# COMPSYS723

Due by Sunday, 9/Jun/2019

Assignment 2 (15% of the final marks)

Lecturer: Dr. Avinash Malik (avinash.malik@auckland.ac.nz)
TA: Ke (Keith) Ren (kren972@aucklanduni.ac.nz)
TA: Nathan Allen (nall426@aucklanduni.ac.nz)

## Motivation and Objective

The goal of this assignment is to provide students with hands-on experience in the design of embedded software using a model-based approach. Based on a set of requirements, students will be required to develop a functional specification for a cruise control system. This functional specification will then be implemented using the synchronous programming language *Esterel*. The result will be an executable reactive program that fulfils the given requirements for the cruise control system.

**Learning outcomes** – After completing this project, students will gain deep insight on the following:

- Development of functional specifications to final implementation of an embedded system.
- Use of hierarchy and concurrency for refining complex embedded systems.
- Experience in implementing high-level functional models of embedded systems using the Esterel programming language.

**Required files** – For completing the assignment, you need the following files:

1) *ReportGuidelines.docx:* file contains guidelines for writing a technical report.
2) *projectReviewForm.docx:* This form must be **filled as a group** to evaluate the overall assignment.
3) *peerReviewForm.docx:* This form must be completed to evaluate the performance of your team member. Fill **one form for each member in your team** (excluding yourself).
4) *cruiseregulation.c:* file contains some code that students must use during the implementation of the assignment.
5) *vectors.in and vectors.out:* Check the correctness of your assignment by using the input vector (*vectors.in*), and by checking against the expected output (*vectors.out)*.
    - *The inputs to the simulator GUI will be the inputs from the environment to the cruise controller.*
    - *Due to the difference between implementations, the outputs do not need to match the exact number of ticks (i.e, the ThrottleCmd can take 1 or 2 ticks to change value depending on your implementation). However, output should be settle to the correct value after a reasonable amount of ticks (i.e., within 3 ticks).*

## Design Requirements

You are required to design a simplified *cruise control system* for a car. The purpose of the cruise control system is to maintain the car at a constant speed without pressing the accelerator or brake pedals. The relevant **cruise control parameters** that are referred during this document are given below:

1) *SpeedMin:* 30.0 km/h
2) *SpeedMax:* 150.0 km/h
3) *SpeedInc:* 2.5 km/h

4) *Kp:* 8.113
5) *Ki:* 0.5
6) *ThrottleSatMax:* 45.0 percent
7) *PedalsMin:* 3.0 percent

The cruise control system has a number of **interface requirements**. It should have the following *inputs*:

1) *On (pure):* Enable the cruise control.
2) *Off (pure):* Disable the cruise control.
3) *Resume (pure):* Resume 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.

Also, the cruise control system should deliver the following *outputs*:

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

The cruise control system has the following **behavioural requirements**:

1) When the car first starts, the cruise control shall be off. The output *CruiseState* should be set to *OFF*. In this state, pressing *Off, Resume, Set, QuickAccel* and *QuickDecel* has no effect.
2) The cruise control shall go on when the on button (*On*) is pressed. The output *CruiseState* should be set to *ON*.
3) The cruise control shall go off whenever the off button (*Off*) is pressed. The output *CruiseState* should be set to *OFF*.
4) When the cruise control is on, if the car speed (*Speed*) is within the speed limit ([*SpeedMin, SpeedMax*]) and the accelerator pedal (*Accel*) is not pressed, the cruise control shall regulate the car speed and the output *CruiseState* shall be set to *ON*.
5) The cruise control shall be disabled when the accelerator pedal is pressed, or the car speed is outside the speed limit. The output *CruiseState* shall be set to *DISABLE*.
6) The cruise control shall go from disable state to the on state when both the accelerator pedal is not pressed, and the car speed is within the speed limit. The previous set cruise speed shall be reused.
7) The cruise control shall be immediately interrupted and should enter the standby state, when the brake (*Brake*) is pressed. The output *CruiseState* shall be set to *STDBY*.
8) From the standby state, the cruise control shall resume either to the on or disable states, depending on the accelerator pedal and the speed value when the *Resume* button is pressed. The last set cruise speed shall be reused.
9) Assume the accelerator pedal and the brake pedal will never be pressed together.

The **car driving control** shall have the following properties:

1) When the cruise control is off, the car speed shall be driven by the accelerator pedal.
2) When the cruise control is on, the car speed shall be automatically regulated.
3) The regulation shall be done using a proportional and integral algorithm, with *Kp* and *Ki* factors.
4) The regulation shall be protected against the overshoot of its integral part: the integral action shall be reset when the cruise control is going on, and frozen when the throttle output is saturated.
5) The throttle command (*ThrottleCmd*) shall be saturated at *ThrottleSatMax* when automatically regulating, in order to limit the car acceleration for passenger comfort.

The **cruise speed management** shall behave in the following manner:

1) The cruising speed (*CruiseSpeed*) shall be managed only when the cruise control is enabled, meaning when it is in the *ON, STBDY,* or *DISABLE* states.
2) The cruise speed shall be set to the current speed when the *On* button is pressed initially, and whenever the *Set* button is pressed thereafter.
3) The cruise speed shall be increased by *SpeedInc* km/h when the *QuickAccel* button is pressed, provided that this new value of the cruise speed is still lower than the maximal speed, *SpeedMax* km/h.
4) The cruise speed shall be decreased by *SpeedInc* km/h when the *QuickDecel* button is pressed, provided that this new value of the cruise speed is still higher than the minimal speed, *SpeedMin* km/h.
5) The cruise speed shall be maintained between *SpeedMin* and *SpeedMax* km/h. This means that whenever an attempt is made to set the cruise speed (either through the *On, Set, QuickAccel,* or *QuickDecel* buttons) to a value less than *SpeedMin* or greater than *SpeedMax*, the cruise speed shall be limited to either *SpeedMin* or *SpeedMax*, respectively.

The **detection of the pedals' pressed** shall have the following behaviour:

1) The accelerator pedal shall be detected as pressed when its value is above *PedalsMin*.
2) The brake pedal shall be detected as pressed when its value is above *PedalsMin*.

The aim of this assignment is to implement the given cruise control specifications using Esterel V5. Students will need to *develop the system context diagram, perform data-flow / control-flow decomposition, and describe the cruise control behaviour using a finite state machine*. The finite state machine should exhibit *concurrency*. Once the formal specification has been developed, students are to map that model to Esterel V5 using the hierarchy, concurrency, and communication primitives within the language. The design of a very similar system, the lift controller from Vahid's book, is taught in the lectures and a design report written by the lecturer is available to students for reference. You may emulate the style of this design in this project.

### How the assignment is to be done

This assignment is to be done in teams of two students.

**Expected time commitment**: Students are expected to spend no more than 20 hours each for the implementation of the assignment. The report writing and resource consolidation should take not more than 5 hours in total for each student. Times spent on the assignment need to be documented in the final report.

## Deliverables

**Only one member of the group** must submit the single zip file (named group_X.zip), through Canvas. The zip file must contain the following:

- A pdf version of a joint report, named **report_group_X.pdf**, which describes the main elements of the solution. In particular, the report should contain the formal specification diagrams that the students have developed for their design. The report must not be longer than 5 A4 pages with 10 point Times New Roman font. The report should be based on the guidelines provided in *reportGuidelines.docx*.

- All the program files, with a README.txt

**In addition, every student must** fill the provided confidential peer review form, *peerReviewForm.docx* and send it to Nathan Allen a nall426@aucklanduni.ac.nz. In subject please state "COMPSYS723 PEER review".

### Academic integrity

*The University of Auckland will not tolerate cheating, or assisting others to cheat, and views cheating in coursework as a serious offence. The work that a student submits for grading must be the student's own work, reflecting his or her learning. Where work from other sources is used, it must be properly acknowledged and referenced. This requirement also applies to sources on the world-wide web. A student's assessed work may be reviewed against electronic source material using computerised detection mechanisms. Upon reasonable request, students may be required to provide an electronic version of their work for computerised review.*