Εργασία στα Ψηφιακά HW 2 σε χαμηλά επίπεδα λογικής

Νίκος Λαδιάς , AEM 9362

16-bit απαριθμητής up/down

Design

Στο πρώτο μέρος της εργασίας ζητείται η υλοποίηση ενός δεκαεξαμπιτου απαριθμητή που μετράει προς τα πάνω αλλά και προς τα κάτω, με δυνατότητα εξώθησης δεδομένων από την είσοδο στην έξοδο και ασύγχρονο reset. Η υλοποίηση έγινε με βάση τις ακριβείς οδηγίες της εκφώνησης και με τα σήματα εισόδου-εξόδου που αναγράφονται. Αφού δηλώθηκαν οι αντίστοιχες ports, σε ένα always_ff block ευαίσθητο σε θετική ακμή ρολογιού και σε αρνητική του rst σήματος, έγιναν όλοι οι απαραίτητοι έλεγχοι για τις κατάλληλες τιμές των δεδομένων εξόδου. Έτσι, με προτεραιότητα το ασύγχρονο rst σήμα, μηδενίζονται τα δεδομένα εξόδου. Με δεύτερη σε σειρά προτεραιότητα το σήμα ld_cnt, όταν είναι στο μηδέν, περνάει τα δεδομένα εισόδου σε αυτά της εξόδου, ενώ διαφορετικά αν υπάρχει count_enb (δηλαδή τιμή 1), ανάλογα την τιμή του updn_cnt (1 ή 0) , τα δεδομένα εξόδου τροποποιούνται (αυξάνονται κατα ένα ή μειώνονται αντίστοιχα). Παρακάτω ο κώδικας του up_down_16bit_counter module, όπως ο απαριθμητής ονομάστηκε. Όλες οι αναθέσεις γίνονται με non-blocking assignment.

```
1  module up_down_16bit_counter(input clk,count_enb,updn_cnt,rst,ld_cnt,
2  input logic [15:0] data_in,output logic [15:0] data_out);
3
4  always_ff@(posedge clk,negedge rst) begin
5  if (!rst) data_out <= 16'b0;   //Priority on active low reset
6  else begin
7     if(!ld_cnt) data_out<=data_in; // Load operation has priority over count_enb
8     else if (count_enb) begin        // Count up or down according to count_enb
9           if(updn_cnt)  data_out<=data_out+1;
10          else      data_out<=data_out-1;
11    end
12   //no else here, if count_enb is low, data remains stable, no operation needed
13 end
14
15 end
16 endmodule
17
```

Verification & Testing

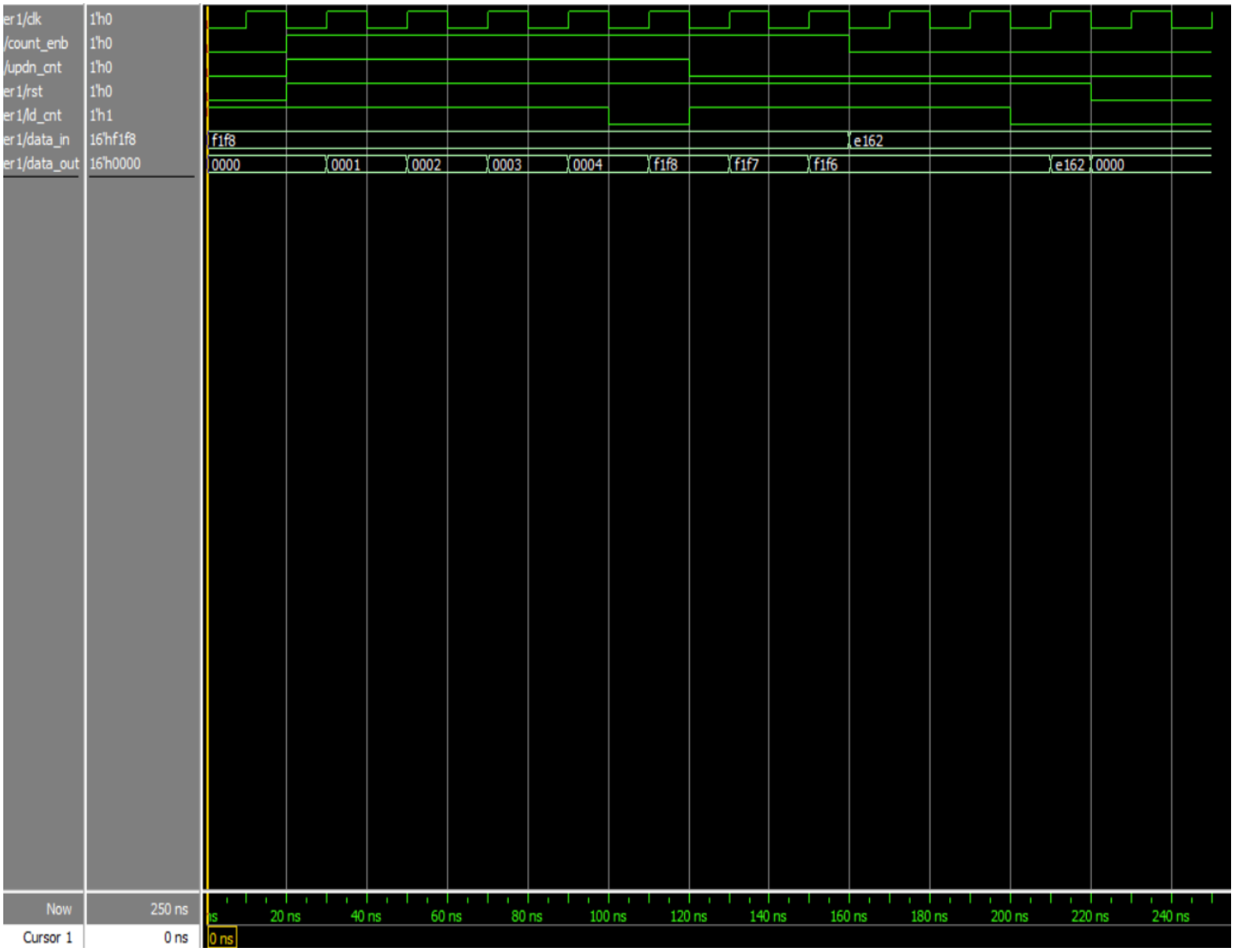
Πρώτα ορίζεται module όπου δηλώνονται τα κατάλληλα properties, τα οποία θα δοκιμαστούν με την χρήση του assert. Αυτά δημιουργήθηκαν με βάση πάλι την ακριβής περιγραφή της εκφώνησης για τις ανάγκες σωστής επαλήθευσης του

απαριθμητή. Αρχικά, υπάρχει ένα property που ελέγχει την τιμή της εξόδου για όταν υπάρχει αρνητική ακμή στο σήμα rst, και κοιτάει στην επόμενη θετική ακμή ρολογιού αν τα δεδομένα εξόδου είναι μηδέν . Έπειτα υπάρχουν δύο ακολουθίες (sequences) , μια για τις συνθήκες σταθερών δεδομένων εξόδου, και μια για τις συνθήκες αλλαγής δεδομένων εξόδου. Δύο properties , ελέγχουν για σταθερότητα δεδομένων εξόδου και αλλαγή αντίστοιχα, τα οποία πυροδοτούνται απο την κατάλληλη ακολουθία. Και τα δύο είναι απενεργοποιημένα όταν υπάρχει ενεργό rst (δηλαδή 0) , ενώ για την σταθερότητα των δεδομένων εξόδου ο έλεγχος γίνεται με την συνάρτηση $stable() , και ελέγχονται αν παραμένουν σταθερά ανάμεσα σε δύο ακμές ρολογιού. Για την αλλαγή των δεδομένων, ο έλεγχος γίνεται με if-else, και χρήση της $past() συνάρτησης, όπου εξετάζεται αν τα δεδομένα έγιναν η παλιά τους τιμή συν ένα , για updn_cnt=1 και η παλιά τους τιμή μείον ένα για updn_cnt=0. Όλα τα assertions είναι concurrent , και τυπώνουν τα κατάλληλα μηνύματα ανάλογα την αποτυχία ή επιτυχία του assertion καθώς και τον χρόνο που αξιολογήθηκαν.

Σε ένα τρίτο module, στο οποίο αφού πρώτα γίνει instantiate ένα αντικείμενο τύπου up_down_16bit_counter, ορίζονται τα κατάλληλα σήματα και δένονται το module που περιέχει τα properties με αυτό του design, με τις εισόδους-εξόδους τους να δηλώνονται ρητά. Έπειτα, δημιουργείται με ένα always block ένα ρολόι με περίοδο 20 time units, και δίνονται τιμές για testing , με την βοήθεια ενός initial block.

Αρχικά ο απαριθμητής γίνεται reset, απενεργοποιεί το ld_cnt, και δίνεται και μια δεκαεξαμπιτη τιμή στα δεδομένα εισόδου. Μετά από μια περίοδο, το rst απενεργοποιείται , και ο απαριθμητής ξεκινά να μετρά προς τα πάνω για 4 περιόδους. Έπειτα ενεργοποιείται το ld_cnt για να εξεταστεί η προτεραιότητα του ως προς την απαρίθμηση. Μετά από μια περίοδο το ld_cnt απενεργοποιείται και ο απαριθμητής ξεκινά να μετρά προς τα κάτω για δύο περιόδου. Ύστερα, σταματά την απαρίθμηση και νέα δεδομένα εισόδου εμφανίζονται, τα οποία μετά από δύο περιόδους δίνεται σήμα να εμφανιστούν στην έξοδο με το σήμα ld_cnt. Τέλος , μια αρνητική ακμή του rst, μηδενίζει τα δεδομένα εξόδου. Ακολουθεί εικόνα των μηνυμάτων που τυπώθηκαν για τα assertions, στην κονσόλα της προσομοίωσης και μια στην οποία φαίνονται όλα τα σήματα και οι τιμές που παίρνουν. Οι τιμές που φαίνονται στα σήματα στην εικόνα της προσομοίωσης, είναι στην αρχή της.

```
 10    up_down_16bit_counter_bind.up_down_16bit_counter1.up_down_16bit_counter_bound.Data_Reset PASS
 30    up_down_16bit_counter_bind.up_down_16bit_counter1.up_down_16bit_counter_bound.Data_Change FAIL
 50    up_down_16bit_counter_bind.up_down_16bit_counter1.up_down_16bit_counter_bound.Data_Change PASS
 70    up_down_16bit_counter_bind.up_down_16bit_counter1.up_down_16bit_counter_bound.Data_Change PASS
 90    up_down_16bit_counter_bind.up_down_16bit_counter1.up_down_16bit_counter_bound.Data_Change PASS
130    up_down_16bit_counter_bind.up_down_16bit_counter1.up_down_16bit_counter_bound.Data_Change FAIL
150    up_down_16bit_counter_bind.up_down_16bit_counter1.up_down_16bit_counter_bound.Data_Change PASS
170    up_down_16bit_counter_bind.up_down_16bit_counter1.up_down_16bit_counter_bound.Data_Stable FAIL
190    up_down_16bit_counter_bind.up_down_16bit_counter1.up_down_16bit_counter_bound.Data_Stable PASS
230    up_down_16bit_counter_bind.up_down_16bit_counter1.up_down_16bit_counter_bound.Data_Reset PASS
```



Εκτός των PASS που επιβεβαιώνουν την σωστή λειτουργία του κυκλώματος, υπάρχουν επίτηδες τρεις περιπτώσεις αποτυχίας. Στην πρώτη τα δεδομένα αλλάζουν αλλά η τιμή που δειγματοληπτείται είναι άγνωστη διότι δεν υπάρχει προηγούμενη τιμή πριν ξεκινήσει η προσομοίωση. Στη δεύτερη περίπτωση , ουσιαστικά είναι παρόμοιο πρόβλημα, πέφτει στο μηδέν το updn_cnt και δειγματοληπτείται η τιμή των δεδομένων εξόδου, όμως φορτώθηκαν δεδομένα από την είσοδο και την πρώτη φορά που καλείται δεν λαμβάνει την τιμή που πρέπει (δηλαδή εδώ η παλιά τους τιμή μείον ένα), αφού όταν καλείται το assertion ελέγχει την τιμή που είχαν πριν, και όχι αυτήν που αποκτούν την στιγμή που καλείται. Έτσι, επειδή τα δεδομένα εισόδου που φορτώθηκαν προφανώς διαφέρουν από την προηγούμενη τιμή μείον ένα , αποτυχαίνει. Ίδια λογική και στην τρίτη αποτυχία, όπου πυροδοτείται το assertion αλλά κοιτάζει αν παρέμειναν τα δεδομένα σταθερά στον προηγούμενο κύκλο ρολογιού, που δεν παρέμειναν διότι το count_enb ήταν ακόμη ένα και άλλαζαν τα δεδομένα.

Απλή σύγχρονη FIFO

Design

Στο δεύτερο μέρος της εργασίας ζητείται η υλοποίηση μιας απλής σύγχρονης FIFO (First In First Out) δομής δεδομένων, με παραμετροποιημένο πλάτος (μέγεθος bit θέσεων) και βάθος (μέγεθος ουράς σε θέσεις), αυτή ονομάστηκε fifo_memory. Αφού οριστούν όλες οι κατάλληλες ports , όπως ζητούνται στην εκφώνηση και ο πίνακας δεδομένων της FIFO, με μέγεθος θέσεων και μέγεθος σε θέσεις όπως προαναφέρθηκε, υλοποιήθηκε η σύγχρονη συμπεριφορά της FIFO σε ένα always_ff block . Η ανάθεση όμως των μεταβλητών fifo_full και fifo_empty είναι συνεχής, αφού πάντα αποτιμούνται με βάση την αντίστοιχη συνθήκη του cnt (cnt>=DEPTH και cnt==0 αντίστοιχα) . Ο cnt ορίστηκε ως integer .

Οι δύο pointers, μηδενίζονται αρχικά αν φτάσανε στην κορυφή της στοίβας (δηλ. DEPTH) . Επιπλέον , με βάση τα ζητούμενα, αρχικοποιούνται μαζί με τον cnt και την fifo_memory σε αρνητική ακμή του reset. Όταν υπάρχει αίτημα για write, περνάνε τα δεδομένα εισόδου στην θέση που δείχνει ο wr_ptr στην fifo_memory, ενώ αυξάνονται ο cnt και ο wr_ptr. Αντίστοιχα, όταν υπάρχει αίτημα για read, περνάνε τα δεδομένα από την θέση που δείχνει ο rd_ptr στην fifo_memory στα δεδομένα εξόδου, και αυξάνεται ο rd_ptr ενώ μειώνεται ο cnt. Οι αυξομειώσεις γίνονται πάντα κατά ένα. Ο κώδικας φαίνεται στην παρακάτω εικόνα.

```
1  module FIFO #( parameter  WIDTH=16,DEPTH=16 ) //parameterized width and depth
2   (input rst,clk,fifo_write,fifo_read,input logic [WIDTH-1:0] fifo_data_in,output logic[WIDTH-1:0] fifo_data_out,output fifo_full,fifo_empty);
3   // output and inputs defined
4   integer cnt;
5   logic [4:0] wr_ptr,rd_ptr;
6   logic [WIDTH-1:0] fifo_memory[DEPTH-1:0]; // the stack is WIDTH bits wide and DEPTH locations in size
7    //assigning fifo_full and fifo_empty condition
8   assign fifo_full= ( cnt==DEPTH );
9   assign fifo_empty = (cnt==0);
10
11  always_ff@(posedge clk,negedge rst) begin
12   // make pointers zero by default if they reached top of the stack
13  if(wr_ptr==DEPTH) wr_ptr=0;
14  if(rd_ptr==DEPTH) rd_ptr=0;
15
16  if (!rst) begin
17  //priority on active low reset
18    for (integer i=0; i<DEPTH-1; i=i+1) fifo_memory[i]<=16'b0;
19  wr_ptr<=0;
20  rd_ptr<=0;
21  cnt<=0; // by making counter zero, fifo_empty and fifo_full automatically reset, since there is a continuous assignment on them
22  fifo_data_out<=16'b0;
23  end
24  else begin
25   if (fifo_write & !fifo_full) begin
26  fifo_memory[wr_ptr]<=fifo_data_in; //put data into the stack
27  wr_ptr<=wr_ptr+1; // write pointer increments on write request
28  cnt<=cnt+1;  //counter increments
29  end
30  else if (fifo_read & !fifo_empty) begin
31   fifo_data_out<=fifo_memory[rd_ptr]; //read data from the stack, data are read from the bottom of the stack cause first in first out architecture
32   rd_ptr<=rd_ptr+1; // read pointer increments on read request
33  cnt<=cnt-1; //counter decrements
34  end // internal else if end
35  end // external if  end
36  end // always_ff end
37  endmodule
```

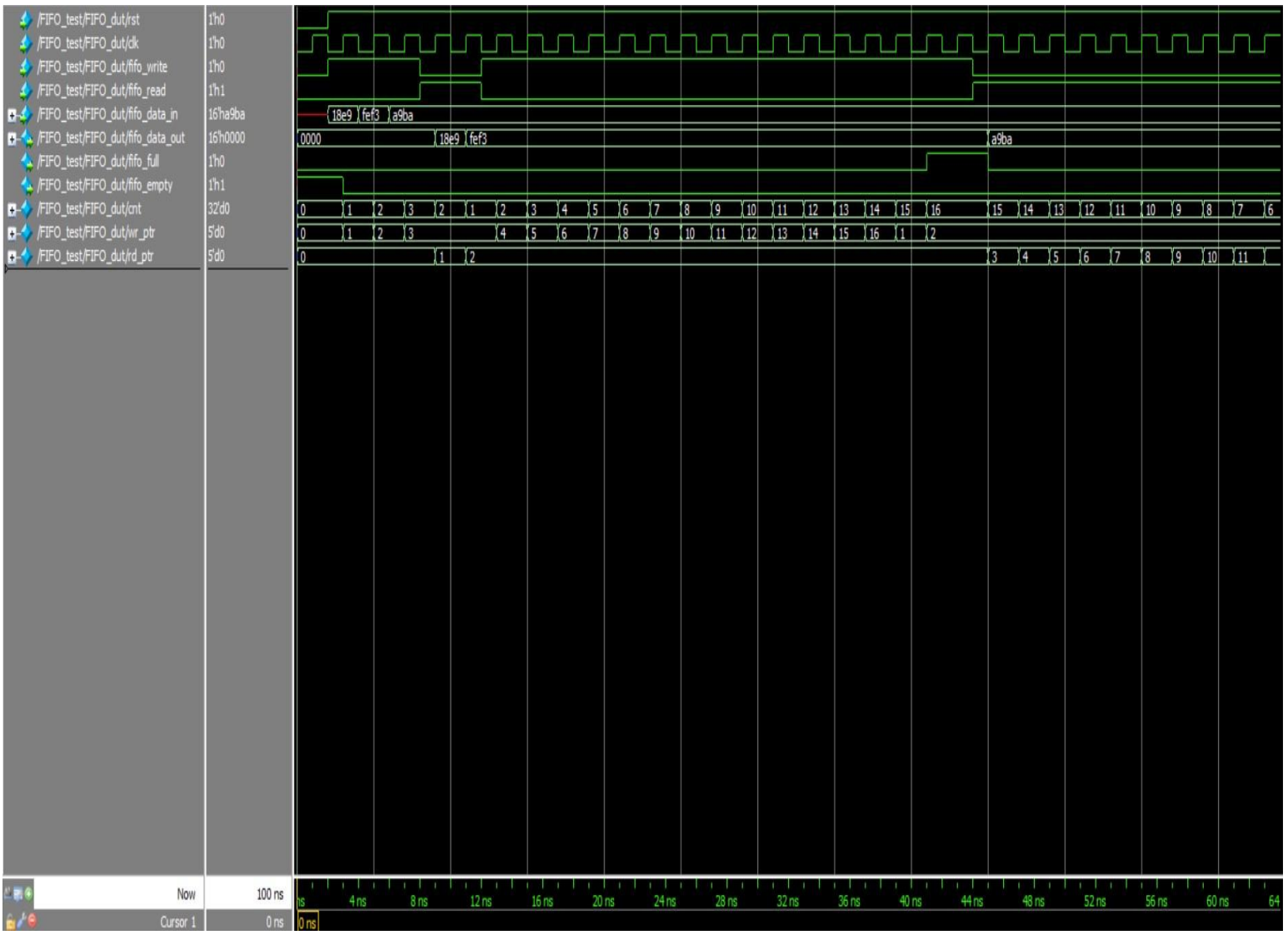
Verification & Testing

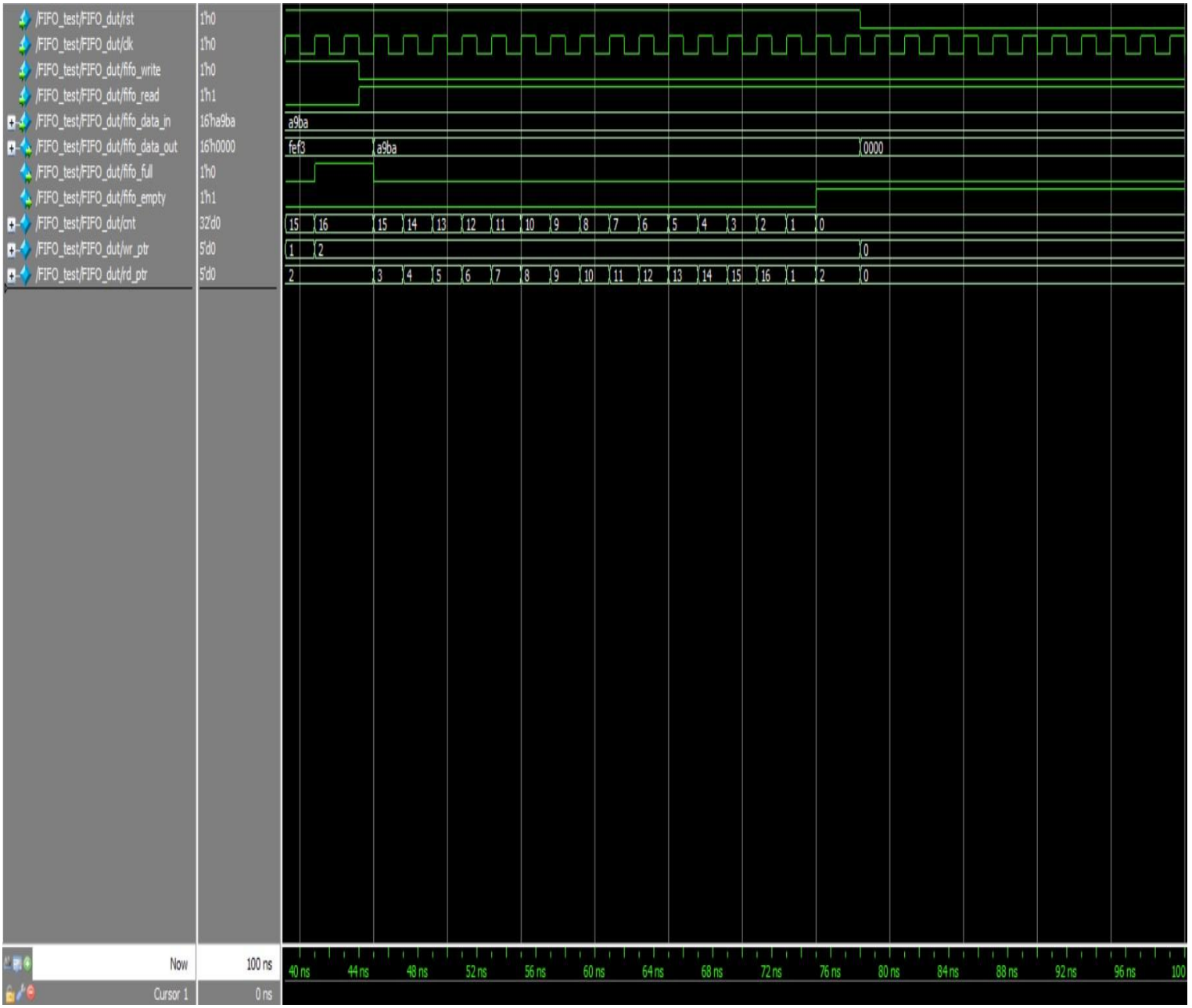
Πρώτα ορίζεται module όπου δηλώνονται τα κατάλληλα properties, τα οποία θα δοκιμαστούν με την χρήση του assert. Αυτά δημιουργήθηκαν με βάση πάλι την ακριβής περιγραφή της εκφώνησης για τις ανάγκες σωστής επαλήθευσης της FIFO. Αρχικά, υπάρχει ένα property που ελέγχει τις ζητούμενες μεταβλητές για όταν υπάρχει αρνητική ακμή στο σήμα rst, και κοιτάει στην επόμενη θετική ακμή ρολογιού αν η μεταβλητή fifo_full και οι δύο pointers είναι 0 καθώς και ο cnt, μαζί με την fifo_empty που πρέπει να είναι 1. Έπειτα, ένα απλό property για τον έλεγχο κενής και γεμάτης FIFO, τα οποία απενεργοποιούνται στο !reset, ο έλεγχος γίνεται αν cnt==0 και αν cnt>=DEPTH αντίστοιχα. Τέλος δύο τελευταία properties , για τον έλεγχο σταθερότητας των pointer στα αντίστοιχα αιτήματα όταν η μνήμη δεν επιτρέπει την ενέργεια αυτή. Δηλαδή σταθερότητα του wr_ptr όταν fifo_write=1 και fifo_full=1, καθώς και σταθερότητα του read_ptr όταν fifo_read=1 και fifo_empty=1.

Αφού γίνει και το τρίτο module που κάνει κατάλληλα bind το property module με αυτό του design, υλοποιείται και η διαδικασία testbench σε ένα initial block, το ρολόι δημιουργήθηκε με ίδια λογική αυτή τη φορά με περίοδο 2 time units.

Στο testing, αρχικά ο απαριθμητής γίνεται reset, έπειτα ενεργοποιείται το write αίτημα και γίνονται τρεις διαδοχικές εγγραφές διαφορετικών δεδομένων. Έπειτα γίνονται δύο διαδοχικές αναγνώσεις, αφου ενεργοποιηθεί προφανώς το αίτημα ανάγνωσης και κλείσει αυτό της εγγραφής. Ύστερα, γίνονται 15 συνεχόμενες εγγραφές των ίδιων δεδομένων (αυτών που εγγράφηκαν στην τρίτη θέση πρίν) και ελέγχεται μετά η πληρότητα της μνήμης, τέλος γίνεται ανάγνωση όλων των δεδομένων και ελέγχεται και η απώλεια δεδομένων της FIFO, η διαδικασία κλείνει με reset. Παρακάτω φαίνονται στις δύο εικόνες η διαδικασία με τις τιμές των pointers και cnt σε δεκαδικό καθώς και όλες οι υπόλοιπες τιμές των μεταβλητών της FIFO. Τα δεδομένα εισόδου-εξόδου είναι σε δεκαεξαδικό.Όλα λειτουργούν κατάλληλα. Ακολουθεί η εικόνα όλων των επαληθεύσεων των assertion σε χρόνο προσομοίωσης που τυπώνει η κονσόλα και τα δύο στιγμιότυπα της προσομοιώσης της FIFO.

```
VSIM 37> run
#                1      FIFO_test.FIFO_dut.FIFO_bind.FIFO_Reset PASS
#                3      FIFO_test.FIFO_dut.FIFO_bind.FIFO_Empty PASS
#               43      FIFO_test.FIFO_dut.FIFO_bind.FIFO_Full PASS
#               45      FIFO_test.FIFO_dut.FIFO_bind.Write_Ptr_Stable PASS
#               45      FIFO_test.FIFO_dut.FIFO_bind.FIFO_Full PASS
#               77      FIFO_test.FIFO_dut.FIFO_bind.FIFO_Empty PASS
#               79      FIFO_test.FIFO_dut.FIFO_bind.FIFO_Reset PASS
#               79      FIFO_test.FIFO_dut.FIFO_bind.Read_Ptr_Stable PASS
#               81      FIFO_test.FIFO_dut.FIFO_bind.Read_Ptr_Stable PASS
#               83      FIFO_test.FIFO_dut.FIFO_bind.Read_Ptr_Stable PASS
#               85      FIFO_test.FIFO_dut.FIFO_bind.Read_Ptr_Stable PASS
#               87      FIFO_test.FIFO_dut.FIFO_bind.Read_Ptr_Stable PASS
#               89      FIFO_test.FIFO_dut.FIFO_bind.Read_Ptr_Stable PASS
#               91      FIFO_test.FIFO_dut.FIFO_bind.Read_Ptr_Stable PASS
#               93      FIFO_test.FIFO_dut.FIFO_bind.Read_Ptr_Stable PASS
#               95      FIFO_test.FIFO_dut.FIFO_bind.Read_Ptr_Stable PASS
#               97      FIFO_test.FIFO_dut.FIFO_bind.Read_Ptr_Stable PASS
#               99      FIFO_test.FIFO_dut.FIFO_bind.Read_Ptr_Stable PASS
```


/FIFO_test/FIFO_dut/rst
/FIFO_test/FIFO_dut/clk
/FIFO_test/FIFO_dut/fifo_write
/FIFO_test/FIFO_dut/fifo_read
/FIFO_test/FIFO_dut/fifo_data_in
/FIFO_test/FIFO_dut/fifo_data_out
/FIFO_test/FIFO_dut/fifo_full
/FIFO_test/FIFO_dut/fifo_empty
/FIFO_test/FIFO_dut/cnt
/FIFO_test/FIFO_dut/wr_ptr
/FIFO_test/FIFO_dut/rd_ptr
1'h0
1'h0
1'h0
1'h1
16'ha9ba
16'h0000
1'h0
1'h1
32'd0
5'd0
5'd0
18e9
fef3
a9ba
0000
18e9
fef3
a9ba
Now
100 ns
Cursor 1
0 ns
4 ns
8 ns
12 ns
16 ns
20 ns
24 ns
28 ns
32 ns
36 ns
40 ns
44 ns
48 ns
52 ns
56 ns
60 ns


/FIFO_test/FIFO_dut/rst
/FIFO_test/FIFO_dut/clk
/FIFO_test/FIFO_dut/fifo_write
/FIFO_test/FIFO_dut/fifo_read
/FIFO_test/FIFO_dut/fifo_data_in
/FIFO_test/FIFO_dut/fifo_data_out
/FIFO_test/FIFO_dut/fifo_full
/FIFO_test/FIFO_dut/fifo_empty
/FIFO_test/FIFO_dut/cnt
/FIFO_test/FIFO_dut/wr_ptr
/FIFO_test/FIFO_dut/rd_ptr
1'h0
1'h0
1'h0
1'h1
16'ha9ba
16'h0000
1'h0
1'h1
32'd0
5'd0
5'd0
a9ba
fef3
a9ba
0000
15 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 2 0
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 1 2 0
Now
100 ns
Cursor 1
0 ns
40 ns 44 ns 48 ns 52 ns 56 ns 60 ns 64 ns 68 ns 72 ns 76 ns 80 ns 84 ns 88 ns 92 ns 96 ns 100