Recapitulare

(* Vă rog să citiți și observațiile de la finalul documentului)

Laboratorul S1

1) Se dă următorul tabel de adevăr:

Inputs				Outputs			
I_3	I_2	I_1	I_0	O_3	O_2	O_1	O_0
0	0	0	0	1	1	1	1
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	1	0	1	0
0	1	0	0	1	0	1	1
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	1
0	0	0	0	0	1	1	0
1	0	0	0	0	1	1	1
1	0	0	1	0	0	0	0

Obs: Mintermii de la 10 → 15 sunt considerați elemente don't care

Cerințe:

a) Să se minimizeze pe foaie fiecare din ieșirile listate în tabelul BCD folosind metoda diagramelor Karnaugh.

b) Să se construiască un modul, folosing limbajul Verilog, care implementează funcția booleană rezultată după minimizare. Modulului i se va atribui un nume sugestiv (ex. ***minimization***).

2) Se dau următoarele expresii booleene:

i) $F(A,B) = \overline{(A+B)} \cdot (\bar{A} + \bar{B})$

ii) $F(A,B,C) = \bar{A} \cdot B \cdot C + A \cdot C$

iii) $F(A,B,C) = \bar{A} \cdot \bar{C} + A \cdot B \cdot C + A \cdot \bar{C}$

iv) $F(A,B,C,D) = (B \cdot C + \bar{A} \cdot D) \cdot (A \cdot \bar{B} + C \cdot \bar{D})$

Cerințe:

a) Simplificați pe foaie expresiile și aduceți-le la o formă echivalentă exprimată cu un număr minim de litere, folosind axiomele și teoremele algebrei booleene.

b) Redactați câte un modul, folosind limbajul Verilog, care implementează funcțiile booleene rezultate în urma simplificării celor 4 expresii listate mai sus. Modulului i se va atribui un nume sugestiv (ex. ***boolean_function_1***).

Laboratorul S2

3) Proiectați, utilizând limbajul Verilog, un modul care selectează, dependent de o intrare ***swch*** cei mai semnificativi 10 biți ai intrării ***i*** sau cei mai puțin semnificativi 10 biți ai intrării ***i***. Intrarea ***i*** are 16 biți iar ieșirea modulului se numește ***o***. Să se redacteze un modul testbench pentru verificarea modului ***switch*** implementat anterior.

4) Construiți, utilizând limbajul Verilog, un modul care calculează rezultatul înmulțirii cu ***x*** a numărului pe ***x*** biți de la intrare, fără a folosi operatorul Verilog de deplasare **<<**. Valoarea lui ***x*** este transformarea în zecimal a rezultatului expresiei ***x = 1 1 1 0 << 2***.

Laboratorul S3

5) Proiectați, folosind limbajul Verilog, un modul pentru calcularea rezultatului împărțirii intrării pe 16 biți la 16, fără a folosi operatorul de divizare Verilog **/**. Redactați un testbench pentru verificarea modului Verilog implementat anterior.

6) Implementați, utilizând Verilog, un modul care atașează bitul de paritate impară unei intrări pe 7 biți. Ieșirea modulului, pe ***8 biți***, va avea bitul de paritate în cea mai semnificativă poziție. Construiți un testbench pentru verificarea modulului Verilog implementat anterior.

Laboratorul S4

7) Construiți, în limbajul Verilog, un multiplexor 8-la-1, pe 1 bit, denumit ***mux_3s_1b***. Se cere desenarea arhitecturii corespunzătoare multiplexorului din enunț și implementarea codului Verilog aferent.

8) Implementați următoarea funcție, folosind multiplexorul 8-la-1 construit la punctul anterior :

$$f = \sum (1, 3, 4, 11, 12, 13, 14)$$

Obs: Restul elementelor care nu apar în sumă de mintermi sunt considerate nule.

Laboratorul S5

9) Redactați, folosind limbajul Verilog, un testbench pentru verificarea exahaustivă a modulului ***mux_3s_1b*** implementat la problema 7.

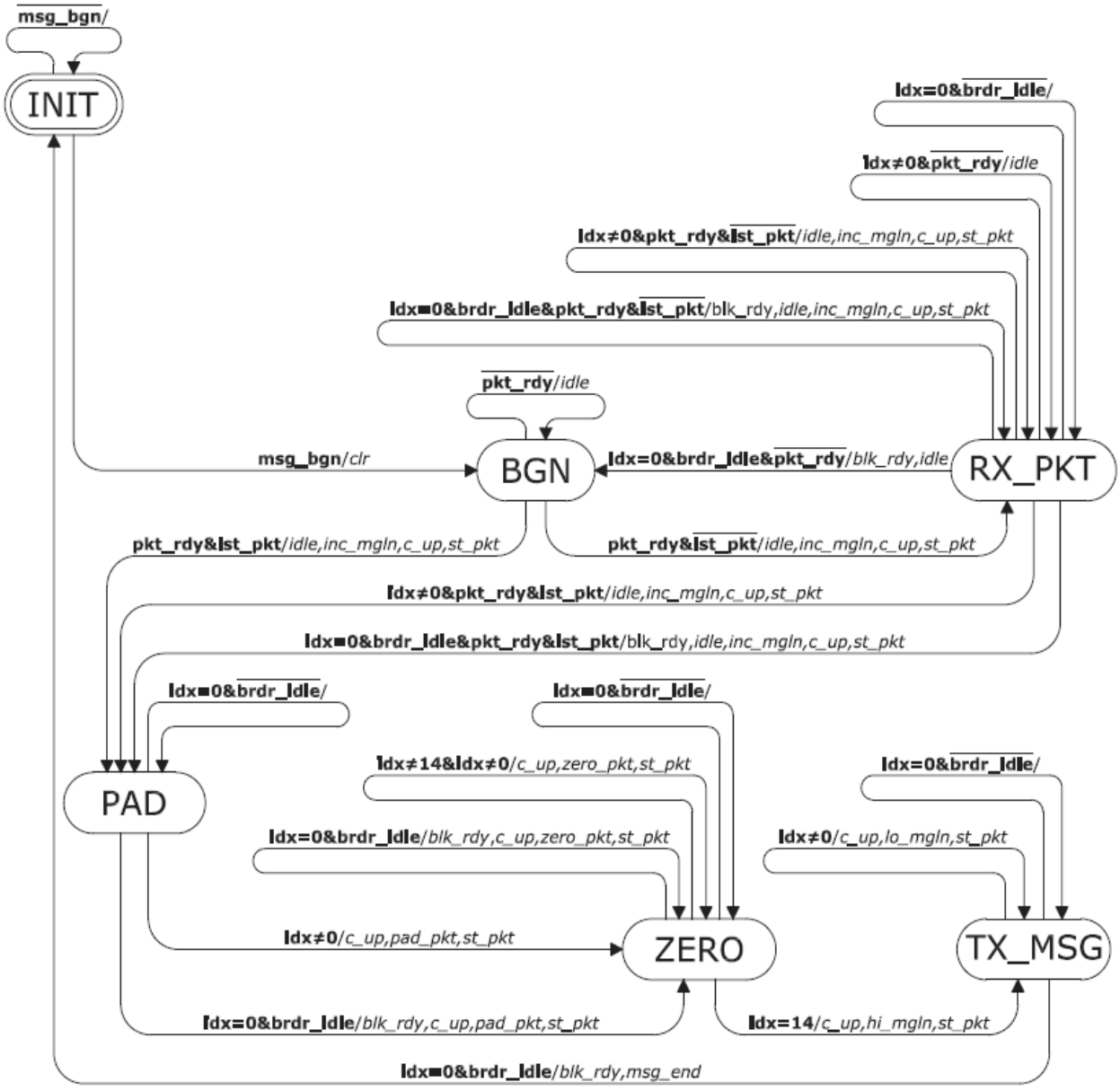
10) Construiți, utilizând limbajul Verilog, un testbench pentru verificarea corectitudinii modulului de implementare a funcției de la problema 8.

Laboratorul S6

11) Pe pagina 5 este ilustrată o mașină Mealy reprezentând unitatea de control a modulului de preprocesare a intrării pentru o aplicație criptografică. Stările interne ale mașinii sunt:

- **INIT** – starea implicită, atinsă după reset
- **BGN (begin) –** atinsă după ce msg_bgn a fost activată
- **RX_PKT** (receive packet) – pachetele sunt primite pe liniile pkt
- **PAD** – adaugă un pachet cu un bit de 1 urmat de 31 de biți de 0
- **ZERO** – adaugă toate pachetele cu valoarea 0
- **TX_MSG** (transmit message) – adaugă lungimea mesajului, finalizează transmiterea mesajului

Construiți pașii pentru diagrama de tranziție fără a implementa modulul de preprocesare a intrării pentru o aplicație criptografică.


msg_bgn/
INIT
idx=0&brdr_idle/
idx≠0&pkt_rdy/idle
idx≠0&pkt_rdy&lst_pkt/idle,inc_mgln,c_up,st_pkt
idx=0&brdr_idle&pkt_rdy&lst_pkt/blk_rdy,idle,inc_mgln,c_up,st_pkt
pkt_rdy/idle
msg_bgn/clr
BGN
idx=0&brdr_idle&pkt_rdy/blk_rdy,idle
RX_PKT
pkt_rdy&lst_pkt/idle,inc_mgln,c_up,st_pkt
pkt_rdy&lst_pkt/idle,inc_mgln,c_up,st_pkt
idx≠0&pkt_rdy&lst_pkt/idle,inc_mgln,c_up,st_pkt
idx=0&brdr_idle&pkt_rdy&lst_pkt/blk_rdy,idle,inc_mgln,c_up,st_pkt
idx=0&brdr_idle/
idx=0&brdr_idle/
PAD
idx≠14&idx≠0/c_up,zero_pkt,st_pkt
idx=0&brdr_idle/
idx=0&brdr_idle/blk_rdy,c_up,zero_pkt,st_pkt
idx≠0/c_up,lo_mgln,st_pkt
idx≠0/c_up,pad_pkt,st_pkt
ZERO
TX_MSG
idx=0&brdr_idle/blk_rdy,c_up,pad_pkt,st_pkt
idx=14/c_up,hi_mgln,st_pkt
idx=0&brdr_idle/blk_rdy,msg_end

Laboratorul S7

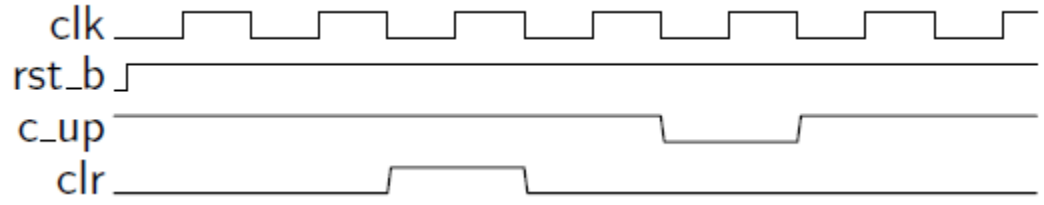
12) Construiți un multiplexor 4-la-1 parametrizat, numit ***mux_2s***. Modulul poate fi parametrizat ținând cont de lățimea de biți ai intrării, respectiv ieșirii, așa cum este ilustrat în interfața de mai jos:

```
1  module mux_2s #(
2          parameter w = 4                  //width parameter
3  )(
4          input [w-1:0] d0,d1,d2,d3,  //4 data inputs
5          input [1:0] s,              //selection input
6          output [w-1:0] o            //data output
7  );
```

13) Construiți un numărător sincron, denumit ***cntr***. Modulul poate fi parametrizat ținând cont de lățimea de biți ai intrării, respectiv al vectorului de inițializare. Interfața numărătorului, pe lângă semnalul de ***clk***, va include următoarele semnale:

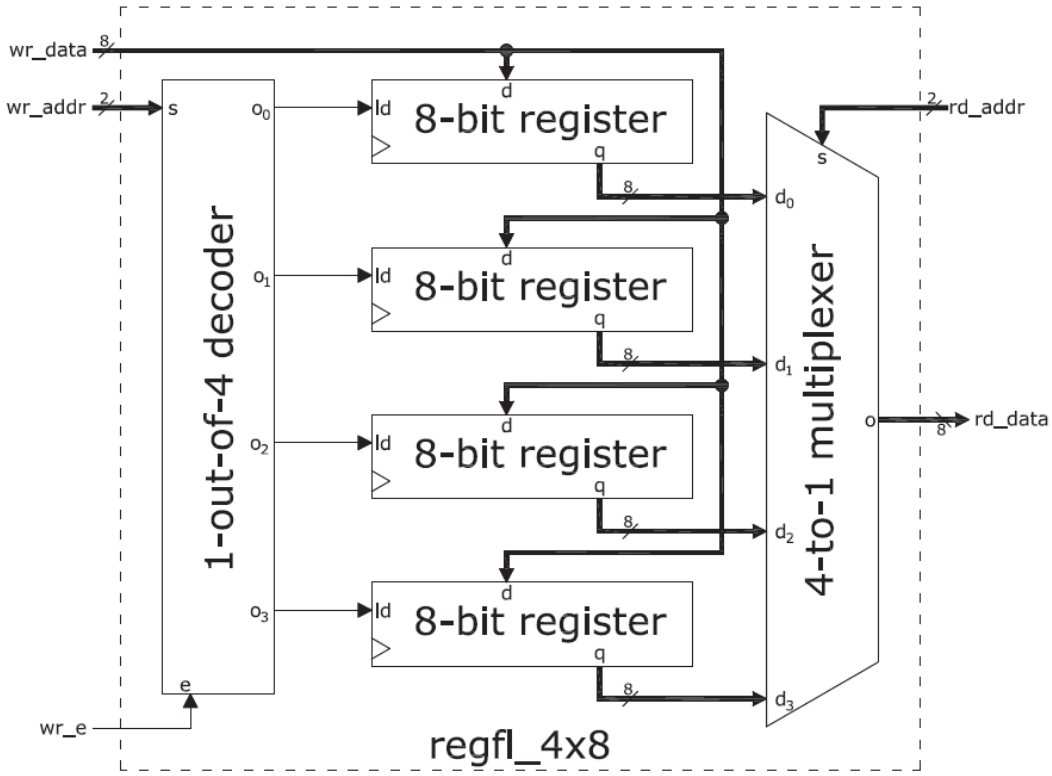
- ***rst_b***, asincron, activ pe LOW, va seta conținutul vectorului de inițializare
- ***c_up,*** sincron, activ pe HIGH, incrementează valoarea numărătorului
- ***clr,*** sincron, activ pe HIGH, va seta conținutul vectorului de inițializare, cu prioritate mai mare față de ***c_up***
- ***q***, ieșirea, memorează conținutul numărătorului.

Parametrizați numărătorul pe 8 biți cu un vector de inițializare având valoarea ***8'hff*** și testați modulul cu intrările generate de următoarea diagramă:



$T_{clk} = 100ns,\ Pulse_{rst\_b} = 5ns$

14) Construiți arhitectura de register file *4x8* așa cum este ilustrată mai jos:



Register File-ul are următoarea interfață:

```
1  module regfl_4x8 (
2     input clk,
3     input rst_b,//asynch
4     input [7:0] wr_data,
5     input [1:0] wr_addr,
6     input wr_e,
7     output [7:0] rd_data,
8     input [1:0] rd_addr
9  );
```

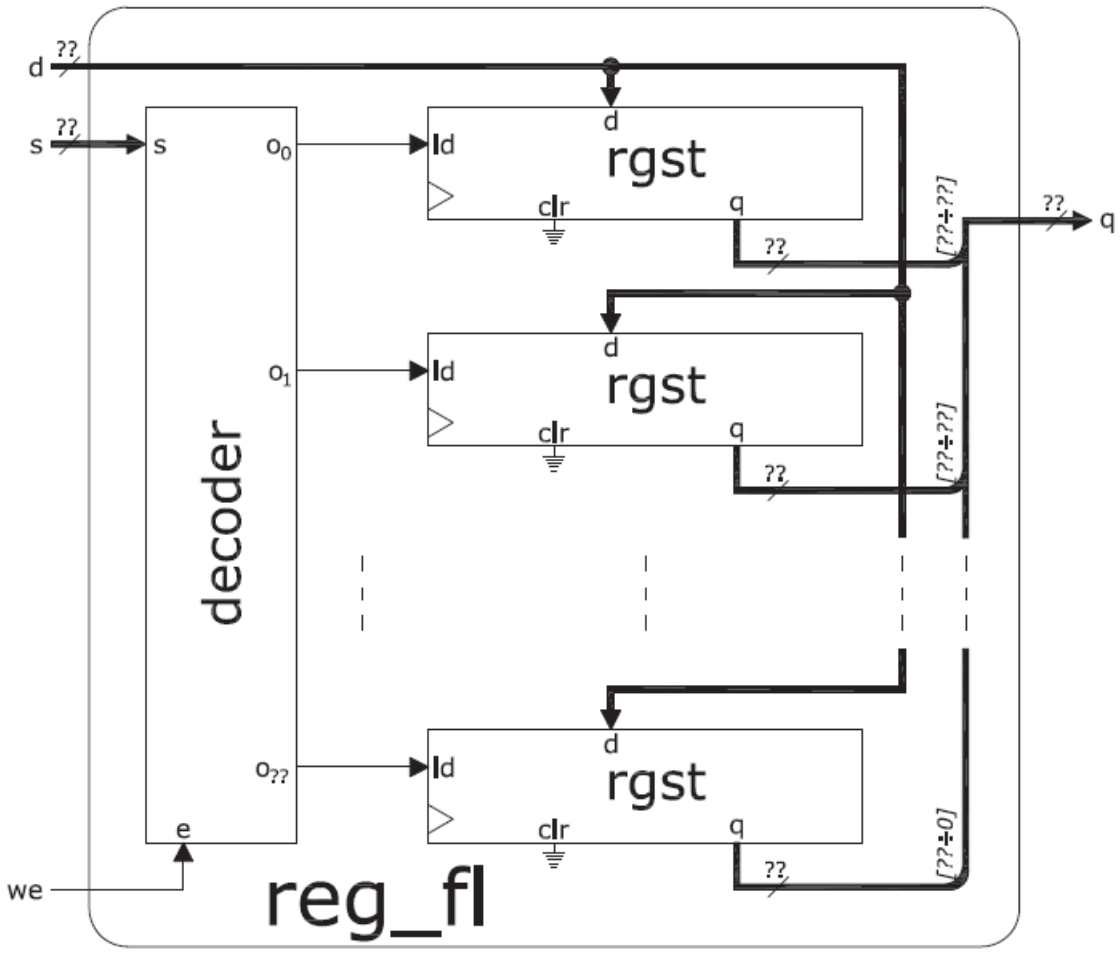
Din modul lipsește intrarea de ***read enable*** (validare intrare), ***rd_e***, însemnând că la oricare moment, unul dintre conținuturile celor 4 registre va fi livrat către ***rd_data***.

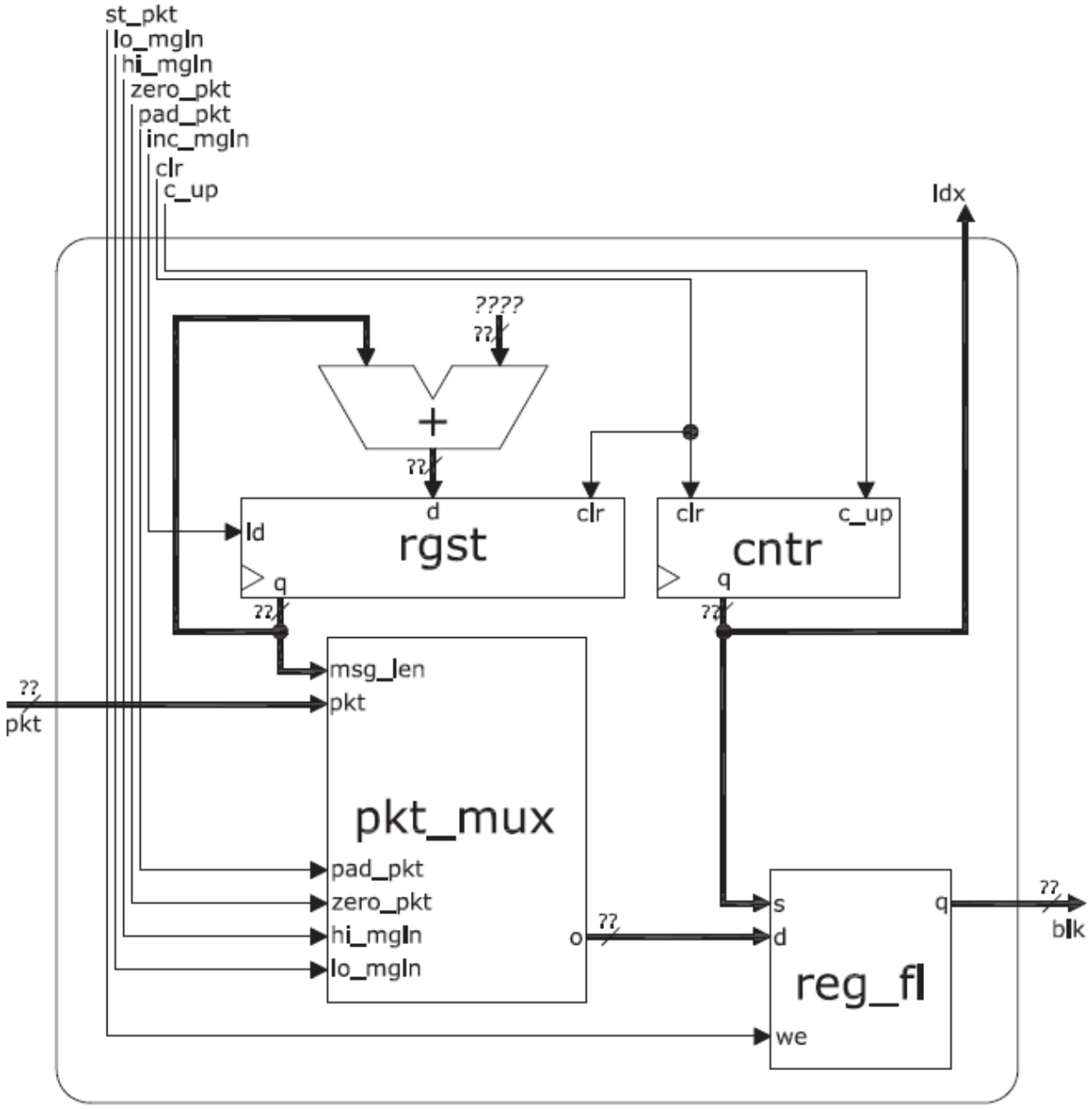
Laboratorul S8

15) Construiți calea de date a unei unități de preprocesare personalizată, inspirată din arhitectura Secure hash Algorithm 2 (SHA-2) pe 256 de biți care este implicată în faza de preprocesare.

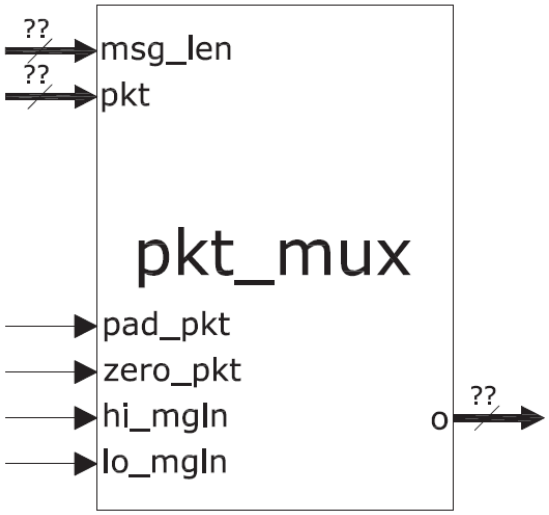
Unitatea de preprocesare operează cu pachete pe 4 biți și blocuri declarate pe 64 de biți. Mesajul inițial are o lungime de ***L*** biți, L fiind un multiplu de 8. Mesajul inițial este completat cu 3 biți de 0 urmați de k biți de 1, astfel încât are loc relația ***L + 3 + k = 56 (mod 64)***. Dimensiunea în bytes al mesajului inițial este atașată la finalul celor k biți de 1, ca un număr reprezentat pe 8 biți. Substituiți registrul și structura de sumator care stochează și actualizează dimensiunea în bytes al mesajului inițial cu un *circuit counter personalizat*.

Obs: Utilizați blocul generate pentru instanțierea registrelor.



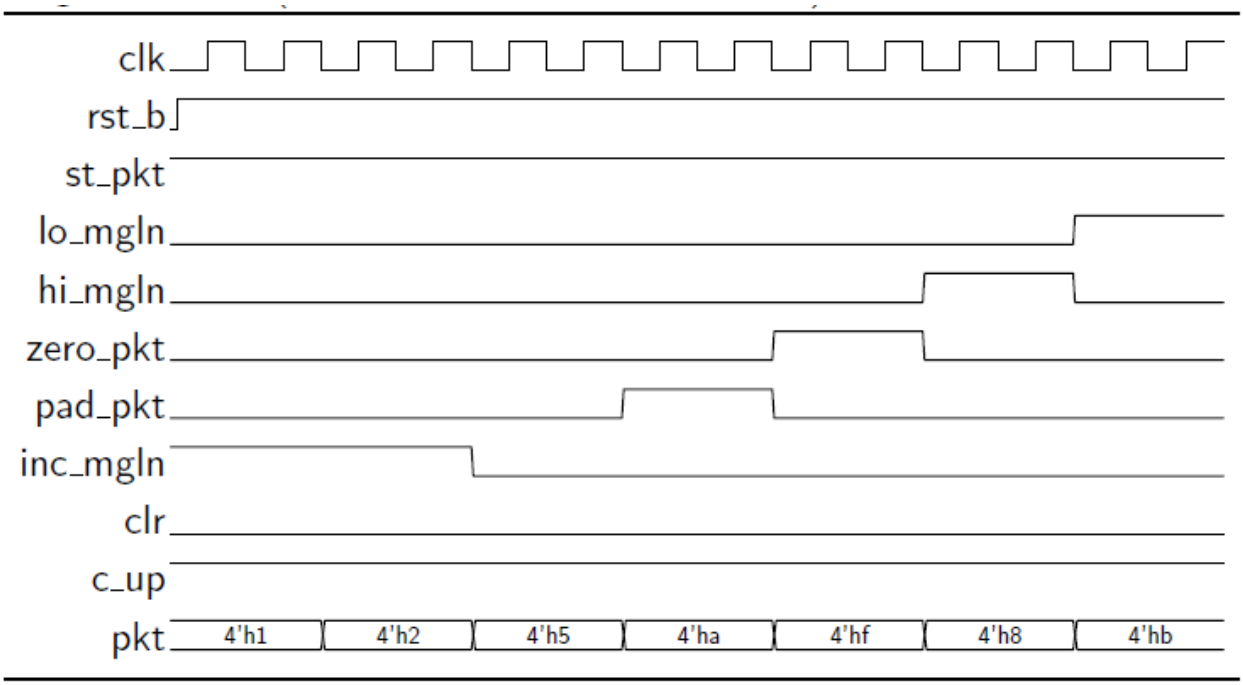


Arhitectura personalizată a intrării de preprocesare



Varianta personalizată de multiplexor

Testați modulul căii de date cu intrări generate conform cronogramei de mai jos:



$T_{clk} = 100ns,\ Pulse_{rst\_b} = 5ns$

Laboratorul S9

16) Construiți implementarea hardware al următorului algoritm:

```
Input: Message M                          ▷ M is made up of 32-bit words M_i, i = 0...15
Output: 32-bit word B_0                   ▷ Deliver B_0 in each iteration
 1: procedure MessageSchedule(M)
 2:     B_0 ← M_0        ▷ Initialize the 16, 32-bit variables B_i from message's words M_i
 3:     B_1 ← M_1
 4:     ...
 5:     B_14 ← M_14
 6:     B_15 ← M_15
 7:     for i = 0 to 63 do                 ▷ Update value of the 16 variables B_i
 8:         U ← σ_1(B_14) + B_9 + σ_0(B_1) + B_0
 9:         B_0 ← B_1
10:         B_1 ← B_2
11:         ...
12:         B_14 ← B_15
13:         B_15 ← U
14:     end for
15: end procedure
```

Cuvântul ***M0*** al mesajului inițial este cel mai semnificativ cuvânt. În mod similar ***B0*** este cea mai semnificativă variabilă. În primul tact de ceas, ***B0*** este setat la valoarea ***M0*** (primul cuvânt al mesajului M) și este livrat la ieșire. Începând cu cel de-al doilea tact de ceas, variabila ***B0*** este actualizată conform algoritmului prezentat și este livrată la ieșire.

Funcțiile ***sigma_0*** și ***sigma_1*** sunt definite astfel:

$$\sigma_0(x) = right\_rotate(x,7) \oplus right\_rotate(x,18) \oplus right\_shift(x,3)$$
$$\sigma_1(x) = right\_rotate(x,17) \oplus right\_rotate(x,19) \oplus right\_shift(x,10)$$

unde *rotate_right(x,p)* rotește cuvântul x către dreapta cu p poziții și *right_shift(x,p)* deplasează către dreapta cu p biți cuvântul de la intrare ***x***.

Proiectați pe foaie diagrama bloc ai implementării și furnizați un testbench pentru verificarea modulului redactat.

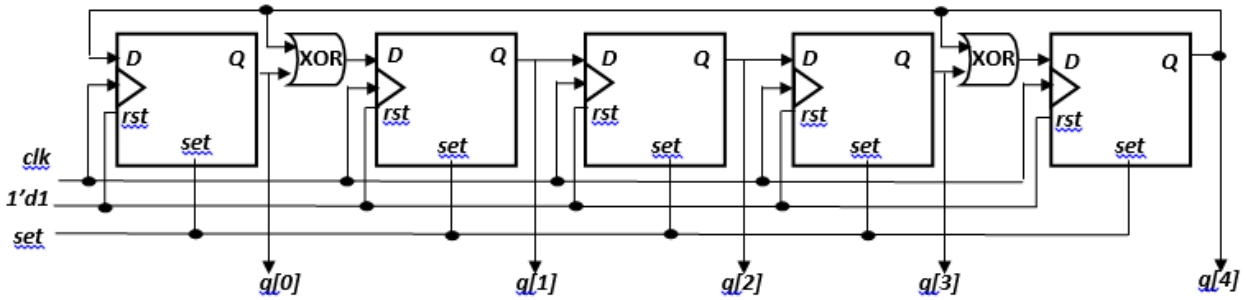
Laboratorul S10

17) Construiți o unitate de control personalizată inspirată din unitatea de control a căii de preprocesare a design-ului SHA-2 pe 256 de biți. Construiți diagrama de tranziții pentru unitatea de control personalizată și implementați modulul în Verilog.
Unitatea de control personalizată diferă de unitatea de control al design-ului SHA-2 prin următoarele aspecte:

- *structura hash* dispune de o ieșire suplimentară ***hng_busy*** (hash engine busy), activată când nu mai poate primi un alt bloc pentru prelucrare.
- curierul de mesaje are o ieșire adițională, ***pkt_val*** (valid packet), activată la fiecare ciclu de clock când încarcă un nou pachet pe liniile de date.
- unitatea de preprocesare are o intrare ***stall2***, după inițializare, unitatea inspectează dacă ***stall2*** este activă și dacă da, va amâna procesarea pachetelor de date inițiale cu două cicluri de clock.

Laboratorul S11

18) Implementaţi, utilizând limbajul Verilog, un LFSR (Linear Feedback Shift Register) pe 5 ranguri, aşa cum este ilustrat mai jos. Determinați, de asemenea, pe foaie periodicitatea LFSR-ului.



Observații privind rezolvarea problemelor din ***Lab 1 → Lab 11***

✞ Se vor crea proiecte noi pentru fiecare problemă în parte, specificând un nume sugestiv, *exemplu:* ***Lab1_Problema1***

✞ Fiecare proiect va conține numărul necesar de module în vederea rezolvării problemei, în funcție de cerințe. Se vor păstra numele modulelor date din enunțul problemei, unde este cazul, altfel se va atribui modulului un *nume abreviat sugestiv*.

✞ Pentru fiecare problemă, pe lângă modulele create, se va cere suplimentar construirea unui *modul* ***testbench*** pentru verificarea funcțională a modulelor implementate, spre exemplu: în cazul problemei cu multiplexorul se va numi fișierul ***mux_3s_1b_tb.v***.

✞ După implementarea modulului *testbench*, se va trece la simularea descrierii comportamentale a modulelor. Pentru aceasta se vor face *print-screen-uri* (capturi) după configurarea parametrilor semnalelor și după forma undelor de intrare.

✞ Desenele arhitecturilor cerute, acolo unde este cazul, vor fi schițate fie de mână sau asistat de calculator. În ambele situații acestea vor fi adăugate la arhiva problemelor rezolvate.

✞ În final se va crea o arhivă cu toate fișierele indicate mai sus și va fi trimisă pe adresa de mail: **rtr.raul@gmail.com**