**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) |  |  |  |  |  |  |
| Q+1 |  |  |  |  |  |  |
| P -> Q |  |  |  |  |  |  |
| P -> (pre(Q)+1) |  |  |  |  |  |  |
| P fby Q |  |  |  |  |  |  |

**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?

**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)**

**(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.)

**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

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

**(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.**

**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);

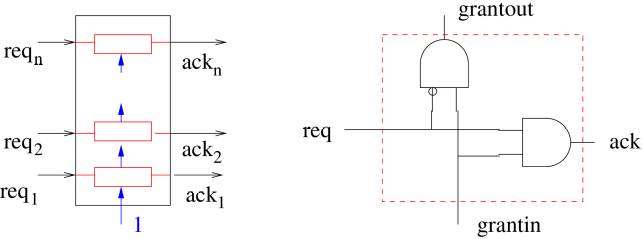
tel

A sample code file with the above code simulating a 4-bit adder is given: [Download File](https://drive.google.com/file/d/1N2xKjaIJKYc6iZbB7ErPer2Gkit6nfLI/view?usp=sharing).

**(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).

**(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)**

**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)

(**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?

1. Mutual exclusion of ack[i], ack[j] for i not= j.
2. No spurious ack, i.e. ack[i] => req[i]
3. 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.

**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

1. Give the output of the above **smonitor** node for the input flow

p = true true true false true false true true

1. 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)**
2. 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)**
3. 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)**