Homework 2 – Solutions

Problem 1 - FSM equivalent to the StateChart given:

```
while (1) {
    t = readTemp();
    p = readPressure();
    display(t);
    display(p);
    SetHeater(t);
    SetPressure(p);
}
```

Problem 2 a) Pros: Simple to implement, no OS support needed
Cons: Not truly concurrent, does not scale very well

```
while (1) {
    t = readTemp();
    p = readPressure();
    lock;
    display(t);
    unlock;
    SetHeater();
    SetPressure(p);
}
```

b) `controlTemp() {`
```
    t = readTemp();
    lock;
    display(t);
    unlock;
    SetHeater();
}
```

```
controlPressure() {
    p = readPressure();
    lock;
    display(t);
    unlock;
    SetPressure();
}
```

main() {
    create_thread(controlTemp);
    create_thread(controlPressure);
}

Pros: Truly concurrent, scales more easily (easy to add more tasks)
Cons: Needs OS support, explicit synchronization
Problem 3

a) There are many possible solutions for this problem
3. b) There are many possible solutions for this problem

Problem 4 (There are also many possible solutions for this problem)
Problem 5:

Entity LFSR is
    Port (i1,i2,i3,i4,i5: in_out BIT
         clk: in std_logic;)
End LFSR;

Architecture arch1 of LFSR is
begin
    process : (clk) – sensitivity list (process triggered on the rising edge of clk)
    begin
        if (clk’event) and (clk=’1’) then
            i5 <= i4;
            i4 <= i3;
            i3 <= i2;
            i2 <= i1 xor i5;
            i1 <= i5;
        end if;
    end process;
end arch1;

a) When there is no delay, we have no way of knowing how many transitions each change of data will cause, and thus the results of design are unpredictable. Note that in this case the functionality of design is to generate random number, so unpredictability is not as much of a problem as it would be in most other designs. The issue with unpredictability here is that the number generated may not be “random enough” since 0-delay transitions might cause repeated generation of the same number.

b) With finite delay we have predictable results. The design functions now as a Linear Feedback Shift Register (LFSR). LFSR is used to generate pseudo-random binary sequences.
Problem 6.
No it is not. Here is a quick proof out of Lee’s paper; it considers process F that is monotonic but not continuous:

\[ F(X) = \begin{cases} [0]; & \text{if } X \text{ is a finite sequence} \\ [0, 1]; & \text{otherwise} \end{cases} \quad (4) \]

Only two outputs are possible, both finite sequences. To show that this is monotonic, note that if the sequence \( X \) is infinite and \( X \subseteq X' \), then \( X = X' \), so

\[ Y = F(X) \subseteq Y' = F(X') . \quad (5) \]

If \( X \) is finite, then \( Y = F(X) = [0] \), which is a prefix of all possible outputs. To show that it is not continuous, consider the increasing chain

\[ \chi = \{ X_0, X_1, \ldots \} \], where \( X_0 \subseteq X_1 \subseteq \ldots \) \quad (6) \]

where each \( X_i \) has exactly \( i \) elements in it. Then \( \cup \chi \) is infinite, so

\[ F(\cup \chi) = [0,1] \neq \cup F(\chi) = [0]. \quad (7) \]

Iterative computation of this function is clearly problematic.

Problem 7.

This is an example of a symmetric bus arbitration mechanism. The bus is a ring on which a bunch of identical stations are hooked. In each instant, the user of the bus can request the bus and he can obtain it or not. A priority mechanism arbitrates simultaneous requests. A token defines the current initial station. At any time, the bus is granted to the first station that asks for it, starting from the initial station in clockwise order. To obtain fairness, the token is moved to the next station in each instant, so that each station is the initial one in turn.

Problem 8.
a. Write its incidence matrix.
   \[ A = \begin{bmatrix} 2 & -4 \\ -3 & e \end{bmatrix} \]

b. Assume \( e=3 \). Does it satisfy PASS test? Prove. NO – rank of \( A = 2 \)

c. Assume \( e=6 \). Does it satisfy PASS test? Prove. Yes – rank of \( A \) is 1

d. Derive a PASS assuming \( e=6 \).
   
   \[ \begin{align*}
   2v_1 - 3v_2 &= 0 \\
   -4v_1 + 6v_2 &= 0
   \end{align*} \]
   
   \[ V = [3,2] \] Thus, a good PASS is \{t1, t1, t1, t2, t2\}

e. Draw a coverability tree assuming that you are starting from \( x_0=[0,0] \) and have enough tokens to go through schedule \( S=\{t1,t1,t1,t2,t2\} \) Our initial starting condition is state \( x_0=[0,0] \):

   \[ [0,0] \rightarrow \text{via } t1 \rightarrow [2,-4] \rightarrow \text{via } t1 \rightarrow [4,-8] \rightarrow \text{via } t1 \rightarrow [6,-12] \rightarrow \text{via } t2 \rightarrow [3,-6] \rightarrow \text{via } t2 \rightarrow [0,0] \]

f. What initial condition is needed in order for schedule \( S \) to be executable? There have to be 12 tokens in \( p2 \)

g. What is the maximum buffer size needed in places \( p1 \) and \( p2 \) when using schedule \( S \)?

   At \( p1 \) we need 6 tokens (see coverability tree) and at \( p2 \) 12.