Assignment 3

Question 1. Complete the following table giving values of expressions in column 1 for the first 5 cycles. The values of integer flows for the first 5 cycles are given in rows, labelled P and Q.

Expression\Cycle	0	1	2	3	4	...
P	2	7	5	1	3	...
Q	0	13	6	5	2	...
pre(P)	nil	2	7	5	1	...
Q+1	1	14	7	6	3	...
P -> Q	2	13	6	5	2	...
P -> (pre(Q)+1)	2	1	14	7	6	...
P fby Q	2	0	13	6	5	...

Question 2. Consider the following **Lustre** nodes

```
node foo(P: int) returns (Q, R: int)
var W, Y, Z: int;
let
    Q  =  W* Z;
    W = 0 -> (pre(Z) +1);
     Z = W + Y;
    Y =  sqr(W);
    R = Q + W;
tel

node sqr(A :int) returns (B: int)
let
    B = A*A;
tel
```

Is this program causally correct? What is the order in which values of variables Q,R,W,Y,Z are calculated in each cycle?

Ans : **Yes**, the program is **casually correct.** The order in which values of variables Q,R,W,Y,Z are calculated in each cycle is as follows (the order is from **top to bottom**, i.e., W first and R last (or 5th here)):

W (1st)
Y (2nd)
Z (3rd)
Q (4th)
R (5th)

Assignment 3

Question 3.
(a) Define a **Lustre** node returning the following sequence of values. **(Code required)**
1, 4, 13, 40, 121, ...
(**Hint**: Consider how the difference of two successive terms grows.)
Note: In lustre, zero input nodes are not allowed. If your node is properly simulated in heptagon you can submit it. But keep the extension as **.lus (Update Feb 12, 19.09 PM)**

Ans : (In lustre, zero input nodes are not allowed. So I took a **dummy** value of type **bool** as **input**. It has **no significance** in the given program. **Can be ignored**. **Compile & Run** the node **sequence** to get the output).

Code :

```
node sequence(dummy: bool) returns (term: int)
var m : int;
let
  m = 1 -> (pre(m) * 3);
  term = 1 -> (pre(term) + m);
tel
```

(b) Complete the definition of following **Lustre** node **(Code required)**

```
node gen(req:bool) returns (ack: bool)
...
```

such that ack is true in the current cycle if and only if req has been true for the last 3 cycles (including the current cycle).
(Hint: First count for how many previous cycles req has been true continuously.)

Ans :

Code :

```
node gen(req:bool) returns (ack: bool)
var count: int;
let
   count = 0 -> if pre(req) then (pre(count) + 1) else 0;
   ack = if count >= 2 and req then 1 else 0;
tel
```

Question 4.
(a) Please describe in English the output produced by the following **Heptagon** node.

```
node t(x: bool^5) returns (y: bool);
let
    y = fold<<5>>(or)(x, false)
tel
```

Ans : The above **Heptagon** node **t** takes a **boolean** array **x** of 5 elements as input and produces a single boolean value **y** as output. **y** is **false (or 0)** if all the elements in **x** are **false,** otherwise, **y** is **true or 1** (i.e. if **at least 1 element** of **x** is **true** then **y** is also **true (or 1)**). Here, actually:

y = false or x[0] or x[1] or x[2] or x[3] or x[4]

Assignment 3

(b) What happens if you change the equation to **`y = fold<<5>>(or)(x, true)`**? Describe the output.

Ans : If we change the equation to **`y = fold<<5>>(or)(x, true)`**, then the output **`y`** will **always** be **`true (or 1)`**. Here, actually:

y = true or x[0] or x[1] or x[2] or x[3] or x[4] = true (or 1)

(c) Complete the definition of the following **Heptagon** node with parameter **n**.
(Code required)

```
node mutex<<n>>(ack: bool^n) returns (ok: bool)
...
```

The node should check for mutual exclusion of ack[i] and ack[j]. That is **ok is true provided expression ack[i] and ack[j] is false for all i, j pairs with i not= j.**

Ans : (We will compile and run a node similar to **display()** to run this **mutex** node)

Code :

```
node f(ack: bool; initial: int) returns (count: int)
let
   count =  if ack then initial + 1 else initial;
tel

node mutex<<n: int>>(ack: bool^n) returns (ok: bool)
var count: int;
let
  count = fold<<n>>f(ack, 0);
  ok = if count < 2 then true else false;
tel

-- Code 1 used for Testing
node display() returns(ok: bool)
let
  ok = mutex<< 5 >>([false,false,false,false,true]);
tel

-- Code 2 used for Testing
--node display() returns(ok: bool)
--let
--   ok = mutex<< 5 >>([false,false,true,false,true]);
--tel
```

Assignment 3

Question 5. Study the following **Heptagon** code for ripple adder.

```
node mxor(x, y: bool) returns (c: bool)
let
    c = (x and not y) or ((not x) and y);
tel

node fa(x, y, cin: bool) returns (z, cout: bool)
let
    z = mxor(mxor(cin, x), y);
    cout = if cin then (x or y) else (x and y);
tel

node rippleadd<<n:int>>(a: bool^n; b: bool^n) returns (c: bool^n;
over: bool)
let
    (c,over) = mapfold<<n>>fa(a, b, false);
tel
```

A sample code file with the above code simulating a 4-bit adder is given: Download File.

(a) Manually compute the output at 0th cycle of rippleadd for constants a=[0,1,1,0,1,1,1,0] and b=[1,1,0,1,0,1,1,1] given as input to rippleadd<<8>>(a,b).

Ans : 1
0
0
0
0
1
1
0
1 (This **1** is value of **over** (i.e. overflow) defined in **rippleadd** node)

(b) Using this node rippleadd, define a node **counter** which counts in binary modulo 64. (It should **output unsigned 8 bit binary numbers corresponding to the decimal numbers** 0, 1, 2, ..., 63, 0, 1, 2,.... in successive cycles). **(Code required)**

Ans : (We will compile and run the node **counter()**. The node counter() also gives **overflow** as output which is **ignored**. The mod 64 values are represented by c7,c6,…,c0).

Code :

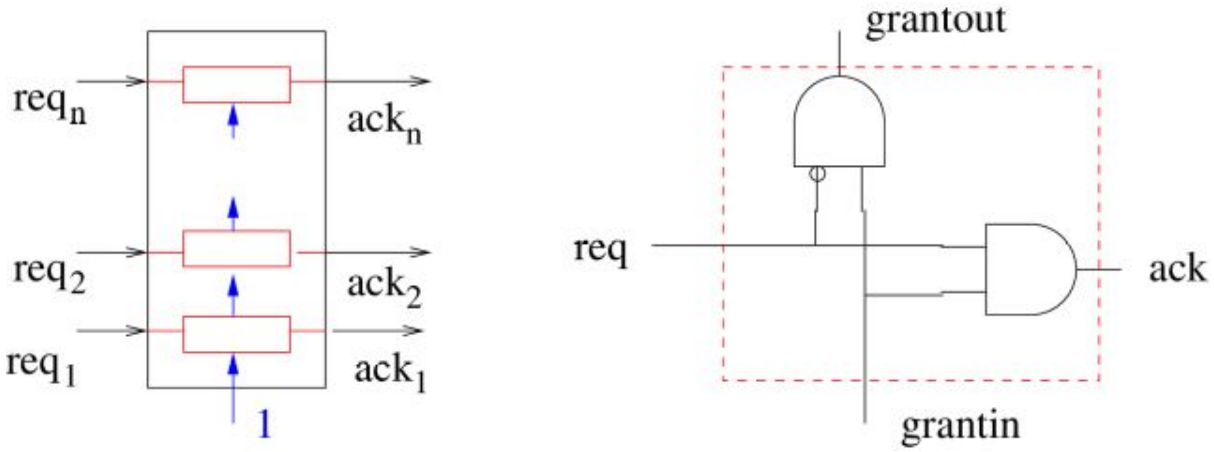
```
node mxor(x, y: bool) returns (c: bool)
let
   c = (x and not y) or ((not x) and y);
tel
```

Assignment 3

```
node fa(x, y, cin: bool) returns (z, cout: bool)
let
   z = mxor(mxor(cin, x), y);
   cout = if cin then (x or y) else (x and y);
tel

node rippleadd<<n:int>>(a: bool^n; b: bool^n) returns (c: bool^n; over: bool)
let
   (c,over) = mapfold<<n>>fa(a, b, false);
tel
```

```
node counter() returns (c0, c1, c2, c3, c4, c5, c6, c7: bool; overflow: bool)
var a, b, c: bool^6;
let
   a = [false, false, false, false, false, false];
   b = [true, false, false, false, false, false];
   (c,overflow) = (a, false) -> rippleadd<<6>>(pre(c), b);
   c0 = c[0]; c1 = c[1]; c2 = c[2]; c3 = c[3]; c4 = c[4]; c5 = c[5]; c6 = false; c7 =
false;
tel
```

Question 6. (Synchronous Bus Arbiter) An arbiter arbitrates between multiple requests coming at each cycle and gives acknowledgement to at most one of them. Consider the arbiter circuit below. **(Code required)**



Each **red box** (denoting a cell) in the left hand side figure is expanded to the **cell** circuit given in the right hand side figure.

Using **Heptagon**, model each cell as a
`node cell(req, grantin: bool) returns (ack, grantout: bool)`

Model a 5 cell arbiter as an assembly of 5 cells as shown in the figure. Call this
`node arbiter(req:bool^5) returns (ack:bool^5)`

Assignment 3

(**Hint**: See the implementation of rippleadd given in the previous question. Extra credit will be given if you can program this as an **n cell arbiter** with **parameter n**.)

Define a suitable **display node** to show the output. Simulate the arbiter using the tool **Heptagon** and check its functioning. Submit a screenshot of sample output using the sim2chro display.

Which of the following properties does this arbiter have?

- **(a)** Mutual exclusion of ack[i], ack[j] for i not= j.
- **(b)** No spurious ack, i.e. ack[i] => req[i]
- **(c)** No lost cycles, i.e. in any cycle, if there is at least 1 true request, the arbiter should have at least one true ack.

Ans : (The code given below is the implementation of **n bit arbiter.** For 5 bit arbiter, uncomment the 2 functions given in comments below and comment the last 2 functions. **Compile and run** the node **display** to get the **outputs**.)

This arbiter has all the 3 properties :

a) Mutual Exclusion
b) No spurious ack
c) No lost cycles

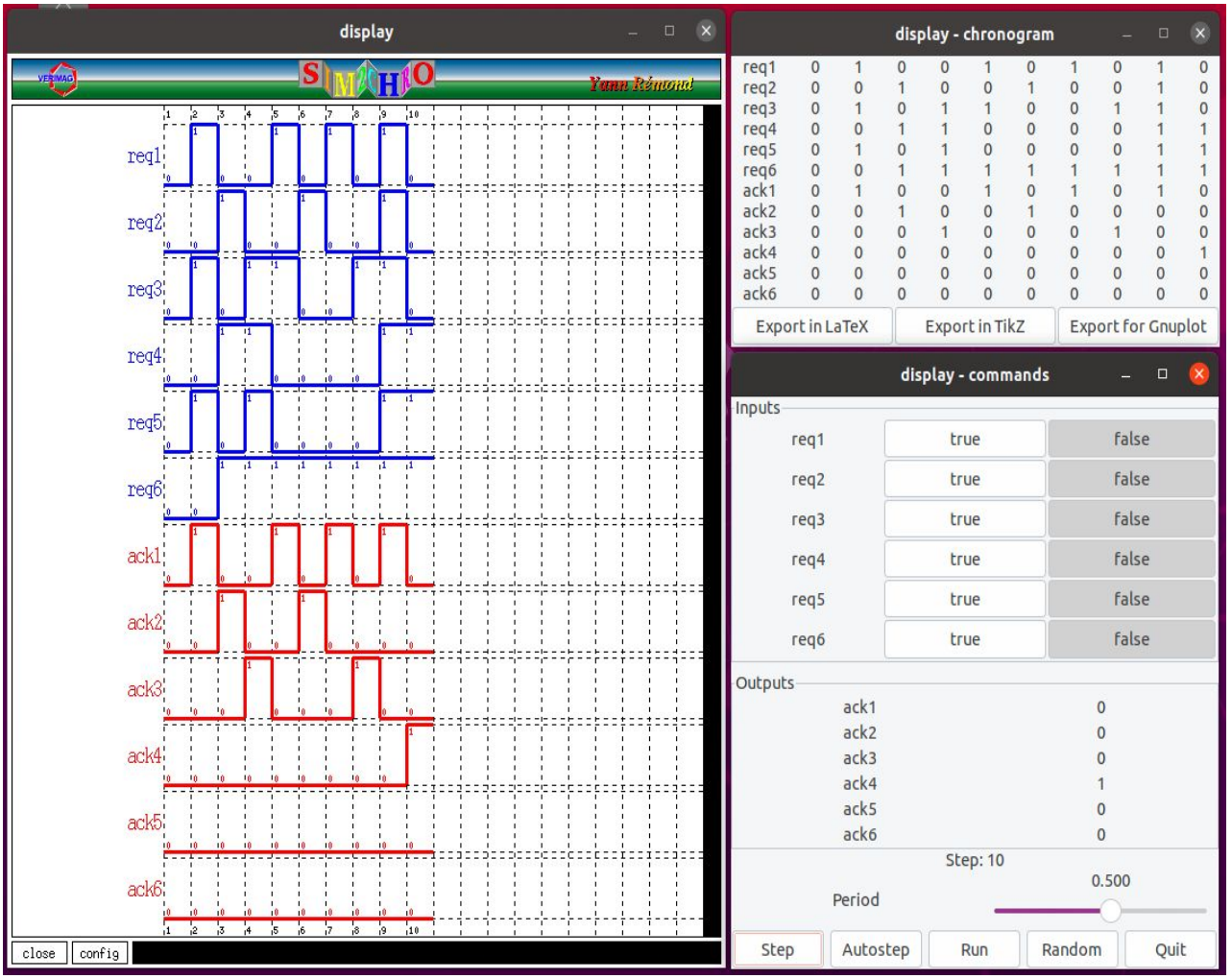
Code :

```
node cell(req, grantin: bool) returns (ack, grantout: bool)
let
   ack = req and grantin;
   grantout = (not req) and grantin;
tel

--node arbiter(req:bool^5) returns (ack:bool^5)
--var out: bool;
--let
--    (ack, out) = mapfold<<5>>cell(req, true);
--tel

--node display(req1, req2, req3, req4, req5: bool)
--returns (ack1, ack2, ack3, ack4, ack5: bool)
--var req, ack: bool^5;
--let
--    req = [req1, req2, req3, req4, req5];
--    ack = arbiter(req);
--    ack1 = ack[0]; ack2 = ack[1]; ack3 = ack[2]; ack4 = ack[3]; ack5 = ack[4];
--tel
```

Assignment 3

```
node arbiter<<n:int>>(req:bool^n) returns (ack:bool^n)
var out: bool;
let
   (ack, out) = mapfold<<n>>cell(req, true);
tel

node display(req1, req2, req3, req4, req5, req6: bool)
returns (ack1, ack2, ack3, ack4, ack5, ack6: bool)
var req, ack: bool^6;
let
   req = [req1, req2, req3, req4, req5, req6];
   ack = arbiter<<6>>(req);
   ack1 = ack[0]; ack2 = ack[1]; ack3 = ack[2]; ack4 = ack[3]; ack5 = ack[4]; ack6 =
ack[5];
tel
```

Screenshot of output :



Assignment 3

Question 7. A monitor node for a property S takes as input a set of flows to observe (e.g. p, q: bool). It outputs a single boolean flow ok. The idea is that at every clock cycle, ok is true if the property S holds for the past sequence of inputs (including the current cycle). For example:
Property S: “p is continuously true in the past” has the monitor node

```
node smonitor(p: bool) returns (ok: bool)
let
    ok = p -> (pre(ok) and p);
tel
```

Answer the following

(a) Give the output of the above **smonitor** node for the input flow
p = true true true false true false true true

Ans : ok = true true true false false false false false (or 1 1 1 0 0 0 0 0)

(b) Give a monitor for the following property: *“p is continuously true in the past AND q has occured at least once in the past.”* **(Code required)**

Ans :

Code :

```
node monitor(p, q: bool) returns (ok: bool)
var s, t: bool;
let
   s = q -> (pre(s) or q);
   t = p -> (pre(t) and p);
   ok = s and t;
tel
```

(c) Give a monitor node for the following property: Assume that a, b, c are boolean flows. *“Everytime **a** occurs, **c** will remain continuously true from then on until a **b** occurs”.* Specify additional assumptions that you make in your design (E.g. what happens if a and b occur simultaneously). **(Code required)**

Ans :

Assumption : if a and b occurs simultaneously, i have given priority to b (i.e., c will be false in this case)

Code :

```
node monitor(a, b: bool) returns (c: bool)
let
   c = (a and (not b)) -> ((pre(c) or a) and (not b));
tel
```

Assignment 3

(d) Give a monitor node for the following property: Assume that req and ack are boolean flows. “If req has been true for the last 3 cycles (including the current cycle) then ack must be true in the current cycle.” **(Code required)**
Clarification: req and **ack** are both **input boolean flows. ok** is the **output flow. ok is true if the property is observed/followed. (Update Feb 12, 18.00 PM)**

Ans :

Code :

```
node monitor(req, ack:bool) returns (ok: bool)
var count: int;
let
    count = 0 -> if pre(req) then (pre(count) + 1) else 0;
    ok = (not (if count >= 2 and req then 1 else 0)) or ack;
tel
```