IL2212 EMBEDDED SOFTWARE

Practical Homework 1

VERSION 1.0

The total amount of points in this first homework is 8 points. To pass the practical homework, 17 points out of 24 points in all three homework examinations are required.

1. (1 POINT) Synchronous data flow (SDF) has a lower expressiveness than cyclo-static data flow (CSDF).

   (a) Implement the tutorial CSDF application described in Section G.3 and in Listing G.2 of the lecture notes with only SDF actors using the `ForSyDe.Shallow` library. Try to be as close as possibe to the original CSDF model regarding the implemented functionality.

   (b) Is it possible to achieve exactly the same functionality as in the original CSDF model? Please give a convincing explanation or a counter-example.

2. (1 POINT) Implement the same scenario-aware data flow application described in Section G.4 and in Listing G.3 of the lecture notes in `ForSyDe.Shallow`, but only using one detector and one kernel process.

3. (2 POINTS) Section 5.2.4 in the lecture notes describes the Boolean data flow (BDF) model of computation (MoC).

   (a) Implement the BDF actors SWITCH and SELECT using the scenario-aware MoC in `ForSyDe.Shallow`.

   (b) Implement the BDF graph in the dashed box of Figure 5.20 in `ForSyDe.Shallow`. The actors C and D shall implement the identity function (C) and negation function (D). The actors A and B shall be replaced by input signals s_a and s_b, and the actor E by an output signal s_e.

4. (2 POINTS) Implement a scenario-aware data flow application of an adaptive system that uses one detector and one kernel using `ForSyDe.Shallow`.

   The kernel shall execute the following operation:

   - In *slow operation* the kernel shall consume 3 input tokens and produce one output token during each firing, which value is the sum of the consumed tokens.
   - In *fast operation* the kernel shall consume 2 input tokens and output one output token during each firing, which value is the sum of the consumed tokens.
   - The kernel shall start in *slow operation* mode.

- The kernel shall switch from *slow operation* mode to *fast operation* mode, when the sum of the consumed tokens is larger than 20.
- The kernel shall switch from *fast operation* mode to *slow operation* mode, when the sum of the consumed tokens is less than 10.

The following simulation example shows the desired functionality.

```
*SADF_Adapter> s_test = signal [4,5,6,8,8,9,9,8,2,4,8,5,2]
*SADF_Adapter> system s_test
{15,25,17,6,15}
```

5. (2 POINTS) Consider the following specification of a vending machine. The vending machine receives coins of the values 5 SEK and 10 SEK. The machine returns a bottle, when the right amount (10 SEK) is inserted. If too much money is inserted, a bottle and a coin (5 SEK) shall be returned. The machine has only a single coin injection slot, so it is not possible to inject two coins at the same time.

   Use the **ForSyDe.Shallow** library to create the following models for the control system of the vending machine. In all subtasks, the synchronous model of computation shall be used.

   (a) Create a model that has the following type:

```
vendingMachine :: Signal Bool -- Signal of 5 SEK coins
               -> Signal Bool -- Signal of 10 SEK coins
               -> Signal (Bool, Bool) -- Signal of
                                      -- (Bottle, Return)
```

   The system shall have the following behaviour during simulation.

```
*VendingMachine> s_coin5 = signal [False,True,True,
  True,False,False]
*VendingMachine> s_coin10 = signal [True,False,False,
  False,True,False]
*VendingMachine> vendingMachine s_coin5 s_coin10
{(True,False),(False,False),(True,False),(False,False),
  (True,True),(False,False)}
*VendingMachine> unzipSY $ vendingMachine s_coin5 s_coin10
({True,False,True,False,True,False},
 {False,False,False,False,True,False})
```

   (b) Create a model that has the following type:

```
data Coin = C5 | C10 deriving (Show, Eq, Ord)
data Bottle = B deriving (Show, Eq, Ord)
data Return = R deriving (Show, Eq, Ord)

type Coin_Event = AbstExt Coin
type Bottle_Event = AbstExt Bottle
type Return_Event = AbstExt Return

vendingMachine :: Signal Coin_Event -- Signal of Coins
               -> Signal (Bottle_Event, Return_Event)
                  -- Signal of (Bottle, Return)
```

The system shall have the following behaviour during simulation.

```
*VendingMachine> s_coin = signal [Prst C10, Prst C5, Prst C5,
  Prst C5, Prst C10, Abst]
*VendingMachine> vendingMachine s_coin
{(B,_),(_,_),(B,_),(_,_),(B,R),(_,_)}
*VendingMachine> unzipSY $ vendingMachine s_coin
({B,_,B,_,B,_},{_,_,_,_,R,_})
```