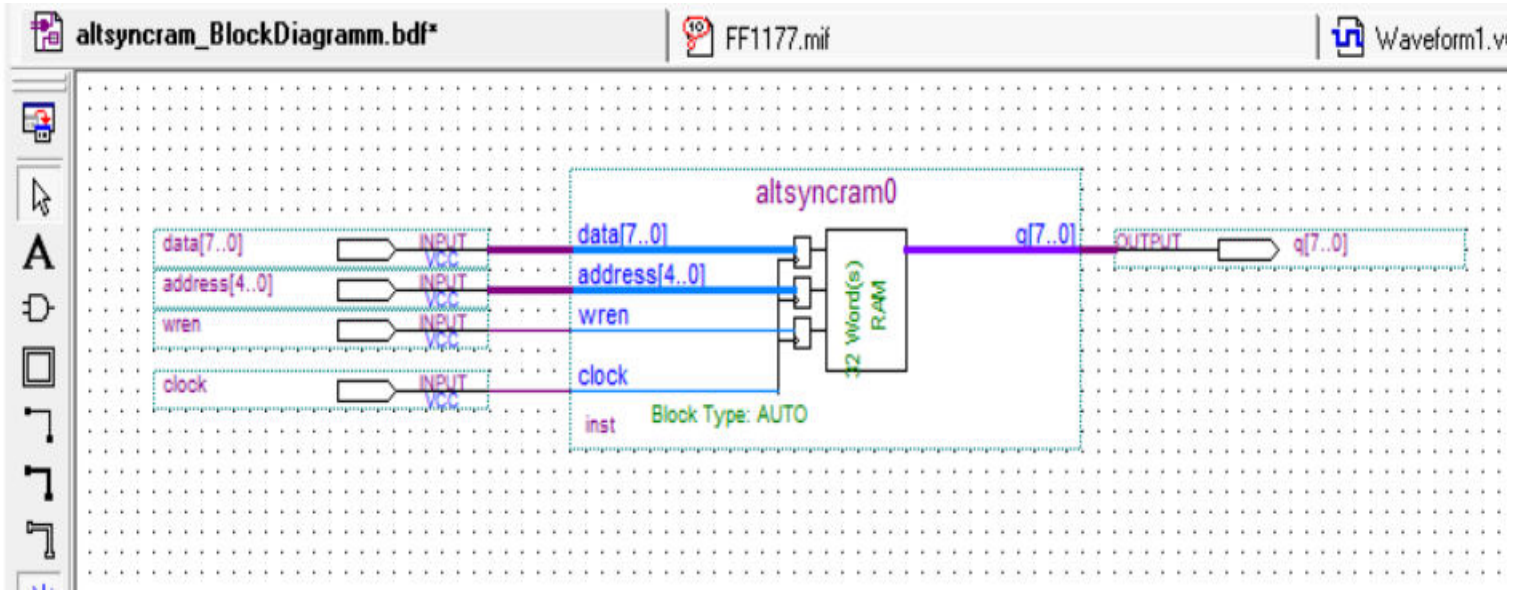
Quartus 6

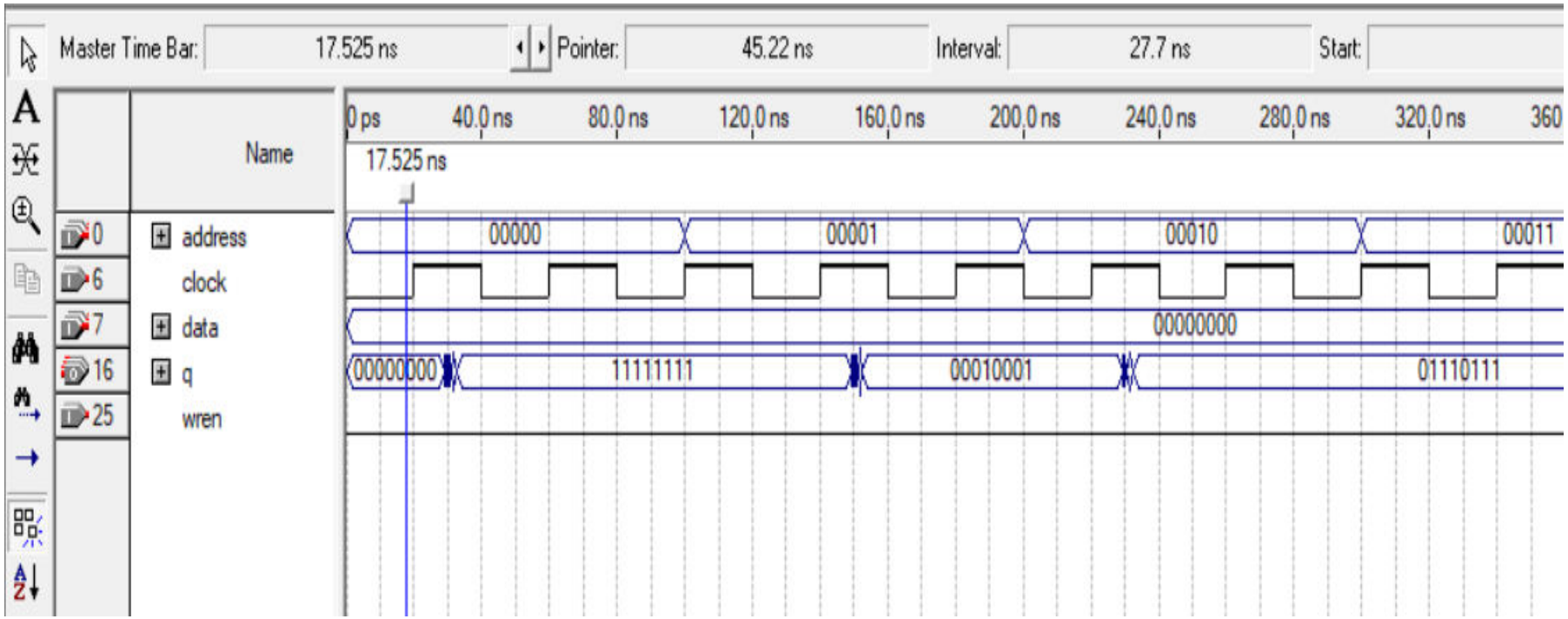
Καραγιαννίδης Χρήστος 4375
Κουφόπουλος Μύρωνας 4398

Χρησιμοποιώντας την Μνήμη

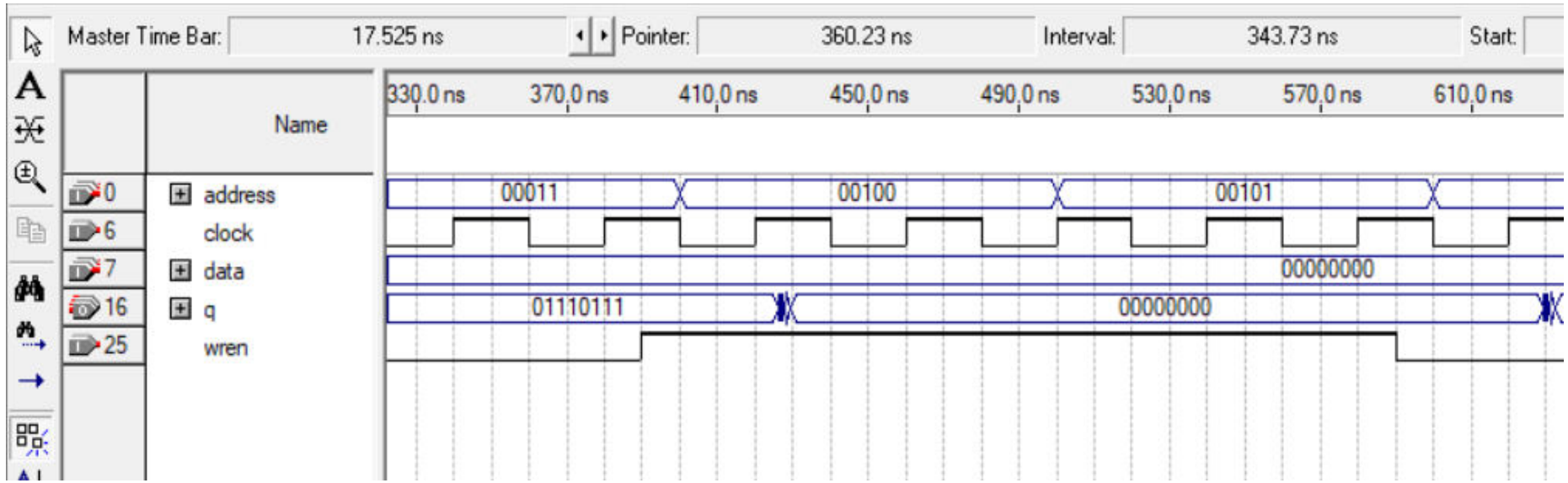


FF1177.mif

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	FF	11	77	77	77	77	77	77
8	77	77	77	77	77	77	77	77
16	77	77	77	77	77	77	77	77
24	77	77	77	77	77	77	77	77



Η έξοδος q της μνήμης μέχρι τον πρώτο θετικό χτύπο του ρολογιού έχει τιμή 00000000 γιατι ακόμα δεν της έχει υποδειχθεί κάποιο address. Στον πρώτο θετικό χτύπο φορτώνει τα δεδομένα που βρίσκονται στην address 00000 δηλαδή τον δεκαεξαδικό αριθμό FF (που είναι ίσο με 11111111 σε δυαδικό). Στα 140ns φορτώνει την τιμή του επόμενου address(00001) η οποία είναι 11_{16} =00010001_2 . Έπειτα για οποιαδήποτε άλλη τιμή του address φορτώνει το 77_{16} = 01110111_2 .Τα παραπάνω ισχύουν όσο το wren=0.

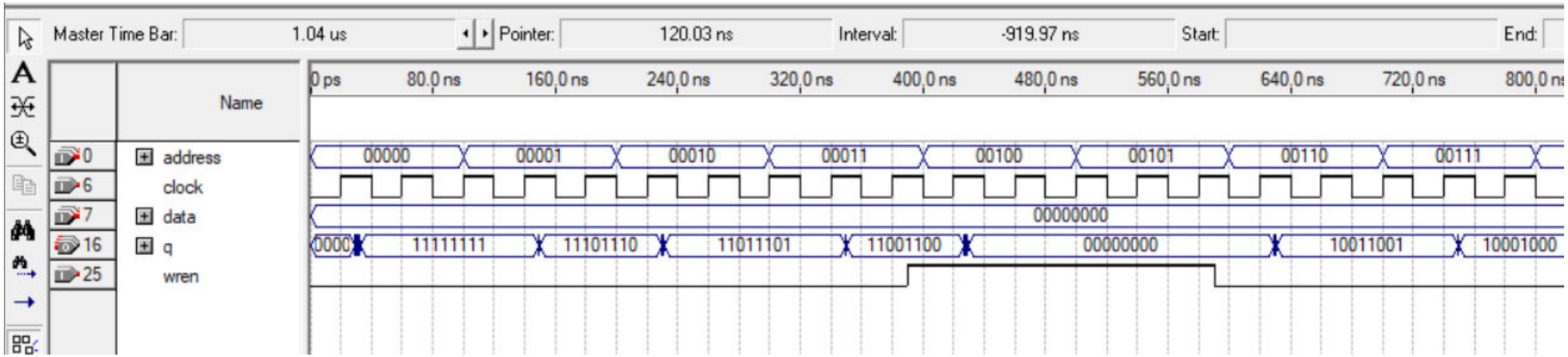


Όσο το wren=1 φορτώνεται στην εκάστοτε τιμή του address η τιμή που έχει εκείνη τη στιγμή το data.

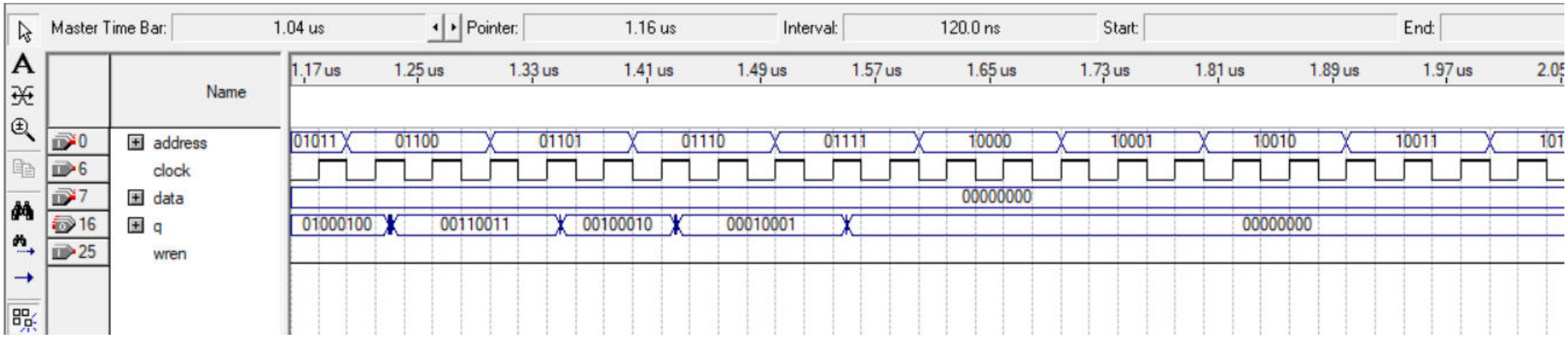
FFEEDDCCBBAA99.mif

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	FF	EE	DD	CC	BB	AA	99	88
8	77	66	55	44	33	22	11	00
16	00	00	00	00	00	00	00	00
24	00	00	00	00	00	00	00	00

Αλλάξαμε στη μνήμη το mif αρχείο που είχε και βάλαμε το παραπάνω.

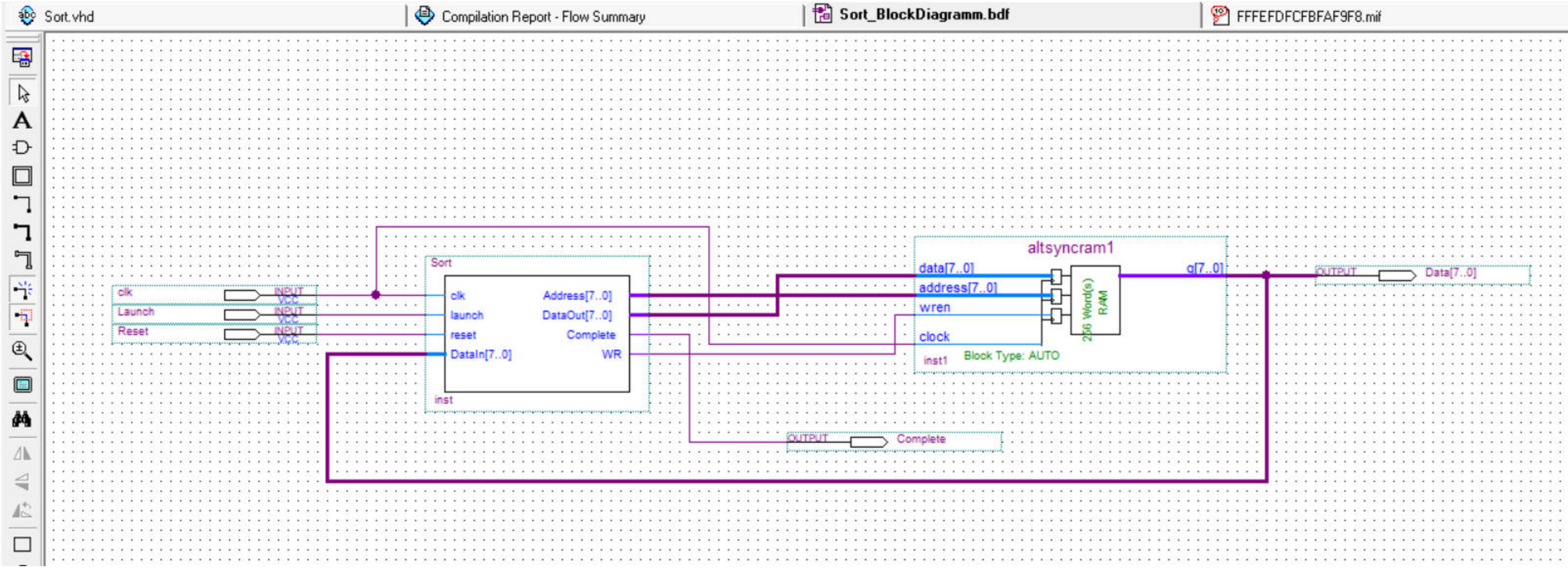


Λειτουργεί αντίστοιχα φορτώνοντας τις τιμές που έχει η κάθε θέση μνήμης και οταν το wren=1 φορτώνει την τιμή του data.



Στο παραπάνω vector waveform παρατηρούμε οτι φορτώνει τις τιμες 44 33 22 11 και μετά έχει συνέχεια το 00_{16} συμφώνα με το καινούριο mif αρχείο.

Σχεδιαση με VHDL

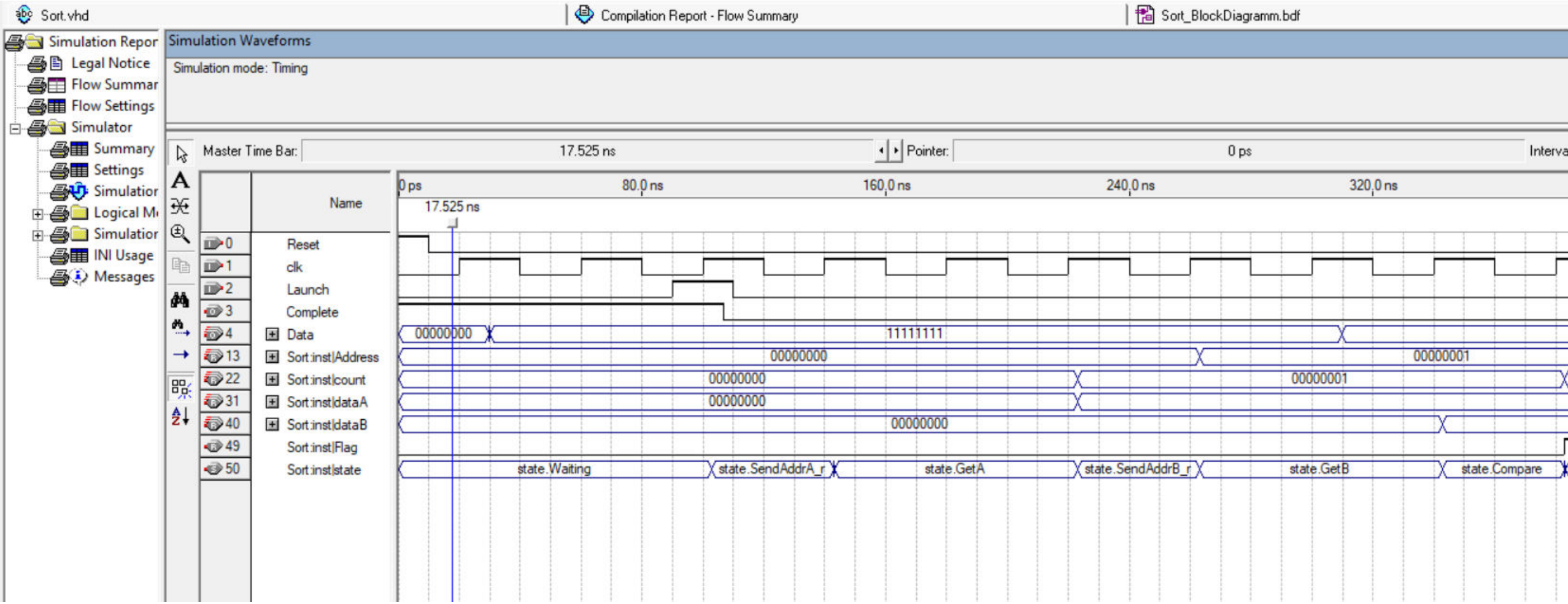


Sort.vhd

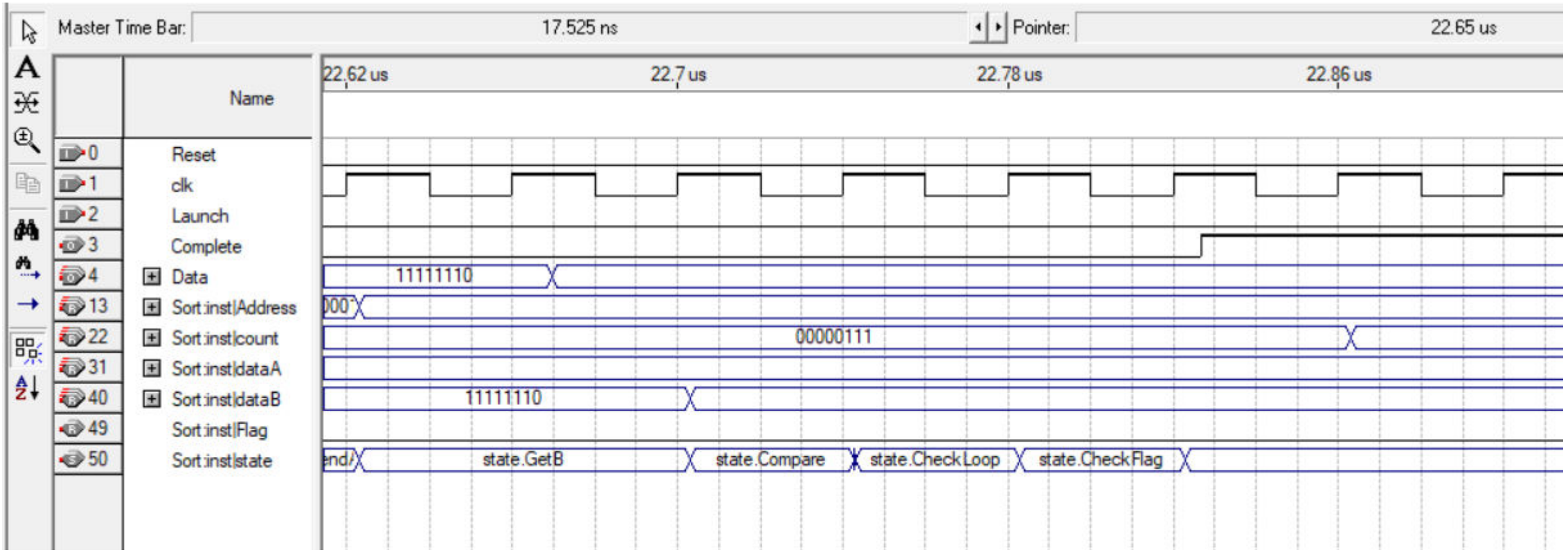
```
1  LIBRARY IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_unsigned.all;
4
5  ENTITY Sort IS
6      PORT (
7          clk,launch,reset : in std_logic;
8          DataIn : in std_logic_vector(7 downto 0);
9          Address,DataOut : out std_logic_vector(7 downto 0);
10         Complete, WR : out std_logic);
11 END Sort;
12
13 ARCHITECTURE RTL OF Sort IS
14         type state_type is (Waiting,SendAddrA_r,GetA,SendAddrB_r,GetB,Compare,SendAddrA_w,WriteA,SendAddrB_w,WriteB,CheckLoop,CheckFlag);
15         SIGNAL state : state_type:=Waiting;
16         signal count : std_logic_vector(7 downto 0);
17         signal delay : std_logic;
18         signal dataA, dataB: std_logic_vector(7 downto 0);
19         signal Flag,Swap,CountEnd : std_logic;
20 BEGIN
21     process(clk,reset)
22     begin
23         if (reset ='1') then
24             state <= Waiting;
25         elsif(clk'event and clk='1') then
26             case state is
27                 when Waiting => if (launch='1') then state <= SendAddrA_r; end if;
28                 when SendAddrA_r => state <=GetA; delay<='0';
29                 when GetA => if (delay='1') then state <= SendAddrB_r; else delay<='1' ; end if;
30                 when SendAddrB_r => state <=GetB; delay<='0';
31                 when GetB => if (delay='1') then state <=Compare; else delay<='1'; end if;
32                 when Compare => if Swap='1' then state<=SendAddrB_w; else state<=CheckLoop; end if;
33                 when SendAddrB_w => state <=WriteB;
34                 when WriteB=> state <= SendAddrA_w;
35                 when SendAddrA_w => state <= WriteA;
36                 when WriteA=> state <= CheckLoop;
37                 when CheckLoop => if (CountEnd='1') then state<=CheckFlag; else state <= SendAddrA_r; end if;
38                 when CheckFlag => if (Flag='0') then state <= Waiting; else state <= SendAddrA_r; end if;
39             end case;
40         end if;
41     end process;
42
43     process (clk)
44     begin
45         if (clk'event and clk='1') then
46             case state is
47                 when Waiting => count<="00000000"; Flag<='0';
48                 when SendAddrA_r => Address <= count;
49                 when GetA => if (delay = '1') then dataA<=DataIn; count <=count+1; end if;
50                 when SendAddrB_r => Address<=count;
51                 when GetB => if (delay='1') then dataB<=DataIn; end if;
52                 when Compare => if Swap = '1' then count<=count-1; Flag<='1'; end if;
53                 when SendAddrB_w => Address<=count;DataOut<=dataB;
54                 when WriteB => count<= count+1;
55                 when SendAddrA_w => Address<=count; DataOut<= dataA;
56                 when WriteA => null;
57                 when CheckLoop => null;
58                 when CheckFlag => if (Flag='1') then Flag<='0'; count<="00000000"; end if;
59             end case;
60         end if;
61     end process;
62
63     WR<= '1' when state= WriteB or state=WriteA else '0';
64     Swap<='1' when dataA>dataB else '0';
65     CountEnd<='1' when count="11111111" else '0';
66     Complete<='1' when state=Waiting else '0';
67 end RTL;
```

Sort.vhd

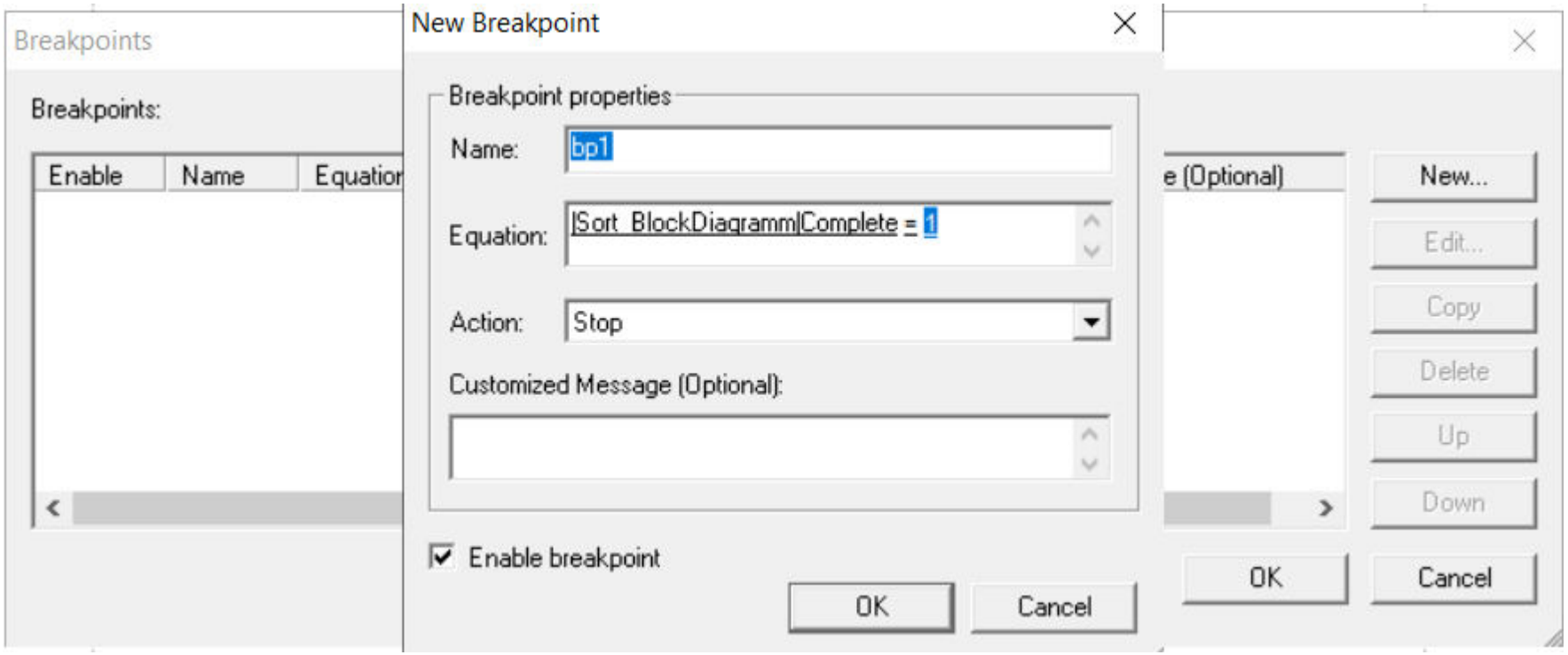
Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	FF	FE	FD	FC	FB	FA	F9	F8
8	00	00	00	00	00	00	00	00
16	00	00	00	00	00	00	00	00
24	00	00	00	00	00	00	00	00
32	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00
48	00	00	00	00	00	00	00	00
56	00	00	00	00	00	00	00	00
64	00	00	00	00	00	00	00	00
72	00	00	00	00	00	00	00	00
80	00	00	00	00	00	00	00	00
88	00	00	00	00	00	00	00	00
96	00	00	00	00	00	00	00	00
104	00	00	00	00	00	00	00	00
112	00	00	00	00	00	00	00	00
120	00	00	00	00	00	00	00	00
128	00	00	00	00	00	00	00	00
136	00	00	00	00	00	00	00	00
144	00	00	00	00	00	00	00	00
152	00	00	00	00	00	00	00	00
160	00	00	00	00	00	00	00	00
168	00	00	00	00	00	00	00	00
176	00	00	00	00	00	00	00	00
184	00	00	00	00	00	00	00	00
192	00	00	00	00	00	00	00	00
200	00	00	00	00	00	00	00	00
208	00	00	00	00	00	00	00	00
216	00	00	00	00	00	00	00	00
224	00	00	00	00	00	00	00	00
232	00	00	00	00	00	00	00	00
240	00	00	00	00	00	00	00	00
248	00	00	00	00	00	00	00	00



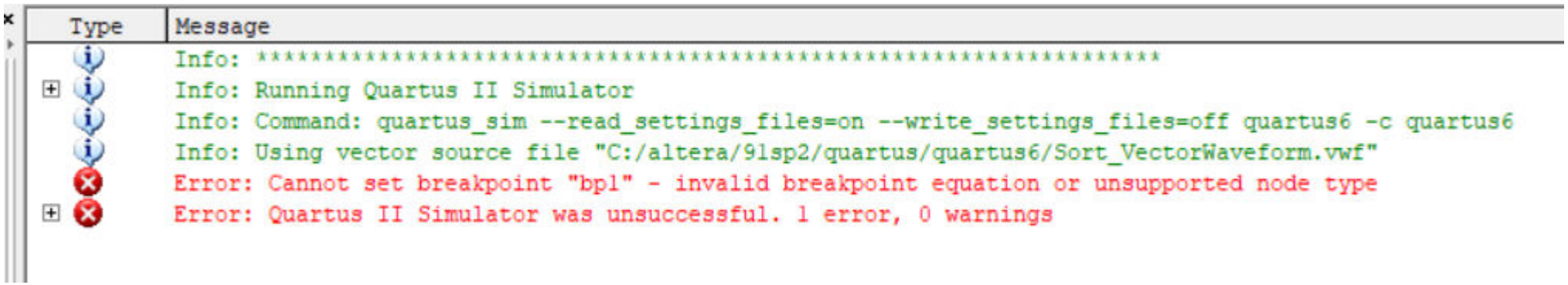
Το παραπάνω vwf μας δείχνει οτι το κύκλωμα βρίσκεται στο state.Waiting μεχρι να ενεργοποιηθεί το σήμα launch (σε θετική ακμή) οπου και ξεκινάνε τα 2 processes του bubble sort μπαίνοντας στο state.SendAddrA_r επειτα στο state.GetA κ.ο.κ. Το flag γινεται 1 στα 380ns γιατι εκεί εχουν φορτωθεί οι αριθμοί A(FF=11111111),B(FE=11111110) και βρισκόμαστε στο state.compare όπου αν A>B τοτε flag=1 για να υποδείξει οτι πρέπει να κάνουμε swap.



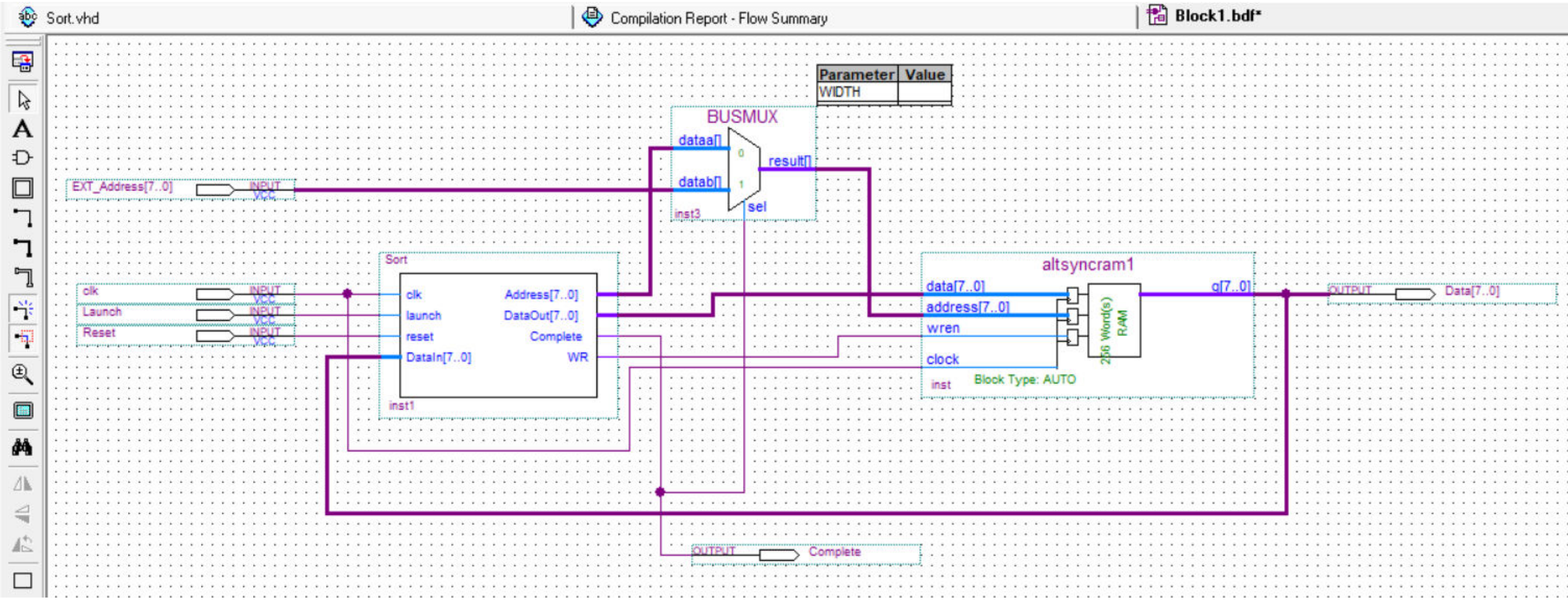
Όταν διαβάσει όλες τις θέσεις μνήμης τότε το complete γίνεται 1 και το πρόγραμμα παραμένει στο state.waiting μέχρι το επόμενο launch.

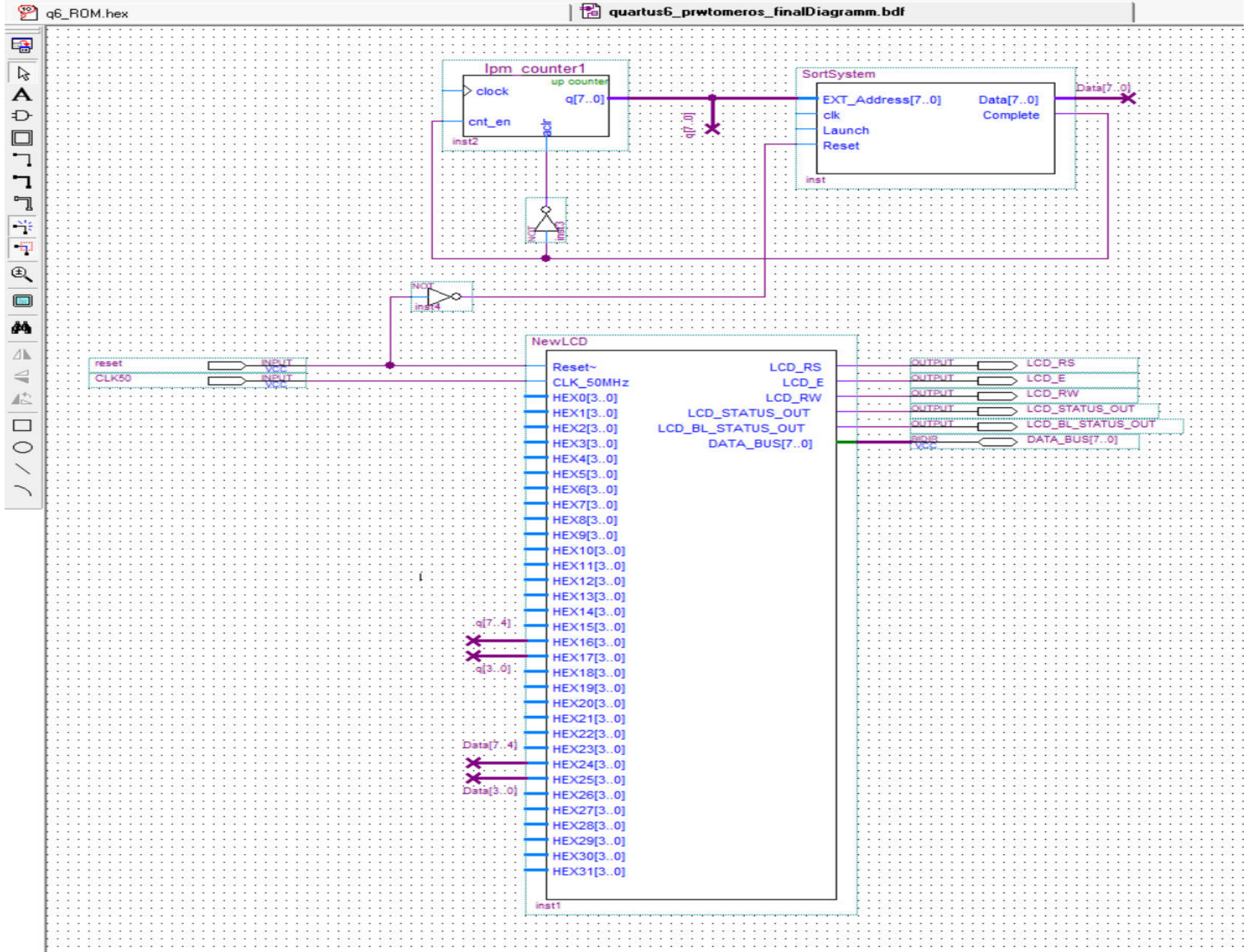


Προσπαθήσαμε να κάνουμε το μέρος με το breakpoint αλλά μας έβγαζε αυτο το ερρορ ή crashare το quartus αν πατούσαμε το embedded memory.



Υλοποίηση Board




q6_ROM.hex
quartus6_prwtomeros_finalDiagramm.bdf
lpm_counter1
up counter
clock
q[7..0]
cnt_en
aclr
inst2
NOT
inst3
q[7..0]
SortSystem
EXT_Address[7..0]
clk
Launch
Reset
Data[7..0]
Complete
Data[7..0]
inst
NOT
inst4
NewLCD
reset
CLK50
INPUT
VCC
Reset~
CLK_50MHz
HEX0[3..0]
HEX1[3..0]
HEX2[3..0]
HEX3[3..0]
HEX4[3..0]
HEX5[3..0]
HEX6[3..0]
HEX7[3..0]
HEX8[3..0]
HEX9[3..0]
HEX10[3..0]
HEX11[3..0]
HEX12[3..0]
HEX13[3..0]
HEX14[3..0]
HEX15[3..0]
HEX16[3..0]
HEX17[3..0]
HEX18[3..0]
HEX19[3..0]
HEX20[3..0]
HEX21[3..0]
HEX22[3..0]
HEX23[3..0]
HEX24[3..0]
HEX25[3..0]
HEX26[3..0]
HEX27[3..0]
HEX28[3..0]
HEX29[3..0]
HEX30[3..0]
HEX31[3..0]
LCD_RS
LCD_E
LCD_RW
LCD_STATUS_OUT
LCD_BL_STATUS_OUT
DATA_BUS[7..0]
OUTPUT
BIDIR
q[7..4]
q[3..0]
Data[7..4]
Data[3..0]
inst1

q6_ROM.hex

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	41	44	44	52	45	53	53	20
8	44	41	54	41	20	20	20	20
16	00	00	20	20	20	20	20	20
24	00	00	20	20	20	20	20	20

ADDRESS								DATA			
AX	DX							DY	AY		

Το Q6_ROM φτιάχτηκε για να προβάλλει αυτό που μας ζητείται στην εκφώνηση. Δόθηκαν τα κατάλληλα γράμματα για τις λεξεις ADDRESS , DATA και έμειναν στο 00 οι είσοδοι XX(16,17) και YY(24,25) στα υπόλοιπα δόθηκε η τιμή 20 (“space”) .

Στο εργαστήριο

(δε μπορούσαμε να το υλοποιήσουμε σε vhdl οπότε απλά εξηγούμε την ιδέα μας)

Απαραίτητο για να ξεκινήσει το πρόγραμμα ειναι το state.Waiting και συνεπώς το launch τα οποία έχουν τις ίδιες ιδιότητες όπως και στο πρόγραμμα που μας δόθηκε προηγουμένως αντίστοιχα και για τα delay , CheckLoop, CheckFlag.
Το πρόγραμμα θέλουμε να διαβάζει δυο τιμές τα A[i],B[i] οπότε χρειαζόμαστε τα state_type : SendAddrA_r , GetA, SendAddrB_r, GetB .
Θέλουμε να γραφουμε στη θέση 000 αρα χρειαζόμαστε και τα SendAddrA_w , WriteA τα οποία θα αποθηκεύουν το άθροισμα των αποκομμένων δεκαδικών ψηφίων.Θα χρειαστούμε αντιστοίχως και τα SendAddrC_w , WriteC για την αποθήκευση του μέσου όρου των A ,B στις θέσεις μνήμης 200-2FF.
Τη πράξη της πρόσθεσης θα την κάνουμε με τον ήδη υπάρχων τελεστή πρόσθεσης (+) και οι αριθμοί θα εχουν προέλθει απο τα getA, getB και θα έχουν αποθηκευτεί στα signals dataA, dataB αντίστοιχα
Το αποτέλεσμα της πρόσθεσης θα αποθηκευτεί στο signal Sum και εκει θα ελέγχουμε αν το LSB είναι '1'.
Αν ειναι 1 η τιμή του Signal value θα γινόταν value<= value+1.
Έπειτα το πρόγραμμα συνεχίζει κάνοντας δεξιά ολίσθηση στη τιμή του Signal Sum. Με αυτόν τον τρόπο διατηρείται το ακέραιο μέρος της διαϊρεσης με το 2[πχ1 1001(9) θα γίνει 0100(4) πχ2 0100(4) θα γίνει 0010(2)]. Οι τιμές αυτές αποθηκεύονται στο Signal Average[7..0].
Μετα κανουμε την εγγραφή στις θέσεις μνημης 200-2FF(μια θέση μνήμης τη φορά) .
Αφού βγεί ο μέσος όρος ολων των A,B και εχει αποθηκευτεί το συνολικο αθροισμα των δεκαδικών ψηφίων που χάθηκαν στο Signal value, μένει μόνο να κανουμε μια δεξια ολίσθηση στο value (γιατι καθε φορα που κάναμε αποκοπή θα έπρεπε να προσθέταμε 0.5 ενω εμείς προσθέταμε 1).
Τελος κάνουμε εγγραφή στη θέση μνήμης 000 την τιμη του Signal value και τερματίζεται το πρόγραμμα κάνοντας Complete=1.