Prof. Dr. Christian Steger
Prof. Dr. Reinhard Wilhelm

Sebastian Altmeyer, M.Sc.
Michael Gerke, B.Sc.
Dipl.-Inf. Hans-Jörg Peter

Embedded Systems 2010/2011 – Assignment Sheet 4

Due: Tuesday, 23rd November 2010, *before* the lecture (i.e., 10:10)
Please indicate your **name**, **matr. number**, **email address**, and which **tutorial** you are planning to attend on your submission. We encourage you to collaborate in **groups** of up to **three** students. Only one submission per group is necessary. However, in the tutorials every group member must be capable to present each solution.

Exercise 1: Modeling with SDL (30 pts.)

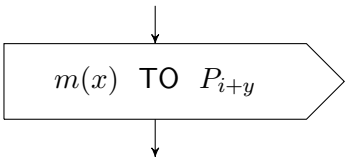
An embedded device comprises n sensors and n independent processing units, where $n = 2^k$ and $k \in \mathbb{N}$. Each processing unit P_i is connected to its designated sensor whose value is given as $S_i \in \mathbb{N}$, $1 \leq i \leq n$. This means that only P_i can read S_i.

Let $\circ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a commutative and associative operation such that $a \circ b$ can be computed in time $O(1)$. Your task is to provide a distributed algorithm in SDL such that the result of

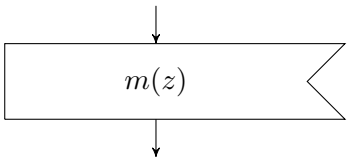
$$S_1 \circ S_2 \circ \ldots \circ S_n$$

is sent after $O(\log n)$ time to a designated result process P_{2n}. Here, "distributed" means that you have to provide an SDL process implementation for each P_i, $1 \leq i \leq n$. P_{2n} does not need to be implemented. Each P_i can have constantly (i.e., independent of n) many local variables, but no arrays may be used.

Recall that each process can only send their local computed values via FIFO queues to other processes. You can assume that, before your algorithm is executed, the current value of S_i is the top element in the input FIFO queue of P_i, for each $1 \leq i \leq n$. For receiving and sending variable values to other processes, you can use the syntax shown in Figure 1.



(a) Sending the value of a local variable x as the parameter of a signal m to a process P_{i+y}.



(b) Receiving a signal m and storing its parameter into a local variable z.

Figure 1: SDL example syntax for sending/receiving values of local variables to/from other processes.

Exercise 2: Timers in SDL (25 pts.)

Recall the timed mutual exclusion protocol from the first assignment sheet (Exercise 3). The StateChart model for two processes is shown in Figure 2.

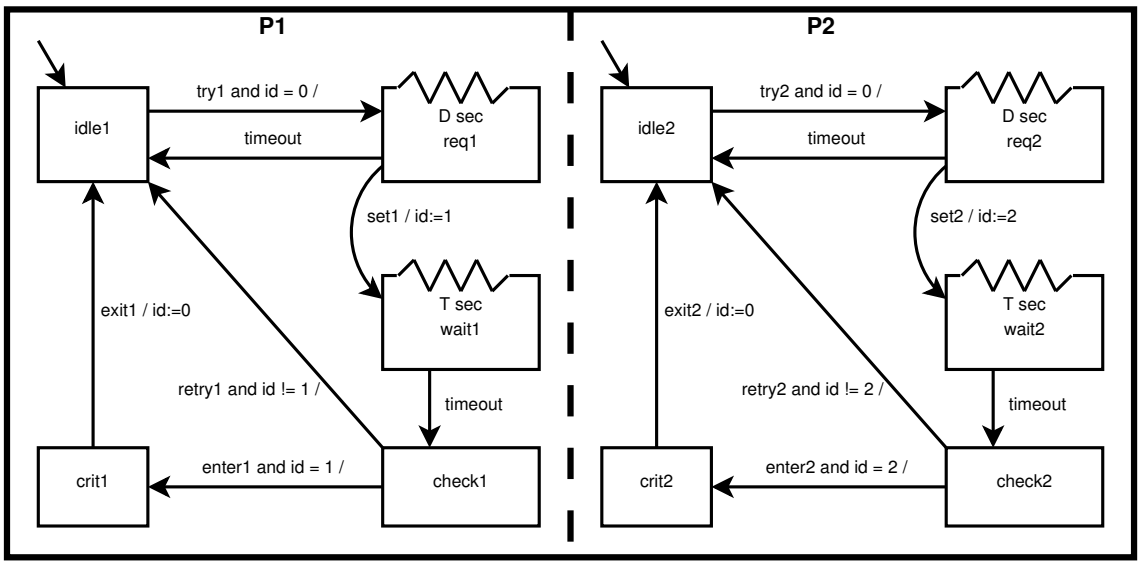


Figure 2: StateChart AND-state modeling a timed mutual exclusion protocol for two processes.

Your task is to provide an equivalent SDL model, assuming a non-deterministic interleaving semantics in case of concurrent effects. Your model should comprise three SDL processes: one for $P1$, one for $P2$, and one for the shared variable. You can assume that the external events `try`, `set`, `retry`, `enter`, and `exit` are sent non-deterministically by some unspecified environment process to $P1$ or $P2$, respectively.

Exercise 3: Basic Modeling with Lustre (20 pts.)

Implement the following functions in Lustre:

(a) A node *Sum*, with two inputs *Val* and *Reset*, and one output *Out*. *Out* is the sum of all values of *Val* that the node has seen since the last *Reset*. (5 pts.)

(b) A node *Still*, with one Boolean input X and one Boolean output Y. Y should be true precisely when X has been true all the time since the first point in time. (5 pts.)

(c) A node *MaxDistance*, with one Boolean input X, one integer input N, and a Boolean output *Ok*. *Ok* shall be true in cycle k, iff X was true at least once during the previous $N + 1$ clock cycles or if $k \leq N$. (10 pts.)

Exercise 4: Modeling a Lift Controller with Lustre (25 pts.)

Consider a simple lift, moving people up and down between two floors. In the lift, there are three buttons: One for going up, one for going down, and one `Stop` button. On each floor, there is a `Call` button. Furthermore, each floor has a sensor, indicating if the lift is on that floor or not. There is also a sensor in the lift, checking if the door to the lift is closed or not.

The lift is moved up and down by a motor that is on the roof of the building. The motor is controlled by two signals, `Motor_Up` and `Motor_Down`.

Your task is to give a Lustre implementation of a controller that observers the buttons that are being pressed, the sensors indicating the current position of the lift, and whether the door is open. Based on these observations, your controller decides if the motor should move the lift up or down, or do nothing.

For simplicity, we do not make a difference between if someone on floor 2 presses the `Call` button or if someone in the lift presses the `Up` button. Similarly for the `Call` button on floor 1 and the `Down` button in the lift. Furthermore, we assume that the `Up` and `Down` buttons are never pressed together in the same time instant or that two people on different floors call the lift in the same time instant.

Furthermore, your controller should satisfy the following requirements:

- The lift may only move when the door is closed and the `Stop` button is not pressed.
- The lift may not pass the end positions (that is: go through the roof or through the floor.)
- A moving lift only stops if either the `Stop` button is pressed, or the door is opened, or the lift has arrived at a floor.
- The lift must stop before changing direction.
- The signals sent to the motor may not be contradictory.

Your Lustre implementation must have the following interface:

```
node Control( Floor_1, Floor_2, Door_Closed, Call_1, Call_2, Stop : bool )
  returns ( Motor_Up, Motor_Down : bool );
```