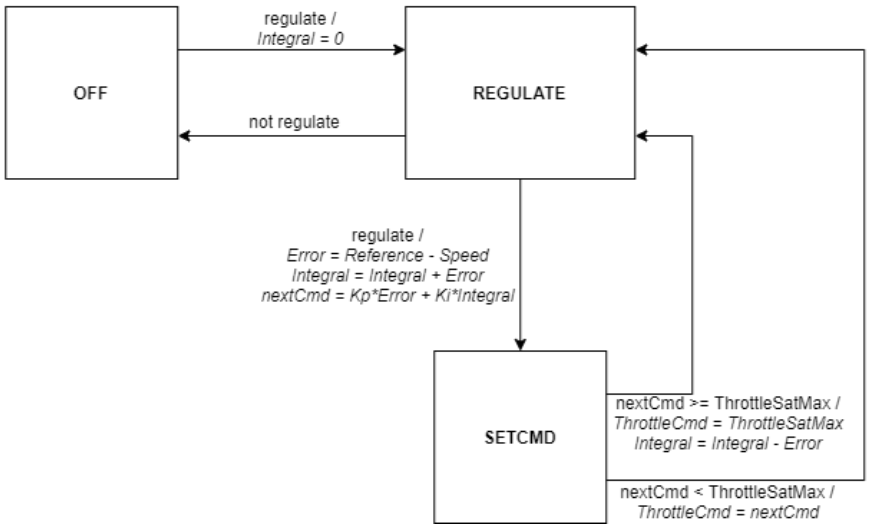

Fig. 7. Throttle Control FSM

5. Design in Esterel

The implementation of our system in esterel follows the design of the FSMs closely with a single top-level module implementing the four FSMs as concurrent processes where the each process operates in a loop to continue to provide the desired reactivity of the system. For the CruiseState output a custom type states was implemented, this required a C function for setting the state value which takes a number (from 1-4) as input and sets the state to corresponding OFF=1, ON=2, STDBY=3, DISABLE=4. For the esterel code this means setting the state of a signal is simplified to a single call to a C function with the required number.

For the control unit the design is implemented as an initial abort loop with the delay condition On, this allows the state to be sustained as OFF until the On signal is received. After this there is a primary operational abort loop that aborts on the Off signal allowing for immediate reactivity to an Off command, absence of the Off signal allows the state management to operate in a loop checking for transisition condtions for adjusting the state using a present statement. For the braking condition we chose to implement a loop once the condition for entering the STDBY state was reached this removed the need for numerous other condtions in the present statement; the loop is implemented in an abort statement that aborts on resume thus providing the desired lock into the STDBY state until the resume button is pressed.

For the speed control the design is simply a present statement that checks for the several button combinations for adjusting the cruising speed, when checking for a out of speed bounds condition for updating the reference the previous tick value of the reference is used (pre(?reference)) to avoid causality issues when then emitting reference with a new value. An additonal present statement is implemented to either emit CruiseSpeed with the value of reference or as 0.0 depending on whether the system is active, this removes the need to emit CruiseSpeed in the various button press states, moving the emission into its own present statement with simplier case conditions.

For the throttle control the design in esterel is simplified significantly by the implementation of the provided cruiseregulation.c file that implements the PI algorithm a C function, the resultant esterel code is simply a call to this function when the regulate signal is present or an output of the Accel value when it is not.

6. Conclusion

From the a top-level context-diagram containing the overall system IO we were able to produce a refined system with simpler single-task orientated modules. From this we developed the functional specification of the system as a series of finite-state-machines for the modules. Implementing this design was then a simple task of converting the FSM designs into esterel; using the synchronous tick based language with its high-level syntax reduced the coding process to a series of function calls.