Designing a Cruise Control System in Esterel

Guangyu, Zhu and Benjamin Tong
Department of Electrical and Computer Engineering
University of Auckland, Auckland, New Zealand

Abstract

We present the design of a cruise control system in Esterel. We first do the paper design outlining the specification of a cruise control system using the well-known control data-flow diagrams. We then show how to implement such a specification, where control-data decomposition is already done, in Esterel [1]. In doing this design, we demonstrate an effective way of mapping a state-machines in the specification to Esterel code efficiently. We also show the mapping of the data-flow aspects to C functions.

1. Introduction

The goal of this assignment [2] is to develop a high level functional specification for a cruise control system using a model-based approach based on some specific requirements. The purpose of the simple cruise control system is to maintain the car at a constant speed without pressing of the accelerator or brake pedals. The cruise control system has four modes, namely OFF, ON, DISABLE and STDBY. Each mode can be entered and exited based on the different values of the input signals, for example, an "On" button pressed will cause the control system to enter "ON" mode from "OFF" mode. Also, the cruise speed is regulated in various modes using different regulation mechanism. The more detailed explanation would be illustrated by first showing the specification and subsequently showing how to map those specifications in Esterel.

This assignment would demonstrate several features of Esterel language, the use of concurrency and synchronous synchronization are demonstrated using signals. It would also demonstrated the data handling ability of Esterel by separately invoking C functions. Moreover, the system consists of serval modules connecting by signals and ports, each has its own interfaces and data objects.

The detailed explanation of the specification of the cruise control system would be illustrated using a control data-flow diagrams in Section 2. In Section 3 the actual implementation on Esterel is explained with Section 3.1 presenting the use of Esterel data and interfaces and Section 3.2 presenting the top level module. Finally Section 3.3 will demonstrate some debugging methods and Section 4 we draw our conclusion.

2. Specification

The overall cruise control system is presented using the following context diagram – Figure 1. As seen from the diagram the overall system has 9 inputs and 3 outputs. The inputs consist of those buttons that the user uses to control the cruise control system. The On/Off button that enable and disable the cruise control, the Resume button to resume to the cruising state, the Set button to set the cruising speed and the QuickAccel/QucickDecel button to directly increase/decrease cruising speed. The inputs also consist of 3 sensors; Accel, Brake and Speed that monitors the position of accelerator and brake and the value of current speed.

The outputs from the control system are the CruiseSpeed to indicate the current cruising speed, the ThrottleCmd that controls the flow of power to the engine to regulate the speed and CruiseState that indicates which mode is the control system currently in.

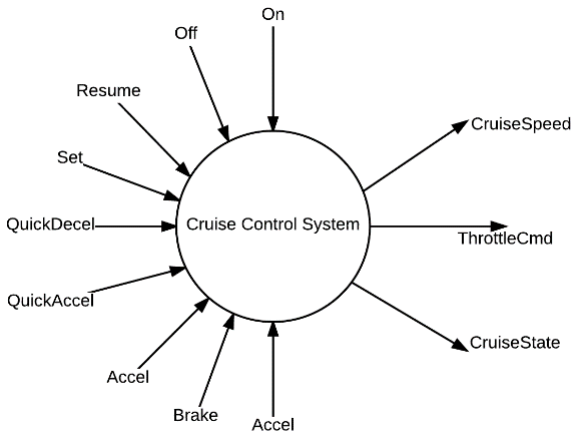


Fig. 1. Context Diagram of Cruise Control System

The system can be divided into three smaller modules that each has different functionalities. Those modules are shown in Figure 2.

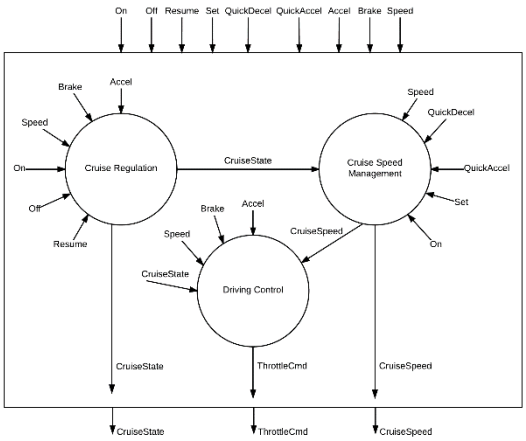


Fig. 2. System breakdown of the overall system (Please Refer to Appendix for the enlarged version)

The main control unit is the Cruise Regulation module which consists of a FSM in Figure 3 to determine which mode the system is currently in. The current mode is sent out as a control signal to the Cruise Speed Management module where it determines the cruising speed of the car and the car regulates the throttle using the Driving Control module.

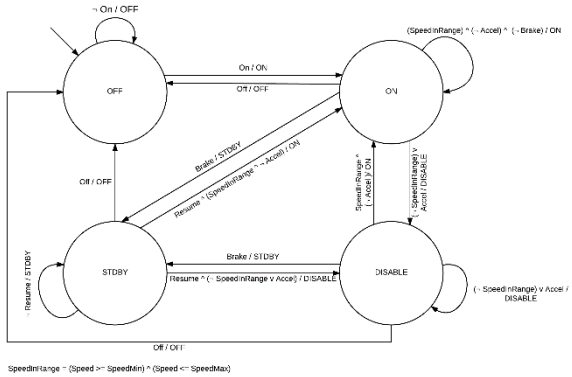


Fig. 3. The Cruise Regulation Module as an FSM (Please Refer to Appendix for the enlarged version)

3. Design in Esterel

The Esterel design of the cruise control system follows the specification described in Section 2. It contains 3 modules, namely the Cruise Regulation module, the Cruise Speed Management module and the Driving Control module. The three modules are contained in a top-level module called Cruise Control System.

- Cruise Regulation module: This module executes an FSM that is driven by various user input signals, On, Off, Resume and also based on sensor values Accel, Brake and Speed. This is a simple FSM that contains four states, the transitions between states are implemented in Esterel using state variables and traps. The current state is outputted to be used by the other two modules.
- Cruise Speed Management module: This module receives user inputs (On/Set/QuickAccel/QuickDecel) and also current speed from speed sensor to set the cruising speed of the car. The cruise speed is maintained between a minimum speed (30km/h) and a maximum speed (150km/h) by calling corresponding C functions declared in a separate C file. The cruising speed is outputted to be used by the Driving Control module.
- Driving Control module: The Driving Control module takes in the input from the above two modules then it determines whether the car speed should be automatically regulated or determined by the accelerator pedal. The output ThrottleCmd is computed using C functions declared in a separate C file.

3.1. The Use of Data and Interfaces

A module's interface consists of data type declarations, constants, functions or procedures, input and output interface signals. In our implementation, we have the following data and interfaces for each module:

```
module cruiseRegulation:

% Function declarations
function speedInRange(float): boolean;
function pedalOn(float): boolean;
% Inputs
input On, Off, Resume;
input Accel : float, Brake : float, Speed : float;

% Defining the State Type which is defined in c.
% It is the enumeration of the CruiseState, which is in either offState (OFF), onState(ON),
% disableState(DISABLE) or stdbyState(STDBY) state.
type StateType;

output CruiseState : StateType;
constant offState : StateType, onState : StateType, disableState : StateType, stdbyState :
StateType;
```

Fig. 4. Cruise Regulation Module Interface

```
% Module Cruise Speed Management: It will decide and set the desired cruise speed of the car.
module cruiseSpeedManagement:

% Inputs
input On, Set, QuickAccel, QuickDecel;
input Speed : float;

type StateType;

% Outputs
output CruiseState : StateType, CruiseSpeed : float;
constant offState : StateType, onState : StateType, disableState : StateType, stdbyState :
StateType;

% Function Declarations
function setSpeed(float) : float;
function cruiseSpeedAdjust(float, integer) : float;
```

Fig. 5. Cruise Speed Management Module Interface

```
% Module Driving Control: It will decide what level for the throttle (ThrottleCmd) should be set
% to reach and maintain the cruise speed in ON, and the corresponding level set in OFF state.
module drivingControl:
% Inputs
input Accel : float, Brake : float, Speed : float, CruiseSpeed : float;

type StateType;
% Outputs
output CruiseState : StateType, ThrottleCmd : float;
constant offState : StateType, onState : StateType, disableState : StateType, stdbyState :
StateType;

% Function Declarations
function regulateThrottle(integer, float, float): float;
function speedAdjust(float, integer): float;
function pedalOn(float): boolean;
```

Fig. 6. Driving Control Module Interface

```
% Module Cruise Control System: This is the top level module that runs three modules in parallel
and connect the correspondng inputs and outputs for the whole system.
module cruiseControlSystem:

% Inputs
input On, Off, Resume, Set, QuickDecel, QuickAccel;
input Accel := 0.0f : float, Brake := 0.0f : float, Speed := 0.0f : float;

type StateType;
% Outputs
output CruiseState : StateType, CruiseSpeed : float, ThrottleCmd : float;
constant offState : StateType, onState : StateType, disableState : StateType, stdbyState :
StateType;
```

Fig. 7. The Top Level Module Interface

3.2. The Cruise Control Module

This is the top level Esterel module that runs the Cruise Regulation module, the Cruise Speed Management module and the Driving Control module in parallel. The interconnection between different modules are described in Figure 2. So, all of the signals that connect to the environment are declared at the interface of the top level module. The ports of each module are connected to the top module using signal assignment. Because the signals are inside different modules, they are allowed to have the same name. For example, inside Cruise Regulation, the "signal On/On" statement would map the signal "On" declared at the interface of top level module to the "On" signal declared at the Cruise Regulation module to supply input to Cruise Regulation module. All the other signals are mapped in this way.

3.3. Causality and Data Handling

Finally, when designing using Esterel, the most important thing to be kept in mind is the causality issues. Causality issues arise when we want to assign to a retrieve the value from a signal and emit its value in the same tick. The issue can be fixed by adding "pre" to the signal to access its pre state or value. For example, when we want to see the state transition we use pre(?CruiseState) to get its previous value. By doing this we can avoid the causality issue which is common in the Esterel language.

Note that Esterel V5 does not natively support enumerations, enumerations must be declared as user defined types. Below is a snippet of the declaration of custom type StateType found in cruiseControlSystem.h:

```
typedef enum {
    OFF = 1, ON, STDBY, DISABLE
} StateType;

void _StateType(StateType *lhs, StateType rhs);
int _eq_StateType(StateType lhs, StateType rhs);
char* _StateType_to_text(StateType state);
void _text_to_StateType(StateType *state, char *text);
int _check_StateType(char *text);
```

Fig. 8. Declaration of the User Defined Type - Enumeration

In the enumeration we declared, we mark "ON" mode as integer 1, "OFF" mode as integer 2, "STDBY" mode as integer 3 and "DISABLE" mode as integer 4 instead of starting from 0 as the normal enumerations do. For each user defined type declared, five functions similar to the above snippet are expected by the Esterel compiler to handle the type. These are functions to handle assignment (the "=" operator), comparison, converting to and from a string, and type sanity check, in respective order declared above. [3]

4. Conclusions

The assignment provides an opportunity to experience the design of embedded software using a model-based approach. The system is implemented using the language Esterel V5 which features concurrency, synchronization and data handling through C. After completing the assignment, we have gained a hands-on experience with the language and we have also gained deeper knowledge in respect to those features provided by Esterel V5 as well as some understanding on embedded system design.

Acknowledgements

We would like to acknowledge the help from Dr. Avinash Malik, our lecturer, from providing the materials and concepts for the course, we also want to thank the two TAs, Keith Ren and Nathan Allen, for their help throughout the assignment.
.

5. References

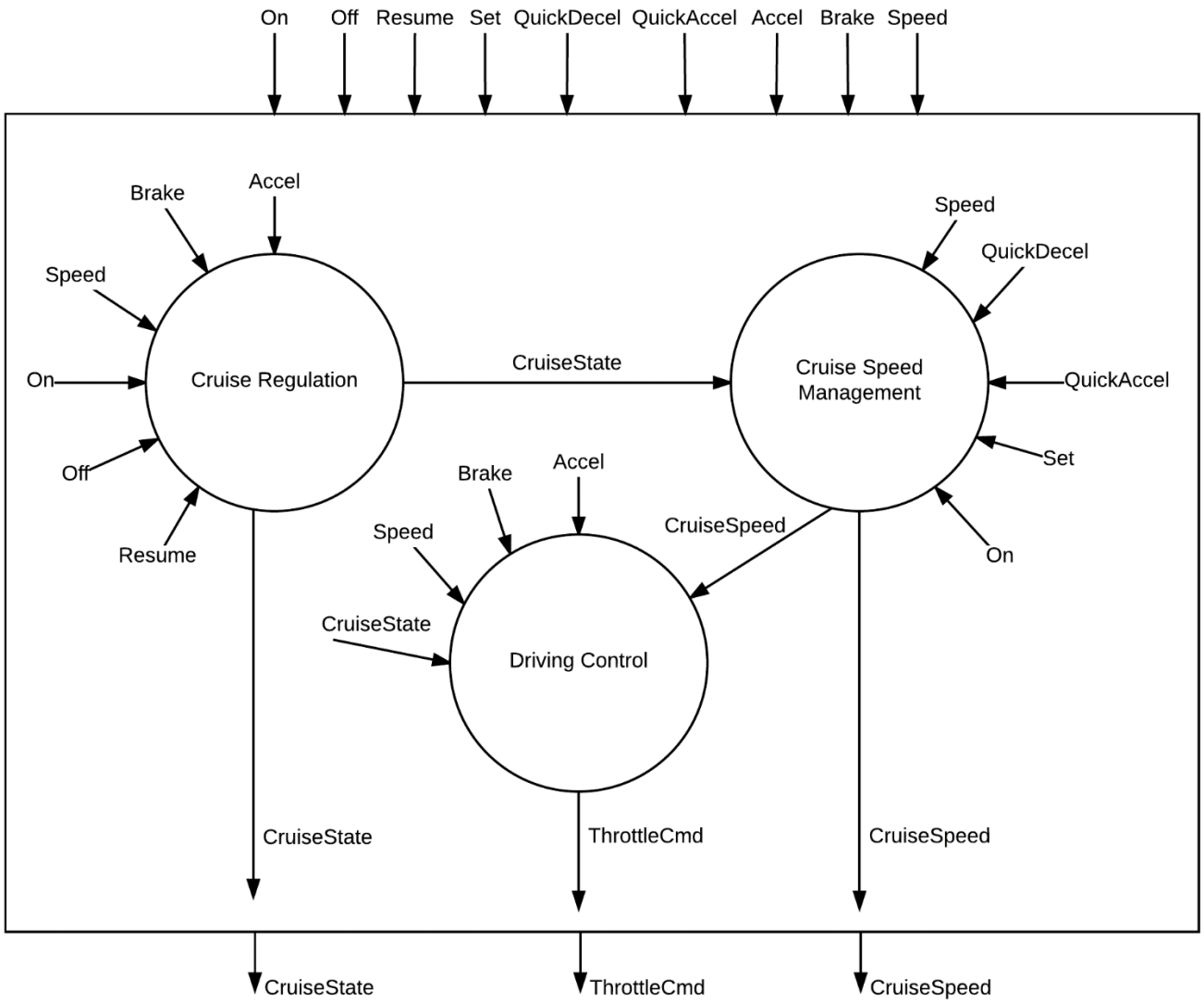
[1] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages12 years later. *Proceedings of the IEEE*, 91(1):64–83, Jan 2003.

[2] A. Malik. Compsys723 Assignment Brief. retrieved from Canvas.

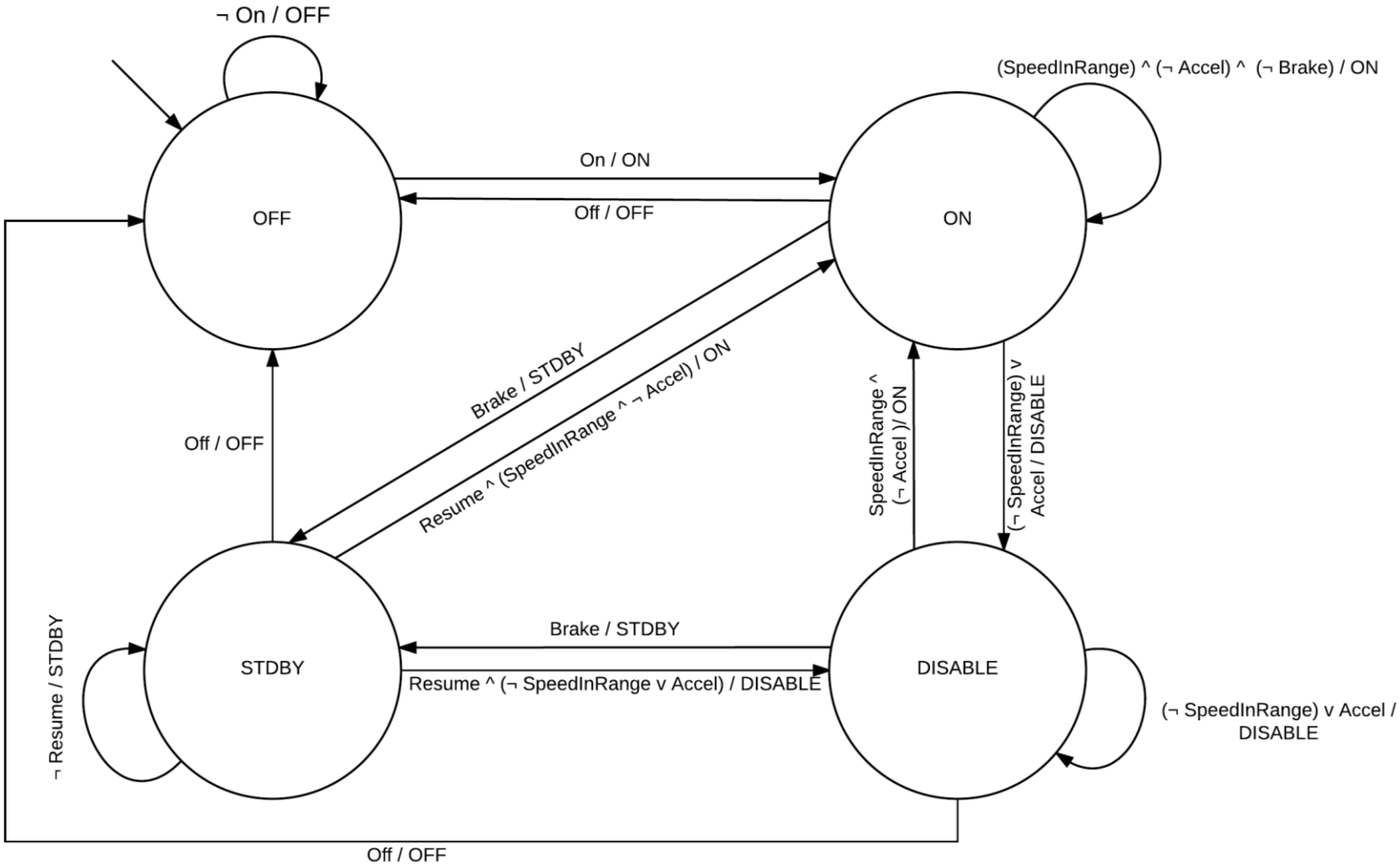
[3] P. S. Roop. Designing A Lift Controller in Esterel. retrieved from Canvas.

Appendix

1) System breakdown of the overall system



2) The Cruise Regulation Module as an FSM



SpeedInRange = (Speed >= SpeedMin) ^ (Speed <= SpeedMax)