KTH Information and
Communication Technology

IL2212 EMBEDDED SOFTWARE

Laboratory 2 : Cruise Control

February 5, 2013

1 Objectives

The objective of this laboratory is to implement a cruise control system for a car using the real-time operating system MicroC/OS-II.

Read the entire laboratory manual in detail before you start with the preparation tasks. Complete the preparation tasks before your lab session in order to be allowed to start the laboratory exercises.

It is very important that students are well-prepared for the labs, since both lab rooms and assistants are expensive and limited resources, which shall be used efficiently. The laboratory will be conducted by groups of two students. However, each student has to understand the developed source code and the preparation tasks. Course assistants will check, if students are well-prepared. Students, who are not well-prepared for the laboratory, have no right to get help from the assistants during the lab sessions!

Whenever you have completed a task of the laboratory, mark the task as completed by putting a cross into the corresponding circle.

Note: All program code shall be well-structured and well-documented. The language used for documentation is English.

2 Documenting Results

While you perform the laboratory tasks, you shall compile a document which contains your findings. In several places in this lab manual, you are asked to write some result down. To identify these tasks more clearly, you will find the symbol ✎ in these places. Please put your results into one document which you shall be able to show when you demonstrate the ☜

lab results to an assistant. You do not need to hand in the document and it does not need to be in a digital format. It is sufficient with hand-written sheets of paper, we just want to see everything in one place to simplify examination.

3 Description of the Cruise Control

If activated, the cruise control system shall maintain the speed of the car at a constant value that has been set by the driver.

The system has the following inputs:

- **Engine (ON/OFF).** The engine is turned on, in case the signal `ENGINE` is active. The engine can only be turned off, if the speed of the car is 0 m/s.
- **Cruise Control (ON/OFF).** The cruise control is turned on, if the signal `CRUISE_CONTROL` is activated and if the car is in top gear (`TOP_GEAR` is active) and if the velocity is at least $20\frac{m}{s}$ and the signals `GAS_PEDAL` and `BRAKE_PEDAL` are inactive.
- **Gas Pedal (ON/OFF).** The car shall accelerate, if the signal `GAS_PEDAL` is active. The cruise control shall be deactivated, if `GAS_PEDAL` is active.
- **Brake (ON/OFF).** The car shall brake, when the signal `BRAKE` is active. Also the cruise control shall be deactivated, if the signal `BRAKE` is activated.
- **Gear (HIGH/LOW).** The car has two different gear positions (high, low) indicated by the signal `TOP_GEAR`. If `TOP_GEAR` is active the the gear position is high, otherwise low. The cruise control is deactivated, when the gear position is moved to low.

The inputs are connected to the following IO-units on the DE2-board:

Signal	Pin	LED
`ENGINE`	SW0	LEDR0
`TOP_GEAR`	SW1	LEDR1
`CRUISE_CONTROL`	KEY1	LEDG2
`BRAKE_PEDAL`	KEY2	LEDG4
`GAS_PEDAL`	KEY3	LEDG6

Table 1: Connection of Signals to IO-Pins on the DE2-Board

The system consists of four periodic tasks as illustrated in Figure 1.

The car will travel on an oval track of the length 2400m, which has the profile as illustrated in Figure 2.

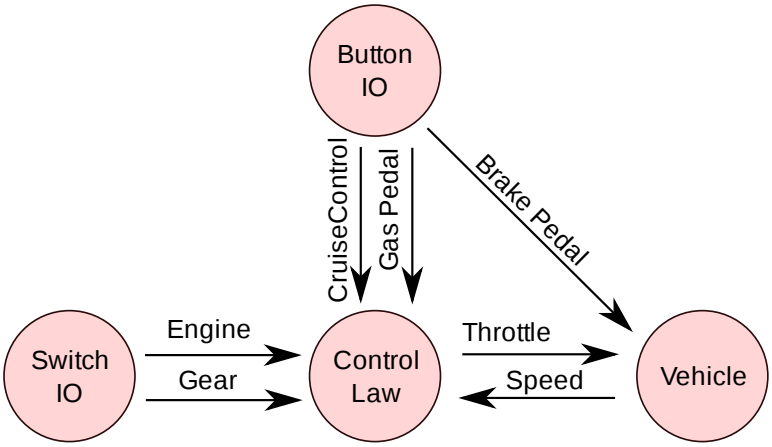

Figure 1: Tasks in the Cruise Control System

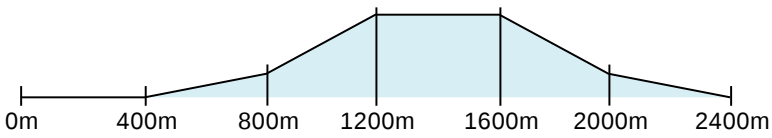

Figure 2: Profile of the oval track, which has a length of 2400m

4 Preparation Tasks

Before lab session 2, Tasks 4.1 - 4.3 and an executable preliminary version of Task 4.4 (not all details need to be correctly implemented yet) must be completed.

Until lab session 3, all preparation tasks shall be completed.

4.1 Understand the Initial Program

Available from the course web page is an executable skeleton program `cruise_skeleton.c`, which implements the vehicle task and an initial skeleton for the control task. In the skeleton program, the control task uses a constant throttle of 40. The task `VehicleTask` implements the behavior of the car and its functionality shall not be changed during this laboratory. The only permitted code modification in `VehicleTask` is the replacement of the timer (see Task 4.2).

Study the initial program carefully in order to have a clear understanding of the program. Execute it on the DE2-board. ◯ 4.1 completed

4.2 Use Soft Timers to Implement Periodic Tasks

The skeleton program uses the statement `OSTimeDlyHMSM` to implement periodic tasks, which will not give an exact period. Use instead soft timers for this purpose and connect them with semaphores to your tasks. Use the same period for the soft timers as in the original skeleton program.

See: `OSTmrCreate` in μC/OS-II Reference Manual, Chapter 16.

NOTE: Do not forget to integrate the C-code for the soft timer by selecting the option 'Enable code for Timers' below 'RTOS-Options' in 'System Library Properties'.

◯ 4.2 completed

4.3 I/O-Tasks

Create the tasks `ButtonIO` and `SwitchIO`, which read the buttons and switches on the DE2-board periodically. The task `SwitchIO` creates the signals `ENGINE` and `TOP_GEAR`, while the task `ButtonIO` creates the signals `CRUISE_CONTROL`, `GAS_PEDAL` and `BRAKE_PEDAL`. Use the red LEDs to indicate that a switch is active and the green LEDs to indicate that a button is active, as specified in Table 1.

◯ 4.3 completed

4.4 Control Law

Implement the control law in the `ControlTask` so that it fulfills the specification from Section 3. Note that the braking functionality is implemented inside the `VehicleTask`, whereas the `ControlTask` sets the throttle.

The control law shall react according to the state of the buttons and switches. When the cruise control is activated, the current velocity shall be maintained with a maximum deviation of $\pm 2\frac{m}{s}$ for velocities of at least $25\frac{m}{s}$. Use the green LED LEDG0 to indicate that the cruise control is active.

Implement the dummy functions `show_position` and `show_target_velocity`. `show_position` shall indicate the current position of the vehicle on the track with the six leftmost red LEDs as specified in Table 2.

LED	Position
LEDR17	[0m, 400m)
LEDR16	[400m, 800m)
LEDR15	[800m, 1200m)
LEDR14	[1200m, 1600m)
LEDR13	[1600m, 2000m)
LEDR12	[2000m, 2400m]

Table 2: LED assignment to show the position of the vehicle

`show_target_velocity` shall display the target velocity, which the cruise control is trying to maintain, on the seven segement display (HEX5 and HEX4). The display shall be reset to 0 when the cruise control is deactivated.

◯ 4.4 completed

4.5 Watchdog

To allow for the detection of an overloaded system, add a watchdog task and an overload detection task to the system. The overload detection task shall report to the watchdog with an 'OK' signal. In case the watchdog task does not receive the signal during a specified interval, it shall issue an overload warning message. Choose a reasonable interval for your watchdog. Which interval did you choose and why? ✎ ☜

Add another task to impose an extra load onto the system. It shall be possible to dynamically adjust the amount of processing time that the task utilizes. To set the task utilization, the switches SW4 to SW9 shall be used. The switch pattern shall be interpreted as a binary number (with SW4 as the lowest bit), i.e., 2^6 values can be represented. The utilization shall be adjustable in 2% steps. Everything higher than 100% (i.e., all numbers above 50) shall be considered as 100% utilization. Note: *The utilization considered here is the ratio between the task's execution time and its period. Not the total utilization of the entire system! Take some function, measure the execution time, create a task with a period which corresponds to the measured execution time, i.e., one execution of the function takes about 2% of the period, two executions take 4% and so on. Put a loop around the function, such that the function is executed as many times as it takes to reach the utilization which is specified by the buttons.*

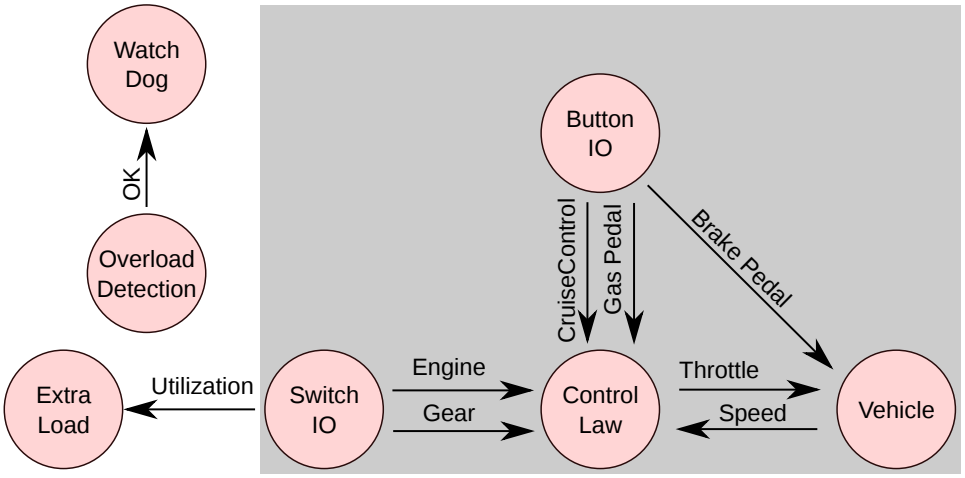

Figure 3: Cruise Control System with Overload Detection

◯ 4.5 completed

Figure 3 illustrates the resulting system with overload detection. The grey box contains the original system. Note that there are *no connections* from the original system to the overload detection system, i.e., watchdog task and overload detection task.

Pay attention to choose the priorities for the tasks so that any system overload is actually detected and that the extra load affects the watchdog system. The overload detection task shall run whenever there is slack time in the system, instead of being activated periodically.

Test at which utilization a system overload occurs under different circumstances (e.g. cruise control activated, car standing still, pushing gas pedal, ...) and write down your results for at least three different scenarios.

☞ ✎

4.6 Optimization

Check the size of your program, using the command `nios2-elf-size` and write down the numbers.

☞ ✎

In the skeleton program, the stack size has been set to 2048. Find out how much stack each task uses at most and reduce the stack sizes accordingly. Hint: Use the μC/OS-II function `OSTaskStkChk()`. How much stack does each task use?

☞ ✎

Optimize the system for code size (i.e. set the optimization level to -Os) and determine to what extent the size has decreased.

☞ ✎

Test again, at which utilization a system overload occurs. Did the optimization for size decrease the performance of the system?

☞ ✎

◯ 4.6 completed

5 Laboratory Tasks

5.1 Demonstration of Preparation Tasks

Demonstrate the programs that you have developed in the preparation tasks for the laboratory staff. Be prepared to explain your programs in detail. Have your documentation of results (see Section 2) at hand.

5.2 Surprise Task

You will be given a 'surprise task', which you need to conduct during the laboratory session. In order to be able to solve the surprise task you need to have a very good understanding of the preparation tasks!

6 Examination

In order to pass the laboratory the student must

- have completed the corresponding preparation tasks of Section 4 before the lab sessions
- have demonstrated the preparation tasks, including documentation, for the laboratory staff and completed the surprise task (Section 5).