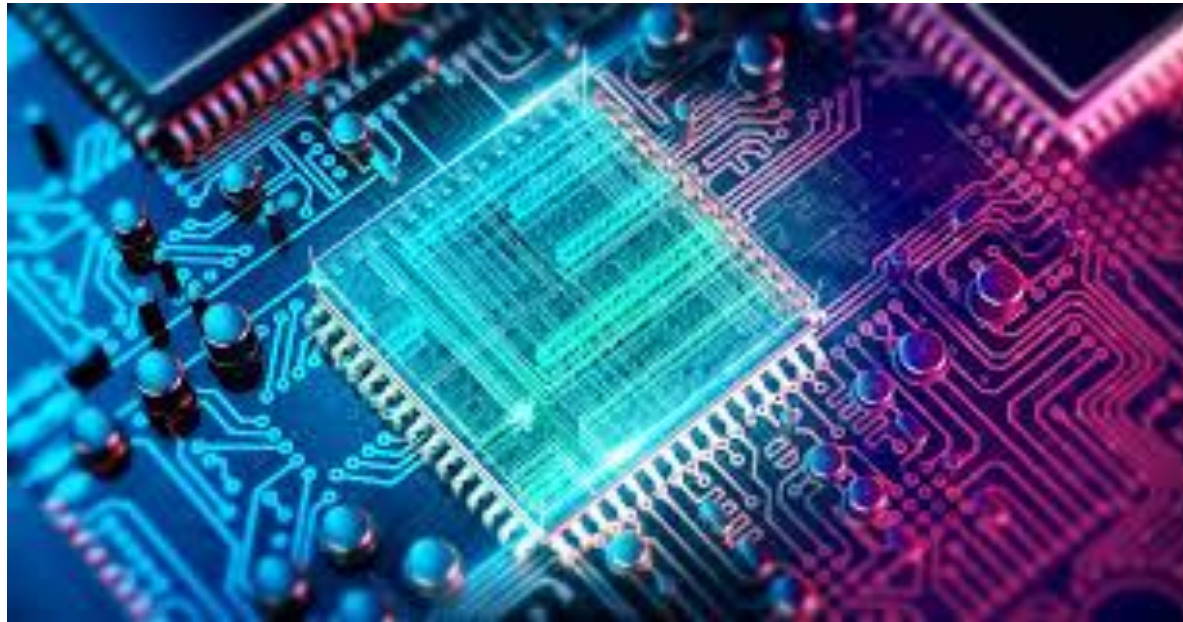
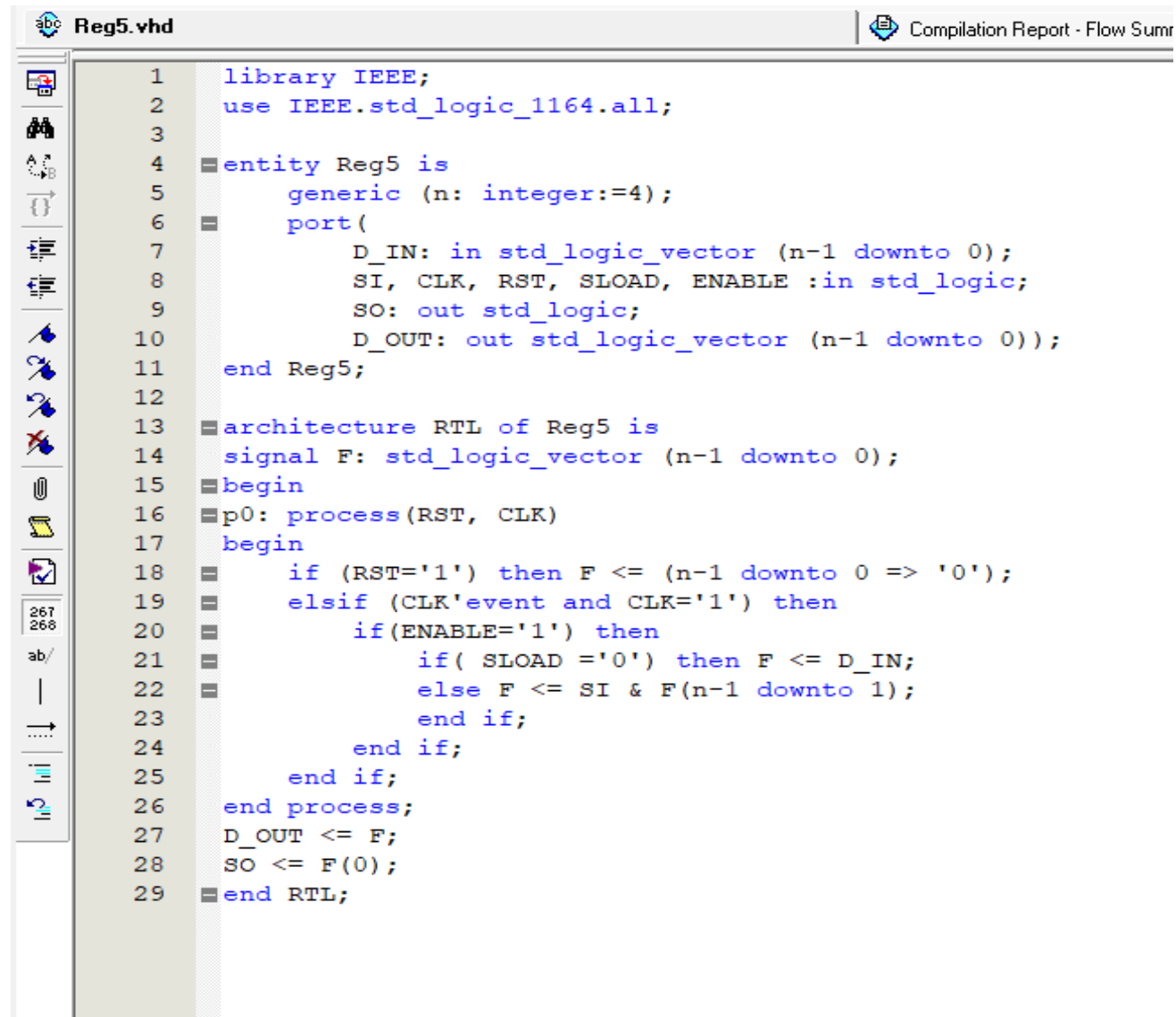


Quartus 5

Μύρων Κουφόπουλος 4398
Χρήστος Καραγιαννίδης 4375



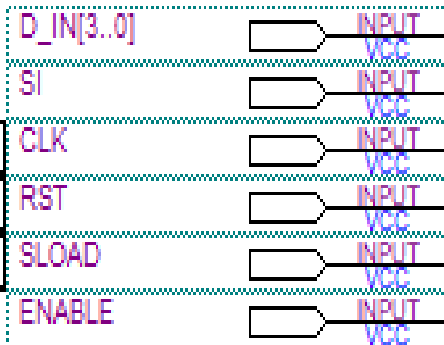
Σχεδίαση Καταχωρητή



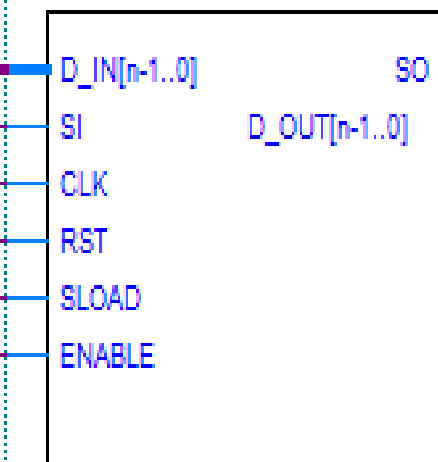
```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity Reg5 is
5      generic (n: integer:=4);
6  port(
7      D_IN: in std_logic_vector (n-1 downto 0);
8      SI, CLK, RST, SLOAD, ENABLE :in std_logic;
9      SO: out std_logic;
10     D_OUT: out std_logic_vector (n-1 downto 0));
11 end Reg5;
12
13 architecture RTL of Reg5 is
14     signal F: std_logic_vector (n-1 downto 0);
15 begin
16     p0: process(RST, CLK)
17     begin
18         if (RST='1') then F <= (n-1 downto 0 => '0');
19         elsif (CLK'event and CLK='1') then
20             if(ENABLE='1') then
21                 if( SLOAD ='0') then F <= D_IN;
22                 else F <= SI & F(n-1 downto 1);
23             end if;
24         end if;
25     end if;
26 end process;
27 D_OUT <= F;
28 SO <= F(0);
29 end RTL;
```



PIN_N26
PIN_G26
PIN_N25

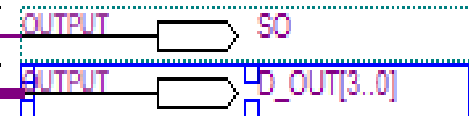


Reg5



inst

Parameter	Value	Type
n	4	Signed Integer

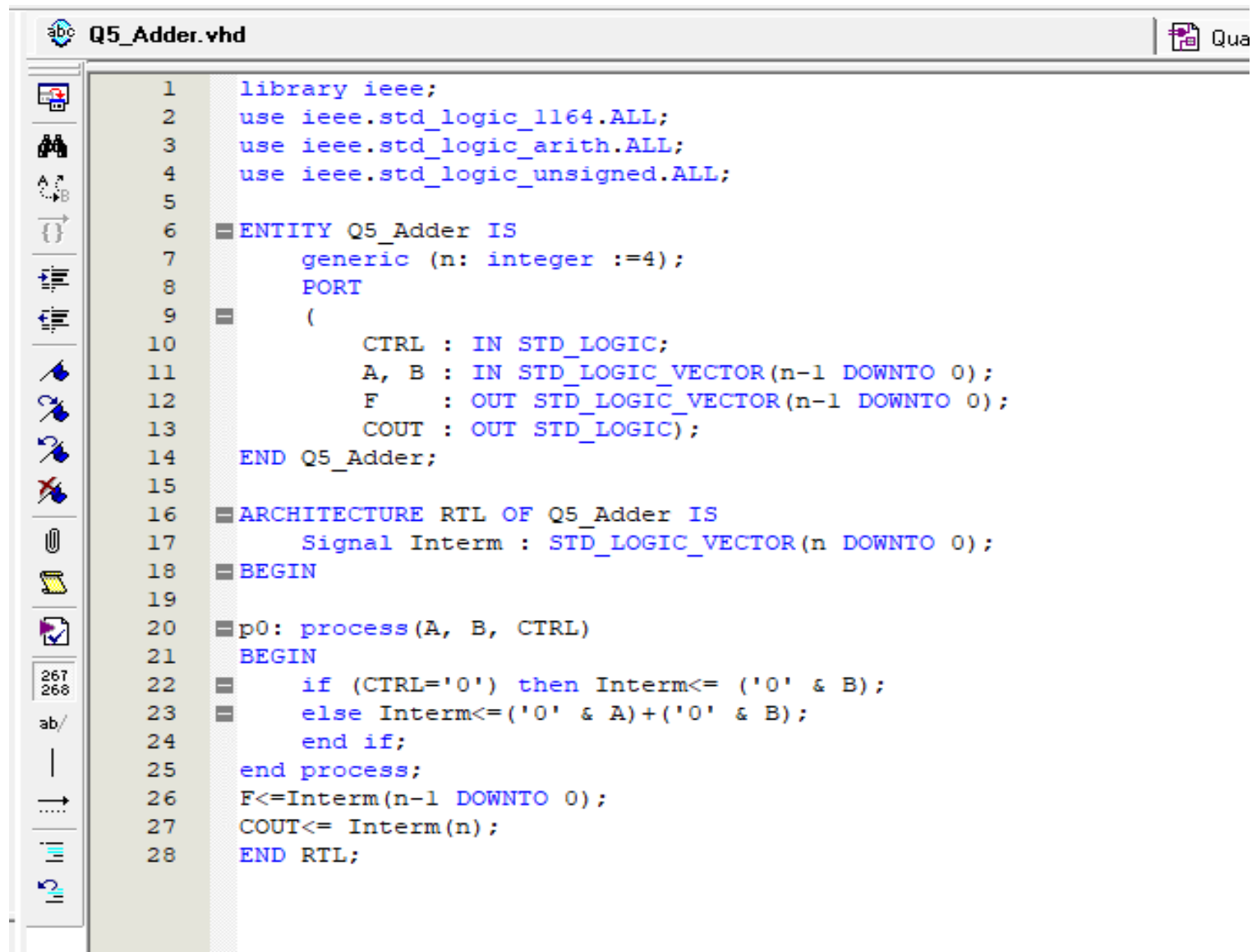


Μέχρι τα πρώτα 30 ns δεν έχουμε κάποια αλλαγή στο κύκλωμα αφού έχουμε το Rst στην μονάδα για να **αρχικοποιήσει** το κύκλωμα μας στο '0000' από εκεί και μετά παραμένει στο 0.

Από τα 30 ns - 160 ns το SLOAD είναι στο 0 οπότε ο καταχωρητής κάνει **παράλληλη φόρτωση** που το παρατηρούμε στο D_OUT. Ωστόσο στην γραμμική περίοδο 90-120 ns, το ENABLE είναι στο 0 οπότε το κύκλωμα αγνοεί την θετική ακμή του ρολογιού στα 100 ns και η παράλληλη φόρτωση της τιμής '0111' δεν γίνεται μέχρι τα 140 ns.

Από τα 160 ns και μετά το SLOAD είναι μόνιμα στο 1 οπότε γίνεται **σειριακή φόρτωση** του 0 ή του 1 αναλόγως με την τιμή του SI (π.χ. στα 180 ns έχουμε το SLOAD = 1 και το SI = 1 και έχουμε θετική ακμή ρολογιού άρα το '0111' θα γίνει '1011'.

Σχεδίαση Αθροιστή



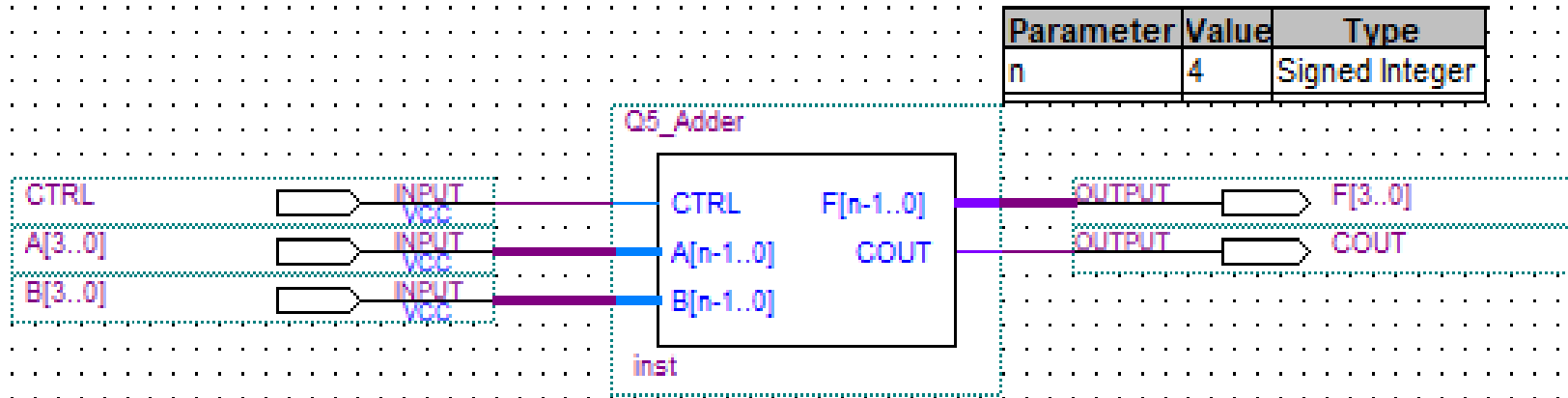
```
1  library ieee;
2  use ieee.std_logic_1164.ALL;
3  use ieee.std_logic_arith.ALL;
4  use ieee.std_logic_unsigned.ALL;
5
6  ENTITY Q5_Adder IS
7      generic (n: integer :=4);
8      PORT
9      (
10         CTRL : IN STD_LOGIC;
11         A, B : IN STD_LOGIC_VECTOR(n-1 DOWNT0 0);
12         F      : OUT STD_LOGIC_VECTOR(n-1 DOWNT0 0);
13         COUT : OUT STD_LOGIC);
14  END Q5_Adder;
15
16  ARCHITECTURE RTL OF Q5_Adder IS
17      Signal Interm : STD_LOGIC_VECTOR(n DOWNT0 0);
18  BEGIN
19
20  p0: process(A, B, CTRL)
21  BEGIN
22      if (CTRL='0') then Interm<= ('0' & B);
23      else Interm<=('0' & A)+('0' & B);
24      end if;
25  end process;
26  F<=Interm(n-1 DOWNT0 0);
27  COUT<= Interm(n);
28  END RTL;
```



Q5_Adder.vhd



Quartus

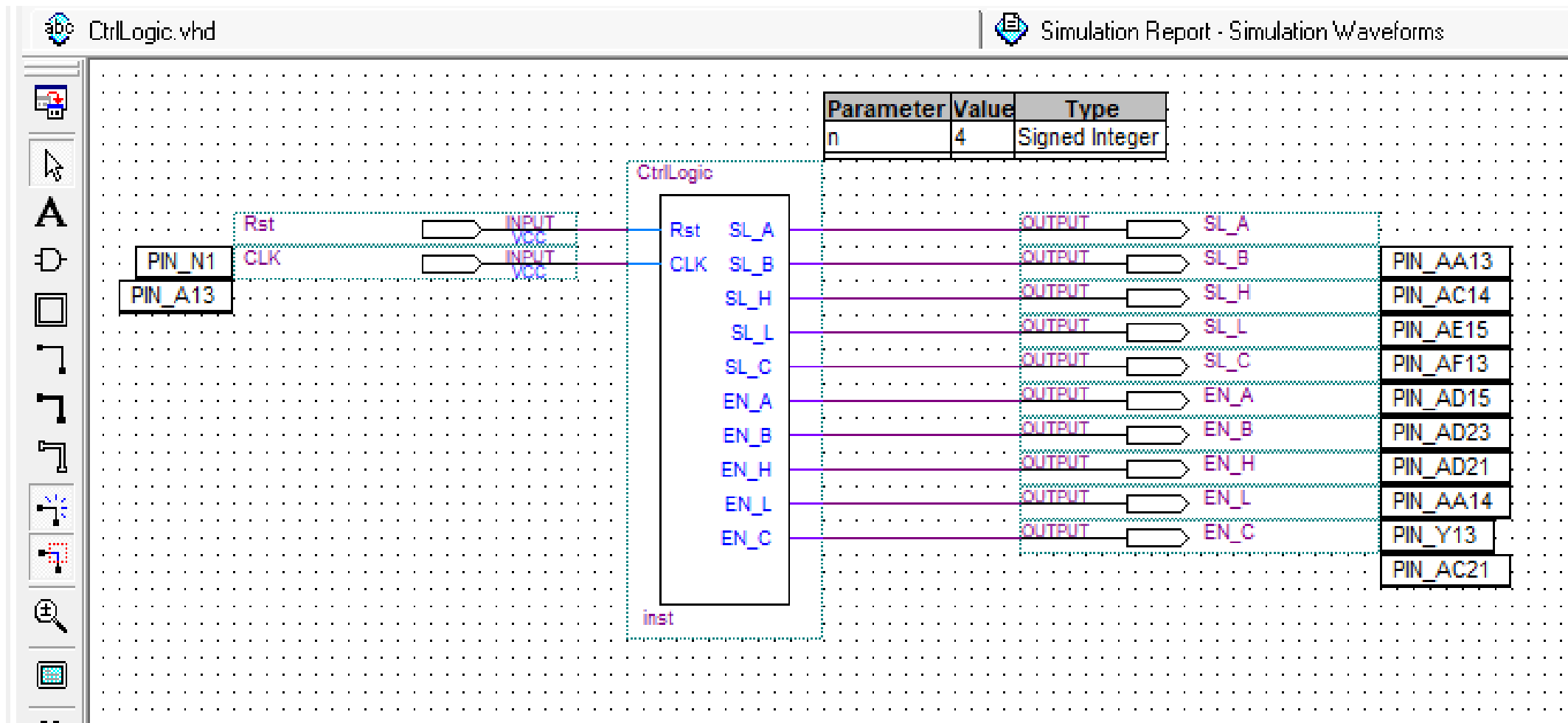


Registered Performance tpd tsu tco th Custom Delays

	Slack	Required P2P Time	Actual P2P Time	From	To	
1	N/A	None	14.126 ns	B[2]	F[3]	
2	N/A	None	14.046 ns	A[2]	F[3]	
3	N/A	None	13.277 ns	B[2]	F[2]	
4	N/A	None	13.200 ns	A[2]	F[2]	
5	N/A	None	13.087 ns	A[1]	F[3]	
6	N/A	None	12.984 ns	A[0]	F[3]	
7	N/A	None	12.891 ns	B[0]	F[3]	
8	N/A	None	12.844 ns	B[1]	F[3]	
9	N/A	None	12.553 ns	A[1]	F[2]	
10	N/A	None	12.450 ns	A[0]	F[2]	
11	N/A	None	12.357 ns	B[0]	F[2]	
12	N/A	None	12.317 ns	A[3]	F[3]	
13	N/A	None	12.310 ns	B[1]	F[2]	
14	N/A	None	12.260 ns	B[2]	COUT	
15	N/A	None	12.180 ns	A[2]	COUT	
16	N/A	None	11.716 ns	A[0]	F[0]	
17	N/A	None	11.627 ns	B[0]	F[0]	
18	N/A	None	11.221 ns	A[1]	COUT	
19	N/A	None	11.118 ns	A[0]	COUT	
20	N/A	None	11.025 ns	B[0]	COUT	
21	N/A	None	10.978 ns	B[1]	COUT	
22	N/A	None	10.819 ns	A[0]	F[1]	
23	N/A	None	10.767 ns	A[3]	COUT	
24	N/A	None	10.726 ns	B[0]	F[1]	
25	N/A	None	10.607 ns	A[1]	F[1]	
26	N/A	None	10.363 ns	B[1]	F[1]	
27	N/A	None	8.607 ns	B[3]	F[3]	
28	N/A	None	7.776 ns	CTRL	F[3]	
29	N/A	None	7.348 ns	CTRL	F[0]	
30	N/A	None	7.274 ns	CTRL	F[2]	
31	N/A	None	7.056 ns	B[3]	COUT	
32	N/A	None	6.152 ns	CTRL	COUT	
33	N/A	None	5.925 ns	CTRL	F[1]	

Η καθυστέρηση του αθροιστή είναι
14.126 ns

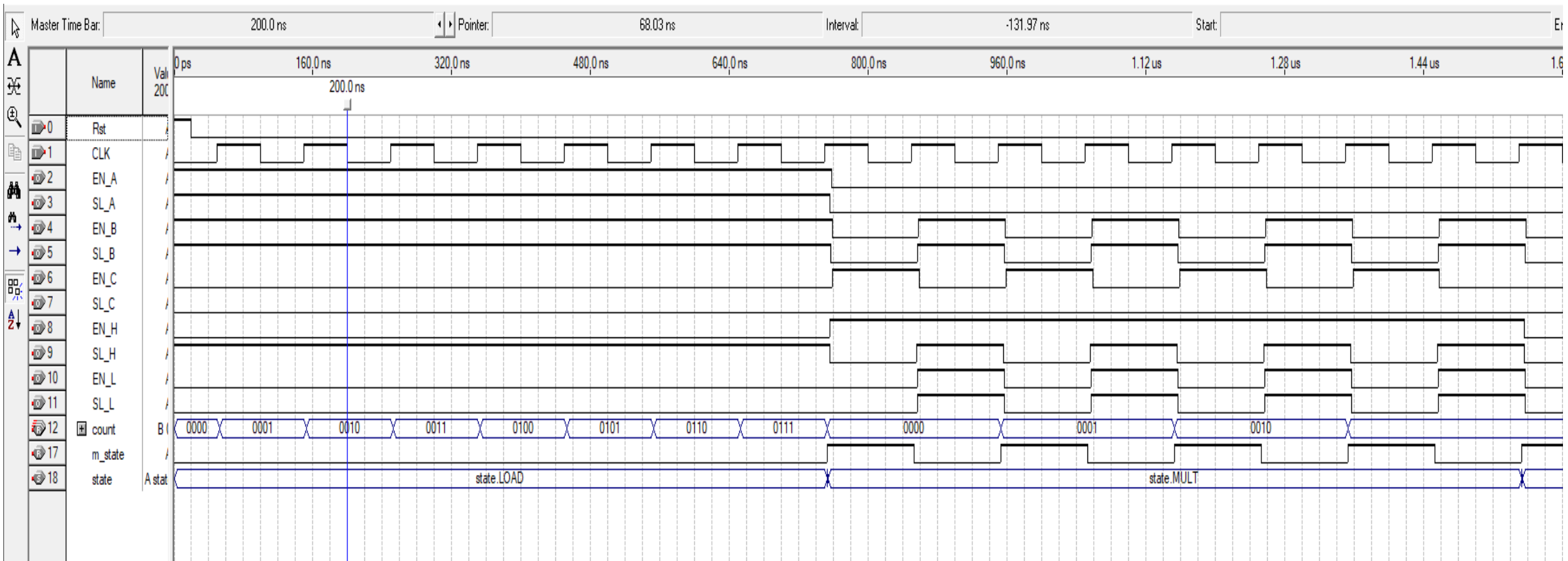
Σχεδίαση Μονάδας Ελέγχου



```

1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use ieee.std_logic_arith.ALL;
4  use ieee.std_logic_unsigned.ALL;
5
6  entity CtrlLogic is
7      generic ( n: integer := 4 );
8      port (
9          Rst, CLK : in std_logic;
10         SL_A, SL_B, SL_H, SL_L, SL_C: out std_logic;
11         EN_A, EN_B, EN_H, EN_L, EN_C: out std_logic );
12 end CtrlLogic;
13
14 architecture ControlLGC of CtrlLogic is
15     type state_type is (LOAD, MULT, HOLD);
16     type mult_type is (SHIFT, ADD);
17     signal state : state_type;
18     signal m_state : mult_type;
19     signal count : std_logic_vector (n-1 downto 0);
20
21 begin
22     p0: process(Rst, CLK)
23     begin
24         if (Rst='1') then
25             state<=LOAD;
26             count<=(n-1 downto 0 => '0');
27         elsif (CLK'event and CLK='1') then
28             case state is
29                 when LOAD => if (conv_integer(count)<2*n-1) then count<=count+1;
30                             else count<=(n-1 downto 0 => '0');
31                             state<=MULT; m_state<=ADD;
32                             end if;
33                 when MULT => if (m_state=ADD) then m_state<=SHIFT;
34                             elsif (m_state=SHIFT) then
35                                 m_state<=ADD;
36                                 if (conv_integer(count)<n-1) then count<=count+1;
37                                 else state<=HOLD;
38                                 end if;
39                             end if;
40                 when HOLD => null;
41             end case;
42         end if;
43     end process;
44     EN_A<= '1' when (state = LOAD) else '0';
45     SL_A<= '1' when (state = LOAD) else '0';
46     EN_B<= '1' when ((state = LOAD) or ((state = MULT) and m_state = shift)) else '0';
47     SL_B<= '1' when ((state = LOAD) or ((state = MULT) and m_state = shift)) else '0';
48     EN_H<= '1' when (state = MULT) else '0';
49     SL_H<= '1' when (m_state = shift) else '0';
50     EN_L<= '1' when ((state = MULT) and m_state = shift) else '0';
51     SL_L<= '1' when ((state = MULT) and m_state = shift) else '0';
52     EN_C<= '1' when ((state = MULT) and m_state = ADD) else '0';
53     SL_C<= '0';
54 end ControlLGC;

```



Η κάθε διακεκομμένη γραμμή είναι 40 ns.

Στα πρώτα 20 ns το Rst βρίσκεται στην μονάδα οπότε αρχικοποιείται το κύκλωμα μας.

Το SL_C βρίσκεται πάντα στο '0' γιατί δεν θέλουμε να κάνουμε ολίσθηση τον καταχωρητή C του κρατουμένου.

Ένας κύκλος του ρολογιού είναι 100 ns.

Μέχρι τα 760 ns το κύκλωμα βρίσκεται στην κατάσταση load όπου η μεταβλητή count πηγαίνει από το '0000' μέχρι το '0111' δηλαδή το 2^n-1 . Όπου και αλλάζει το state σε mult. Παράλληλα τα EN_A, EN_B, SL_A και SL_B παίρνουν την τιμή '1' όπως προβλέπεται από τις αντίστοιχες εντολές (π.χ. `EN_A<='1' when state=LOAD else '0';`). [παρατηρούμε ότι και το SL_H παίρνει την τιμή '1' όμως αυτό γίνεται λόγω του ότι το m_state='0' το οποίο δεν μας επηρεάζει αυτή την στιγμή καθώς δεν βρισκόμαστε στο stage MULT]

Από το 760-1560 ns βρισκόμαστε στο state.MULT.

Όσο βρισκόμαστε στο state.MULT η τιμή του m_state και οι αλλαγές της καθορίζουν τις τιμές των υπόλοιπων εξόδων.

Όταν το m_state= '1' : EN_C= '1' και EN_H= '1' και οι υπόλοιποι έξοδοι στο '0'

Όταν το m_state= '0' : EN_B='1', SL_B='1', EN_H='1', SL_H='1' ,EN_L= '1'
,SL_L='1' και οι υπόλοιποι έξοδοι στο '0'.

Κάθε φορά που το m_state πάει από '0' σε '1' η τιμή του count αυξάνεται κατά 1 όσο βρισκόμαστε state.MULT (στο 1560 ns σταματάει η αύξηση αυτή γιατί μπαίνουμε στο state.hold).

Παρατηρούμε ότι αυτό το μοτίβο επαναλαμβάνεται σε κάθε θετική ακμή του ρολογιού όσο βρισκόμαστε state.MULT.

Από τα 1560 ns έως το Rst='1' βρισκόμαστε στο state.hold.


Όσο είμαστε στο state.hold όλες οι τιμές εξόδου είναι '0' (when HOLD => null;).

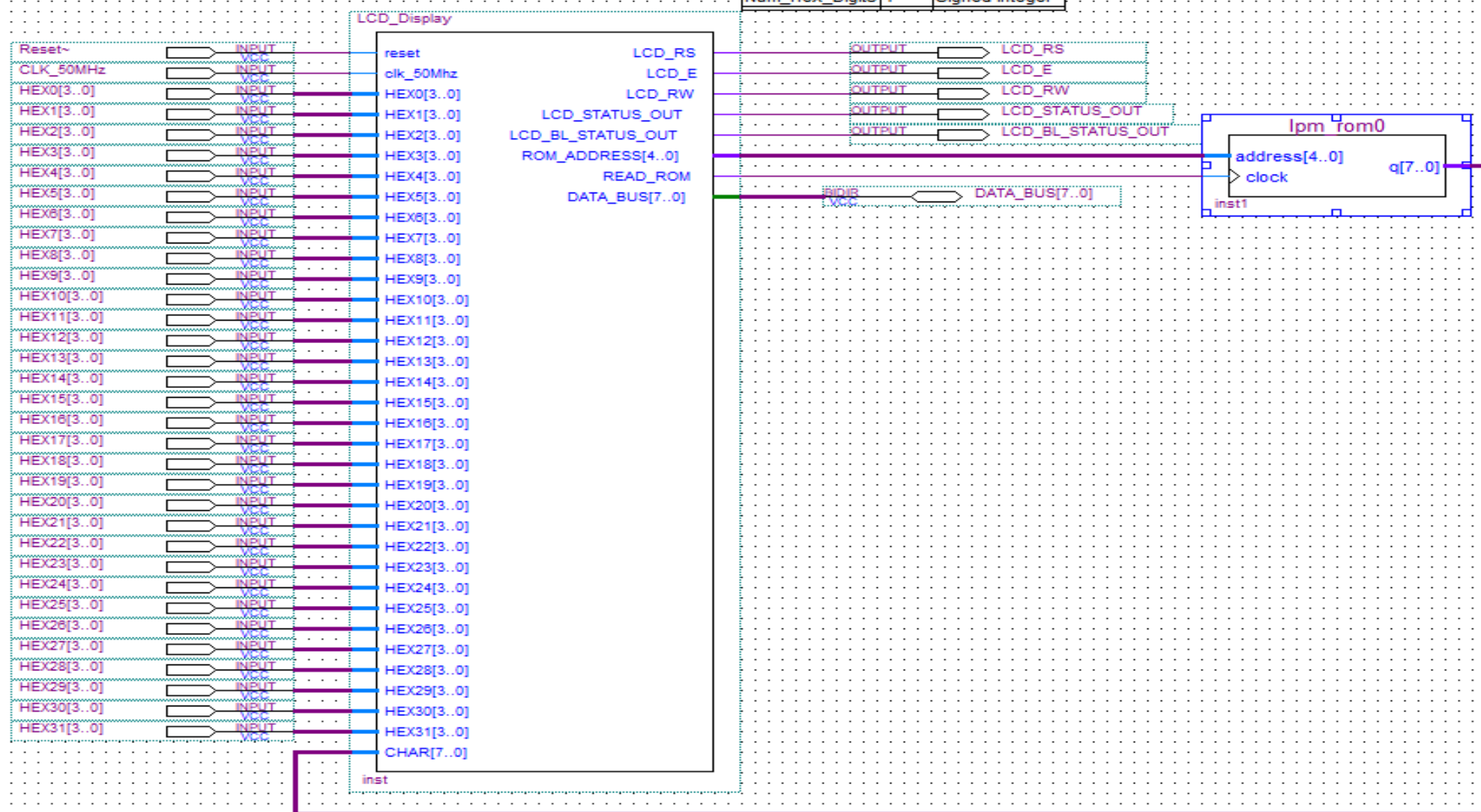
Σχεδίαση Πολλαπλασιαστή

Εισαγωγή δεδομένων σε κάθε κελί της ROM.

Τα 0,1,2,3 τα αφήνουμε 00 θα βάλουμε τις εισόδους μας σε αυτά.

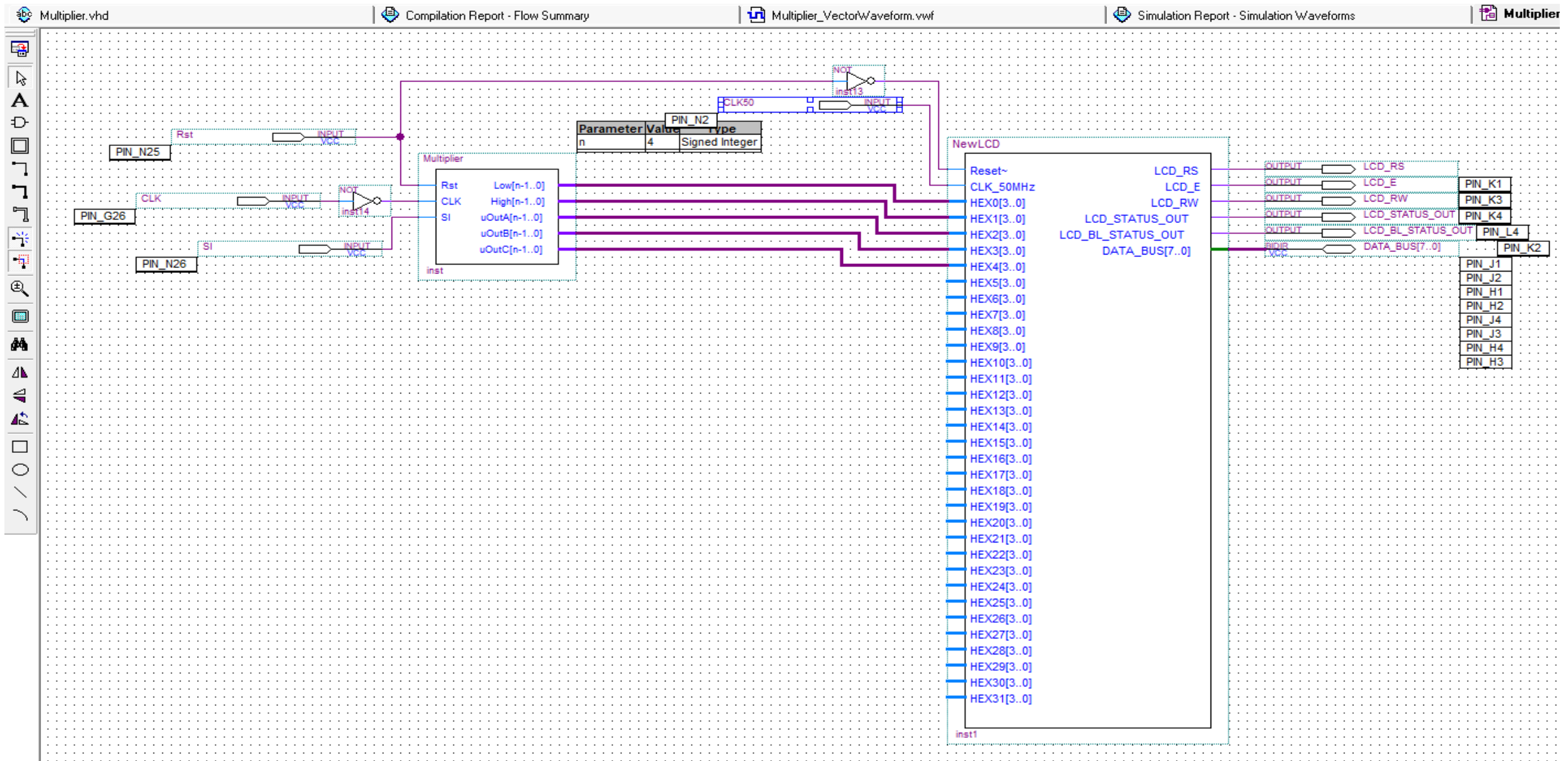
Στα υπόλοιπα βάζουμε τον αριθμό 20 για να έχουν κενό χαρακτήρα.

 Multiplier.vhd								
Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	00	00	00	00	00	20	20	20
8	20	20	20	20	20	20	20	20
16	20	20	20	20	20	20	20	20
24	20	20	20	20	20	20	20	20




```
Multiplier.vhd
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use ieee.std_logic_arith.ALL;
4  use ieee.std_logic_unsigned.ALL;
5
6  entity Multiplier is
7      generic (n: integer :=4);
8  port (
9      Rst, CLK, SI : in std_logic;
10     Low, High : out std_logic_vector (n-1 downto 0);
11     uOutA, uOutB, uOutC : out std_logic_vector (n-1 downto 0));
12 end Multiplier;
13
14 architecture Multip of Multiplier is
15
16     component CtrlLogic
17         generic ( n: integer := 4 );
18         port ( Rst, CLK : in std_logic;
19             SL_A, SL_B, SL_H, SL_L, SL_C : out std_logic;
20             EN_A, EN_B, EN_H, EN_L, EN_C : out std_logic );
21     end component;
22
23     component Q5_Adder
24         generic (n: integer :=4);
25         port ( CTRL : in std_logic;
26             A, B : in std_logic_vector (n-1 downto 0);
27             F : out std_logic_vector (n-1 downto 0);
28             COUT : out std_logic );
29     end component;
30
31     component Reg5
32         generic (n: integer:=4);
33         port ( D_IN: in std_logic_vector (n-1 downto 0);
34             SI, CLK, RST, SLOAD, ENABLE: in std_logic;
35             SO: out std_logic;
36             D_OUT: out std_logic_vector (n-1 downto 0));
37     end component;
38
39     signal SL_A, EN_A, SO_A, SL_B, EN_B, SO_B, SL_C, EN_C, SO_C : std_logic;
40     signal SL_H, EN_H, SO_H, SL_L, EN_L, SO_L : std_logic;
41     signal A,B,F_ADD, H: std_logic_vector(n-1 downto 0);
42     signal C,COUT :std_logic_vector (0 downto 0);
43
44 begin
45     uA: Reg5 generic map(n=>4)
46     port map(D_IN=>(n-1 downto 0=>'0'), SI=>SI, CLK=>CLK, RST=>RST, SLOAD=> SL_A,
47             ENABLE=>EN_A,SO=>SO_A,D_OUT=>A);
48     uB: Reg5 generic map(n=>4)
49     port map(D_IN=>(n-1 downto 0=>'0'), SI=>SO_A, CLK=>CLK, RST=>RST, SLOAD=>SL_B,
50             ENABLE=>EN_B, SO=>SO_B, D_OUT=>B);
51     uAdder: Q5_Adder generic map (n=>4)
52     port map (CTRL=>B(0), A=>A, B=>H, F=>F_ADD, COUT=>COUT(0));
53     uC: Reg5 generic map (n=>1)
54     port map (D_IN=>COUT, SI=>'0', CLK=>CLK, RST=>RST, SLOAD=>SL_C, ENABLE=>EN_C,
55             SO=>SO_C, D_OUT=>C);
56     uH: Reg5 generic map (n=>4)
57     port map(D_IN=>F_ADD, SI=>SO_C, CLK=>CLK, RST=>RST, SLOAD=>SL_H, ENABLE=>EN_H,
58             SO=>SO_H, D_OUT=>H);
59     uL: Reg5 generic map(n=>4)
60     port map(D_IN=>(n-1 downto 0=>'0'), SI=>SO_H, CLK=>CLK, RST=>RST, SLOAD=>SL_L,
61             ENABLE=>EN_L, SO=>SO_L, D_OUT=>Low);
62     uCTRL:CtrlLogic generic map(n=>4)
63     port map(Rst=>RST, CLK=>CLK, SL_A=>SL_A, SL_B=>SL_B, SL_H=>SL_H, SL_L=>SL_L,
64             SL_C=>SL_C, EN_A=>EN_A, EN_B=>EN_B, EN_H=>EN_H, EN_L=>EN_L, EN_C=>EN_C);
65
66     High<=H;
67     uOutA<= A;
68     uOutB<= B;
69     uOutC<= "000" & C;
70 end Multip;
```

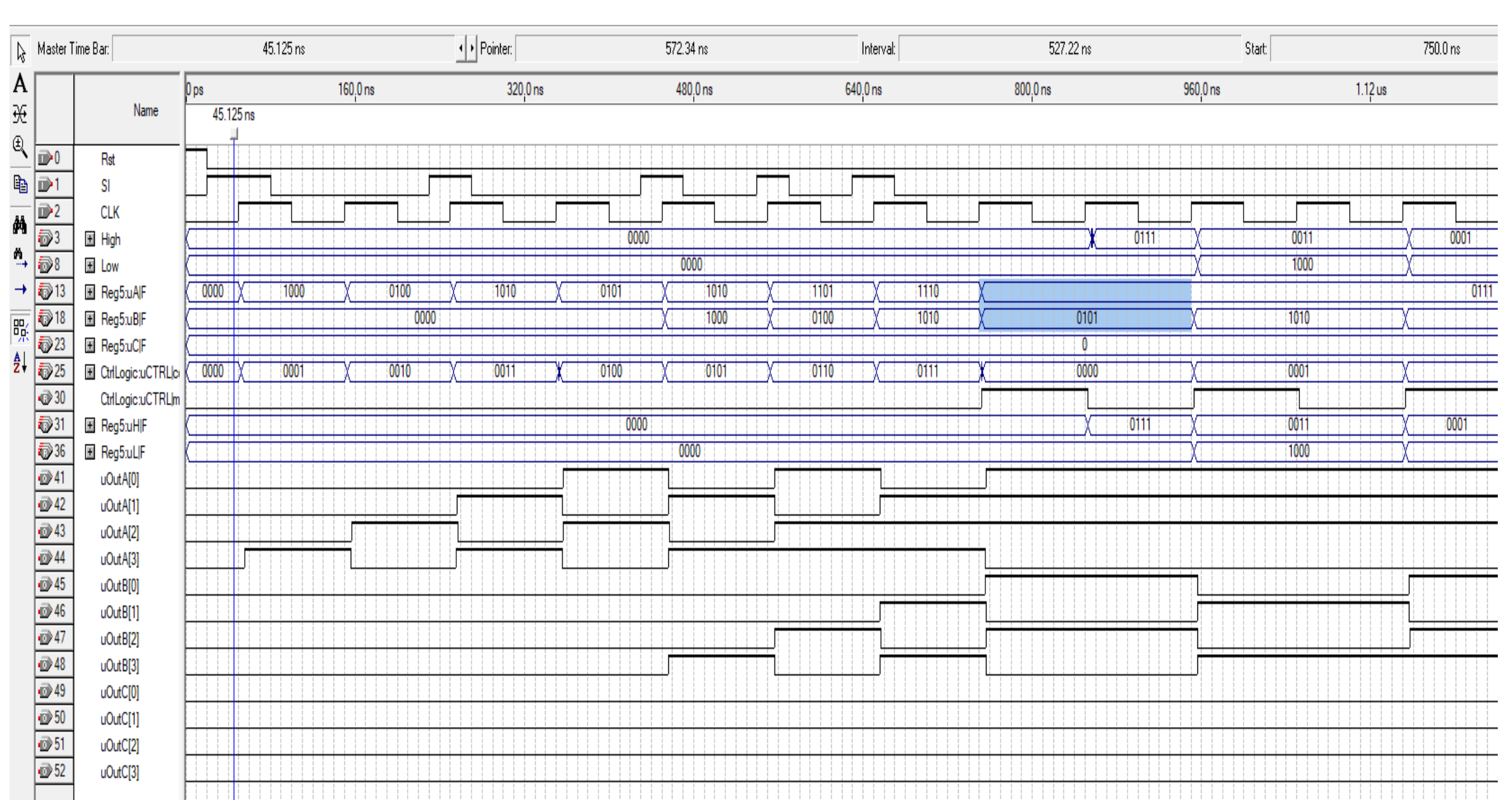
Στο κώδικα έχουν προστεθεί 3 δικές μας έξοδοι καθώς μας ζητείται στην υλοποίηση του πολλαπλασιαστή να απεικονίσουμε τα περιεχόμενα των καταχωρητών uA, uB, uC .



Στο CLK θα βάλουμε not γιατί όταν πιέζουμε τα pushbutton θέλουμε να λαμβάνει χώρα η θετική ακμή.

Αυτή η διαφοροποίηση υπάρχει γιατί δεν χρειαζόμαστε την τιμή του low κάπου αλλού εκτός από το τελικό αποτέλεσμα δεν επηρεάζει το τελικό αποτέλεσμα σε κάθε κύκλο του ρολογιού.

Αντίθετα η τιμή της HIGH μας είναι απαραίτητη σε κάθε κύκλο του ρολογιού καθώς επηρεάζει τον πολλαπλασιασμό.



Σειριακή Ανάθεση Τιμών

Η κάθε διακεκομμένη γραμμή στο παραπάνω vector waveform είναι 10 ns.

Το Rst='1' για τα πρώτα 20 ns για να αρχικοποιήσει τις τιμές.

Σε κάθε θετικό κύκλο του ρολογιού φορτώνεται ως MSB(most significant bit) του καταχωρητή A η τιμή του SI και όλα τα υπόλοιπα ψηφία του πηγαίνουν μία θέση δεξιά κάνοντας το LSB του A, MSB του B(πχ 0001 0000 -> 0000 1000).

Σύμφωνα με τα παραπάνω κατά την σειριακή ανάθεση τιμής η τιμή που θα εισαχθεί στο πρώτο θετικό χτύπο του ρολογιού(από το SI στο καταχωρητή A) θα είναι και η πρώτη τιμή που θα φτάσει στο LSB του καταχωρητή B μετά από 8 κύκλους του ρολογιού.

Εμείς θέλουμε να εισάγουμε A=7(0111) και B=5(0101) οπότε οι αριθμοί θα πρέπει να εισαχθούν με την σειρά 1,0,1,0,1,1,1,0 σε κάθε διαδοχική θετική ακμή του ρολογιού(το SI βρίσκεται στις αντίστοιχες τιμές στις χρονικές στιγμές 50,150,250,350,450,550,650,750 ns)

Οπότε οι καταχωρητές A και B θα έχουν τις ζητούμενες τιμές την χρονική περίοδο 750-950 ns.

Το αποτέλεσμα του πολλαπλασιασμού $7*5=35$ (00100011) εμφανίζεται στους καταχωρητές High='0010' και Low='0011' μετά την χρονική στιγμή 1550 ns. Αυτό συμβαίνει μετά από 2η θετικές ακμές του ρολογιού από την στιγμή που φορτώθηκαν οι τιμές των καταχωρητών A='0111' και B='0101'. (όπου $n=4$ άρα 8 θετικές ακμές), οι οποίες είναι 850,950,1050,1150,1250,1350,1450,1550 ns

Το κρατούμενο C παραμένει '0' καθόλη την διάρκεια της εκτέλεσης αφού καμία φορά το αποτέλεσμα μας δεν θα έχει υπερχείλιση (υπάρχουν άλλες αρχικές τιμές των καταχωρητών A και B όπου θα έκαναν το C='1').

Το uCtrl|Count μετράει από το 0-7 όταν βρισκόμαστε στο state.load στα 750 ns θα σταματήσει το state.load και θα ξεκινήσει το state.mult. Όπου τα uCtrl|Count και uCtrl|m_state αλλάζουν με βάση όσα αναφέραμε στο Control Unit.

800.0 ns

960.0 ns

1.12 μ s1.28 μ s1.44 μ s1.6 μ s

0111

0011

0001

1000

0100

0010

1000

1100

0110

0011

0111

0101

1010

1101

1110

1111

0

0000

0001

0010

0011

0111

0011

0001

1000

0100

0010

1000

1100

0110

0011