2022

Ψηφιακά Συστήματα HW σε χαμηλό επίπεδο λογικής ΙΙ

ΕΡΓΑΣΙΑ ΕΞΑΜΗΝΟΥ

ΝΙΚΟΛΑΟΣ ΓΙΑΝΝΟΠΟΥΛΟΣ
ΑΕΜ: 9629
NGIANNOP@ECE.AUTH.GR

Εισαγωγή

Στην παρούσα εργασία θα χρησιμοποιήσουμε το εργαλείο Questa- Intel FPGA Starter Edition 2021 για την υλοποίηση σε HDL SystemVerilog & το verification αυτής. Αποτελείται από 2 κομμάτια το πρώτο είναι η περιγραφή ενός Up-Down counter μήκους 16 bit και το δεύτερο είναι η υλοποίηση μιας σύγχρονης FIFO μνήμης 16x16 bit.

Up-Down Counter

Ο counter αυτός αποτελείται από τα εξής

- 16-bit data_in
- 16-bit data_out
- Ασύγχρονο rst_
- Σήμα ενεργοποίησης μετάβασης δεδομένων από την είσοδο στην έξοδο ld_cnt
- Σήμα ενεργοποίησης για την ενεργοποίηση του counter με όνομα count_enb
- Σήμα ενεργοποίησης για την μέτρηση προς τα πάνω ή προς τα κάτω updn_cnt
- Τέλος, το ρολόι του μετρητή clk

RTL-Code Up-Down Counter

Η υλοποίηση του RTL ήταν εύκολη καθώς δεν αποτελεί περίπλοκο κύκλωμα όπως φαίνονται και στις παρακάτω εικόνες.

```
1   `timescale 1 ns / 1 ns
2   module upDownCounter16bit (
3       data_in,
4       rst_,
5       ld_cnt,
6       updn_cnt,
7       count_enb,
8       clk,
9       data_out
10  );
11    input logic [15:0] data_in;
12    input logic rst_, ld_cnt, updn_cnt, count_enb, clk;
13    output logic [15:0] data_out;
14
15    always_ff @(posedge clk, negedge rst_)begin
16      if(!rst_)begin  // active low reset
17        data_out <= 0;
18      end
19      else if(!ld_cnt)begin      //actice low load_counter
20        data_out <= data_in;
21      end
22      else if(count_enb == 1)begin
23        if(updn_cnt == 1)begin
24          if(data_out == 16'b1)begin
25              data_out <= 0;
26          end else begin
27              data_out <= data_out + 1;
28          end
29        end else if(updn_cnt == 0) begin
30          if(data_out == 16'b0)begin
31              data_out <= 16'hffff;
32          end else begin
33              data_out <= data_out - 1;
34          end
35        end
36      end
37    end
38
39  endmodule
40
```

Όπως φαίνεται η υλοποίηση του RTL στην γραμμή 15 έγινε η χρήση του always_ff όπου δημιουργεί ένα FF με το sensitive list του να αποτελείται από το posedge clk , negedge rst_ καθώς έχουμε ασύγχρονο σήμα reset .

Αν οποιαδήποτε στιγμή ενεργοποιηθεί το σήμα rst_ (active low) τότε τα δεδομένα στην έξοδο θα γίνουν μηδέν διαφορετικά αν το σήμα ld_cnt ενεργοποιηθεί (active low) τότε θα περαστούν τα δεδομένα από την είσοδο στην έξοδο. Αν το σήμα count_enb είναι 1 (active high) τότε θα ξεκινήσει να μετράει ο counter διαφορετικά θα παραμείνουν όπως είχαν στον προηγούμενο κύκλο.

Testbench Up-Down Counter

Στην παρακάτω εικόνα φαίνεται η υλοποίηση του testbench αρχείου για τον counter.

```
1   `timescale 1 ns / 1 ns
2   module upDownCounter16bit_testBench;
3
4     logic [15:0] data_in;
5     logic rst_, ld_cnt, updn_cnt, count_enb, clk;
6     logic [15:0]data_out;
7
8     initial begin
9
10    $display ("time\t clk \tld_cnt\t data_in\t\t data_out ");
11    $monitor ("%g\t %b\t %b\t %b\t", $time, clk, ld_cnt, data_in, data_out);
12
13    clk = 0;
14    rst_ = 1'b0;    // start with reset
15    //updn_cnt = 1'b1;    //
16    //ld_cnt = 1'b1;    // do load
17          #10
18
19    rst_ = 1'b1;    // reset at not trigger
20    data_in = 2;  // 2
21    ld_cnt = 1'b0;  // do load
22          #20
23
24    count_enb = 1'b1;
25          ld_cnt = 1'b1;  //load off
26    updn_cnt = 1'b1;  // count up
27    rst_ = 1'b1;    // reset off
28
29
30    #35
31    rst_ = 1'b1;
32    count_enb = 1'b0;
33          ld_cnt = 1'b1;
34    updn_cnt = 1'b0; // start count down
35    end
36
37  always #5 clk = ~clk;
38  upDownCounter16bit u0(.*);
39
40  initial  // Dump waveform for debug
41  $dumpvars;
42
43  endmodule
```

Στις γραμμές 10 και 11 έχουν υλοποιηθεί για την εκτύπωση στην console του Questa για να παρέχει μια εύκολη αποτίμηση των τιμών στον χρόνο προσομοίωσης και στην συνέχεια αρχικοποιούμε το clk = 0_{10} , rst_ = 0_{10} οπότε θα γίνει data_out = 0_{10}. Μετά από 10 t.u(time units) το rst γίνεται off και τα data_in = 2 (ακέραια τιμή) και ενεργοποιούμε το ld_cnt = 1’b0 (active low) για να περαστεί είσοδος στην έξοδο μετά από 10 t.u. ενεργοποιούμε τον μετρητή να μετράει προς τα πάνω μετά από 15 t.u. ξεκινάει να μετράει προς τα κάτω. Τέλος υπάρχει ένα always block (γραμμή 37) όπου έχω ορίσει την περίοδο ενός κύκλου να είναι ίση με 10 t.u. δηλαδή 5 t.u. ON 5 t.u. off.Η γραμμή 40 – 41 το χρησιμοποίησα για την εκτύπωση των σημάτων πάνω στο simulation

Στην παρακάτω εικόνα φαίνονται οι κυματομορφές της προσομοίωσης του testbench αρχείου όπου την χρονική στιγμή 20ns φαίνεται να φορτώνονται τα δεδομένα εισόδου στην έξοδο στην θετική ακμή του ρολογιού. Στα 30ns ενεργοποιείται το σήμα ld_cnt και στην επόμενη χρονική στιγμή γίνεται enable το count_enb και ξεκινάει να

Εικόνα 1 TestBench Up-Down Counter Simulation

μετράει προς τα πάνω. Καθώς στα 65ns το count_enb γίνεται disable το σήμα και έτσι σταματάει να μετράει ο counter.

Η έξοδος του transcript

# time	clk	ld_cnt	data_in	data_out
# 0	0	x	xxxxxxxxxxxxxxxx	0
# 5	1	x	xxxxxxxxxxxxxxxx	0
# 10	0	0	0000000000000010	0
# 15	1	0	0000000000000010	2
# 20	0	0	0000000000000010	2
# 25	1	0	0000000000000010	2
# 30	0	1	0000000000000010	2
# 35	1	1	0000000000000010	3
# 40	0	1	0000000000000010	3
# 45	1	1	0000000000000010	4
# 50	0	1	0000000000000010	4
# 55	1	1	0000000000000010	5
# 60	0	1	0000000000000010	5
# 65	1	1	0000000000000010	5
# 70	0	1	0000000000000010	5
# 75	1	1	0000000000000010	5
# 80	0	1	0000000000000010	5
# 85	1	1	0000000000000010	5
# 90	0	1	0000000000000010	5
# 95	1	1	0000000000000010	5

SystemVerilog Assertions Up-Down Counter

Στην παρακάτω εικόνα φαίνεται η υλοποίηση του αρχείου με τα assertions

```
1  `timescale 1ns/1ns
2  module upDownCounter16bit_property( data_in,
3      rst_,
4      ld_cnt,
5      updn_cnt,
6      count_enb,
7      clk,
8      data_out);
9
10   input logic [15:0] data_in , data_out;
11   input logic rst_, ld_cnt, updn_cnt, count_enb, clk;
12 `ifdef check_rst
13     property pr1;
14         @(posedge clk) $fell(rst_) |-> (data_out === 0);
15     endproperty
16
17     rstPr1: assert property (pr1) $display($stime,,,"\t\t %m PASS");
18         else $display($stime,,,"\t\t %m FAIL");
19
20
21 `elsif check_ld_cnt_and_count_en_notEnable
22     property pr2;                    //ld_cnt active low and count_enb active high
23         @(posedge clk) disable iff (!rst_) !ld_cnt && !count_enb |-> data_out === $past(data_out);
24     endproperty
25                             // if ld_cnt == 1 then load is off
26     ldCntCounterEnb: assert property(pr2) $display($stime,,,"\t\t %m PASS");
27         else $display($stime,,,"\t\t %m FAIL");
28 `elsif check_ld_cnt_notEnable_and_countEnb_and_updncntEnb
29     property pr3;                          //active low      active high
30         @(posedge clk) disable iff(!rst_)
31          ld_cnt && count_enb |->
32          if(updn_cnt)
33          ##1
34         data_out === $past(data_out) + 1
35          else
36         ##1
37         data_out === $past(data_out) -1;
38     endproperty
39
40     ldCntCountrEnbUpCnt: assert property(pr3) $display($stime,,,"\t\t %m PASS");
41         else $display($stime,,,"\t\t %m FAIL");
42
43 `endif
44 endmodule
```

καθώς έκανα την χρήση macro για την πιο εύκολη διαχείρισή του επιπλέον χρησιμοποίησα ως είσοδο τα data_out για να μπορέσω να υλοποιήσω το 1ο property. Στην γραμμή 14 καθώς όταν έχουμε θετική ακμή ρολογιού και με την κατερχόμενη ακμή του σήματος rst_ και στον ίδιο κύκλο (|->) ελέγχουμε αν τα data_out είναι ίσο με μηδέν αν όλα είναι θετικά τότε το property pr1 κάνει PASS αλλιώς κάνει FAIL.

Το 2ο property pr2 στην θετική ακμή του ρολογιού αν όμως υπάρχει σήμα rst_ τότε απενεργοποιείται το συγκεκριμένο property(disable iff($fell(rst_))). Διαφορετικά αν το ld_cnt είναι disable και το count_enb disable τότε στην ίδια ακμή του ρολογιού (|->) αν τα data_out είναι ίδια με εκείνα της προηγούμενης ακμής του ρολογιού τότε το property κάνει PASS διαφορετικά FAIL.

Το 3ο property pr3 στην θετική ακμή του ρολογιού αν όμως υπάρχει σήμα rst_ τότε απενεργοποιείται το συγκεκριμένο property(disable iff($fell(rst_))). Διαφορετικά αν το ld_cnt είναι disable και το count_enb enable τότε στην ίδια ακμή του ρολογιού (|->) αν το

updn_cnt είναι 1 (προς τα πάνω μέτρηση) τότε μετά από έναν κύκλο ρολογιού ##1 ελέγχουμε αν τα data_out είναι ίσα με του προηγούμενου κύκλου ρολογιού + 1 και αν ισχύει αυτό το property κάνει PASS. Αν στην περίπτωση όπου το updn_cnt είναι 0 τότε μετά από έναν κύκλο ρολογιού ##1 ελέγχουμε αν τα data_out είναι ίσα με του προηγούμενου κύκλου ρολογιού – 1 αν ισχύει αυτό το property κάνει PASS διαφορετικά FAIL.

Testing των property Up-Down Counter

Στην παρακάτω εικόνα φαίνεται η υλοποίηση του αρχείου όπου κάνουμε simulation με τα property

```
1   `timescale  1ns/1ns
2   module updownCOunter16bit_test_with_bind;
3           logic [15:0] data_in;
4           logic rst_, ld_cnt, updn_cnt, count_enb, clk;
5           logic [15:0] data_out;
6           /*
7   module upDownCounter16bit (
8       data_in,
9       rst_,
10      ld_cnt,
11      updn_cnt,
12      count_enb,
13      clk,
14      data_out
15  );*/
16          upDownCounter16bit counter(.data_in(data_in), .rst_(rst_), .ld_cnt(ld_cnt), .updn_cnt(updn_cnt), .count_enb(count_enb), .clk(clk), .data_out(data_out));
17
18          bind upDownCounter16bit upDownCounter16bit_property dut(data_in, rst_, ld_cnt, updn_cnt, count_enb, clk, data_out);
19
20      always @(posedge clk)
21          $display($stime,,,"clk=%b\t rst_= %b\t ld_cnt=%b\t count_enb=%b\t updn_cnt=%b\t data_in=%b\t\t data_out=%b", clk, rst_, ld_cnt, count_enb, updn_cnt, data_in, data_out);
22
23      always #5 clk = ~clk;
24
25      initial begin
26          rst_ = 1'b1 ;        // set reset off
27          ld_cnt = 1'b1;      // set load off
28          count_enb = 1'b0;   // set off counter enable
29          clk = 1'b0;         // set clk at 0
30          rst_ = 1'b0;
31
32       @(posedge clk) rst_ = 1'b1; data_in = 2; ld_cnt = 1'b0;
33       @(posedge clk) count_enb = 1'b1; ld_cnt = 1'b1; updn_cnt = 1'b1; rst_ = 1'b1;
34       @(posedge clk) rst_ = 1'b1; count_enb = 1'b1; ld_cnt = 1'b1; updn_cnt = 1'b0;
35       @(posedge clk);
36       @(posedge clk);
37      end
38  endmodule
```

Στην γραμμή 16 κάνω δήλωση του module upDownCounter16bit counter() με την κατάλληλη αντιστοίχηση των εισόδων/εξόδων καθώς στην γραμμή 18 κάνω bind μεταξύ του module upDownCounter16bit και του upDownCounter16bit_property dut(..) την αντιστοίχιση του module upDownCounter16bit & του property αρχείου.

Στην γραμμή 20 υπάρχει ένα always block για την εκτύπωση στην κονσόλα τα σήματά

Στην γραμμή 25 στο initial block ξεκινάμε με την αρχικοποίηση τον

1. Rst_
2. Ld_cnt
3. Count_enb
4. Clk

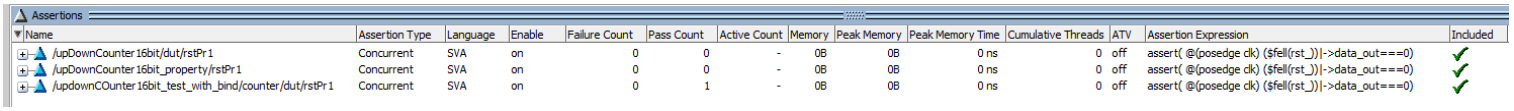
Και στην συνέχεια την ενεργοποίηση του rst_ .

Για το 1ο Property γράφουμε στο Transcript την εξής εντολή

vlog -sv upDownCounter16bit_property.sv +define+check_rst

Τρέχοντας τώρα το simulation για 100ns προκύπτουν οι εξής κυματομορφές

φαίνεται στον χρόνο 0 ns έως 5ns το rst_ είναι ενεργό (active low) και η έξοδος των δεδομένων data_out = 0_{10} και η έξοδος αν έκανε το property pr1 PASS ή FAIL φαίνεται στην Εικόνα 8 και στην Εικόνα 9 είναι η έξοδος του Transcript



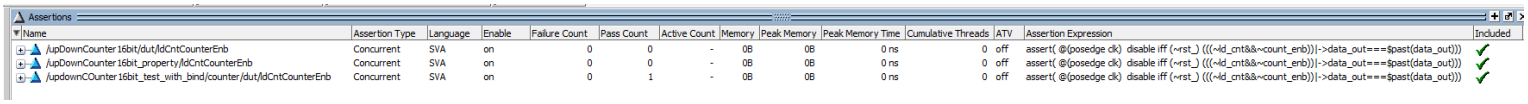

```
VSIM 9> run
#          5  clk=1  rst_= 0  ld_cnt=1  count_enb=0  updn_cnt=x  data_in=xxxxxxxxxxxxxxxx  data_out=0000000000000000
#          5    updownCOunter16bit_test_with_bind.counter.dut.rstPr1 PASS
#         15  clk=1  rst_= 1  ld_cnt=0  count_enb=0  updn_cnt=x  data_in=0000000000000010  data_out=0000000000000000
#         25  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=1  data_in=0000000000000010  data_out=0000000000000010
#         35  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=0000000000000011
#         45  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=0000000000000010
#         55  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=0000000000000001
#         65  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=0000000000000000
#         75  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=1111111111111111
#         85  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=1111111111111110
#         95  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=1111111111111101
```

Για το 2ο Property γράφουμε στο Transcript την εξής εντολή

vlog -sv upDownCounter16bit_property.sv
+define+check_ld_cnt_and_count_en_notEnable

Τρέχοντας τώρα το simulation για 100ns προκύπτουν οι εξής κυματομορφές

φαίνεται στον χρόνο 0ns το ld_cnt είναι ανενεργό (active low) και το count_enb είναι ανενεργό (active high) όμως έχει προηγηθεί ο μηδενισμός της εξόδου data_out 0ns άρα στον επόμενο κύκλο ρολογιού 15ns θα δει αν η τιμή του data_out ήταν ίδια με εκείνη του προηγούμενου κύκλου 0ns και έτσι κάνει PASS το property pr2 φαίνεται στην Εικόνα 10 και στην Εικόνα 11 είναι η έξοδος του Transcript




```
VSIM 14> run
#          5  clk=1  rst_= 0  ld_cnt=1  count_enb=0  updn_cnt=x  data_in=xxxxxxxxxxxxxxxx  data_out=0000000000000000
#         15  clk=1  rst_= 1  ld_cnt=0  count_enb=0  updn_cnt=x  data_in=0000000000000010  data_out=0000000000000000
#         15     updownCOunter16bit_test_with_bind.counter.dut.ldCntCounterEnb PASS
#         25  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=1  data_in=0000000000000010  data_out=0000000000000010
#         35  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=0000000000000011
#         45  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=0000000000000010
#         55  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=0000000000000001
#         65  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=0000000000000000
#         75  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=1111111111111111
#         85  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=1111111111111110
#         95  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=1111111111111101
```

Για το 3ο Property γράφουμε στο Transcript την εξής εντολή

vlog -sv upDownCounter16bit_property.sv
+define+check_ld_cnt_notEnable_and_countEnb_and_updncntEnb

Τρέχοντας τώρα το simulation για 100ns προκύπτουν οι εξής κυματομορφές

φαίνεται στον χρόνο 35ns το ld_cnt είναι ανενεργό (active low) και το count_enb είναι ενεργό (active high) και το updn_cnt = 0 αρα ο counter θα αφαιρεί έτσι το property pr3 κάνει PASS όπως φαίνεται η έξοδος του Transcript

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression
/upDownCounter16bit/dut/ldCntCountrEnbUpCnt	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_) (((ld_cnt&&count_enb))|->if (updn_cnt) (1 ##[1]data_out==.
/upDownCounter16bit_property/ldCntCountrEnbUpCnt	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_) (((ld_cnt&&count_enb))|->if (updn_cnt) (1 ##[1]data_out==.
/updownCOunter16bit_test_with_bind/counter/dut/ldCntCountrEnbUpCnt	Concurrent	SVA	on	1	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (~rst_) (((ld_cnt&&count_enb))|->if (updn_cnt) (1 ##[1]data_out==.

```
VSIM 29> run
#          5  clk=1  rst_= 0  ld_cnt=1  count_enb=0  updn_cnt=x  data_in=xxxxxxxxxxxxxxxx  data_out=0000000000000000
#         15  clk=1  rst_= 1  ld_cnt=0  count_enb=0  updn_cnt=x  data_in=0000000000000010  data_out=0000000000000000
#         25  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=1  data_in=0000000000000010  data_out=0000000000000010
#         35  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=0000000000000011
#         35    updownCOunter16bit_test_with_bind.counter.dut.ldCntCountrEnbUpCnt PASS
#         45  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=0000000000000010
#         45    updownCOunter16bit_test_with_bind.counter.dut.ldCntCountrEnbUpCnt PASS
#         55  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=0000000000000001
#         55    updownCOunter16bit_test_with_bind.counter.dut.ldCntCountrEnbUpCnt PASS
#         65  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=0000000000000000
#         65    updownCOunter16bit_test_with_bind.counter.dut.ldCntCountrEnbUpCnt PASS
#         75  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=1111111111111111
#         75    updownCOunter16bit_test_with_bind.counter.dut.ldCntCountrEnbUpCnt FAIL
#         85  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=1111111111111110
#         85    updownCOunter16bit_test_with_bind.counter.dut.ldCntCountrEnbUpCnt PASS
#         95  clk=1  rst_= 1  ld_cnt=1  count_enb=1  updn_cnt=0  data_in=0000000000000010  data_out=1111111111111101
#         95    updownCOunter16bit_test_with_bind.counter.dut.ldCntCountrEnbUpCnt PASS
```

Επίσης φαίνεται χαρακτηριστικά όταν φτάνει στην ελάχιστη τιμή στον χρόνο 65ns δηλαδή 0_{10} τότε γίνεται ανάθεση στην έξοδο η μέγιστη τιμή 16'hffff γιατί δεν γίνεται κάποια αφαίρεση αλλά ανάθεση με την μεγαλύτερη τιμή που μπορεί να διαχειριστεί ο counter.

Σύγχρονη FIFO

Η sync FIFO αποτελείται από τα εξής

- 16-bit fifo_data_in
- 16-bit fifo_data_out
- Ασύγχρονο rst_
- Σήμα ενεργοποίησης για την εγγραφή δεδομένων από την είσοδο στην εσωτερική μνήμη fifo_write
- Σήμα ενεργοποίησης για την ανάγνωση δεδομένων από την εσωτερική μνήμη της FIFO με όνομα fifo_read
- Σήμα flag αν η FIFO είναι γεμάτη fifo_full
- Σήμα flag αν η FIFO είναι άδεια fifo_empty
- Τέλος, το ρολόι της FIFO clk

Πρώτου δούμε το fifo module πρώτα αναλύουμε τον μετρητή cnt που χρειάστηκε για να μετράμε όταν γράφουμε και όταν διαβάζουμε από την fifo ώστε να ενημερώνουμε καταλληλά τα flags fifo_full & fifo_empty

RTL-Code Counter (5-bit)

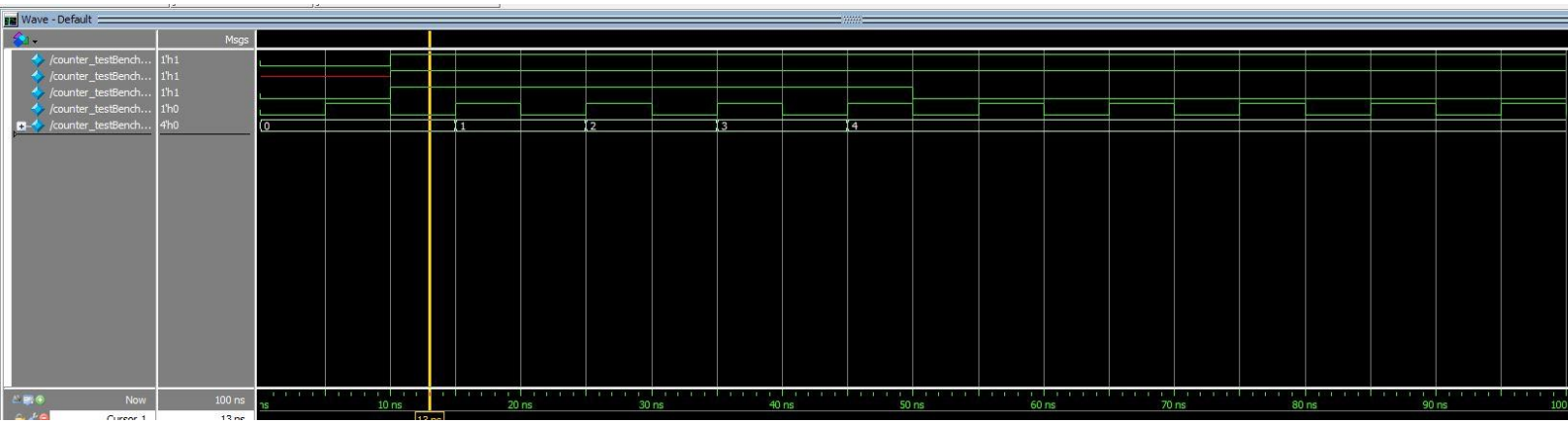
```
1   `timescale 1 ns / 1 ns
2   module counter (
3       rst_,
4       updn_cnt,
5       count_enb,
6       clk,
7       data_out
8   );
9     input logic rst_, updn_cnt, count_enb, clk;
10    output logic [4:0] data_out;
11
12    always_ff @(posedge clk, negedge rst_)begin
13      if(!rst_)begin  // active low reset
14        data_out <= 0;
15      end
16      else if(count_enb == 1)begin
17        if(updn_cnt == 1)begin
18              data_out <= data_out + 1;
19        end else if(updn_cnt == 0) begin
20              data_out <= data_out - 1;
21        end
22      end
23    end
24
25  endmodule
26
```

Καθώς όταν έχουμε θετική ακμή του ρολογιού και το ανάλογο το σήμα updn_cnt τότε μετράει προς τα πάνω ή προς τα κάτω .

TestBench Counter(5-bit)

```
`timescale  1ns/1ns
module counter_testBench;
    logic rst_,updn_cnt,count_enb,clk;
    logic [3:0] data_out;


    initial begin

        $display ("time\t clk\t rst_\t updn_cnt\t count_enb \t\tdata_out");
        $monitor ("%g\t %b\t %b\t\t%b\t%b\t%b", $time, clk, rst_, updn_cnt, count_enb, data_out);

        clk = 0;
        rst_ = 1'b0; // start with reset
        count_enb = 1'b0;

        #10

        rst_ = 1'b1;
        updn_cnt = 1'b1;
        count_enb = 1'b1;

        #40
        rst_ = 1'b1;
        count_enb = 1'b0;

    end

always #5 clk = ~clk;
counter cnt(.*);
initial  // Dump waveform for debug
$dumpvars;
endmodule
```

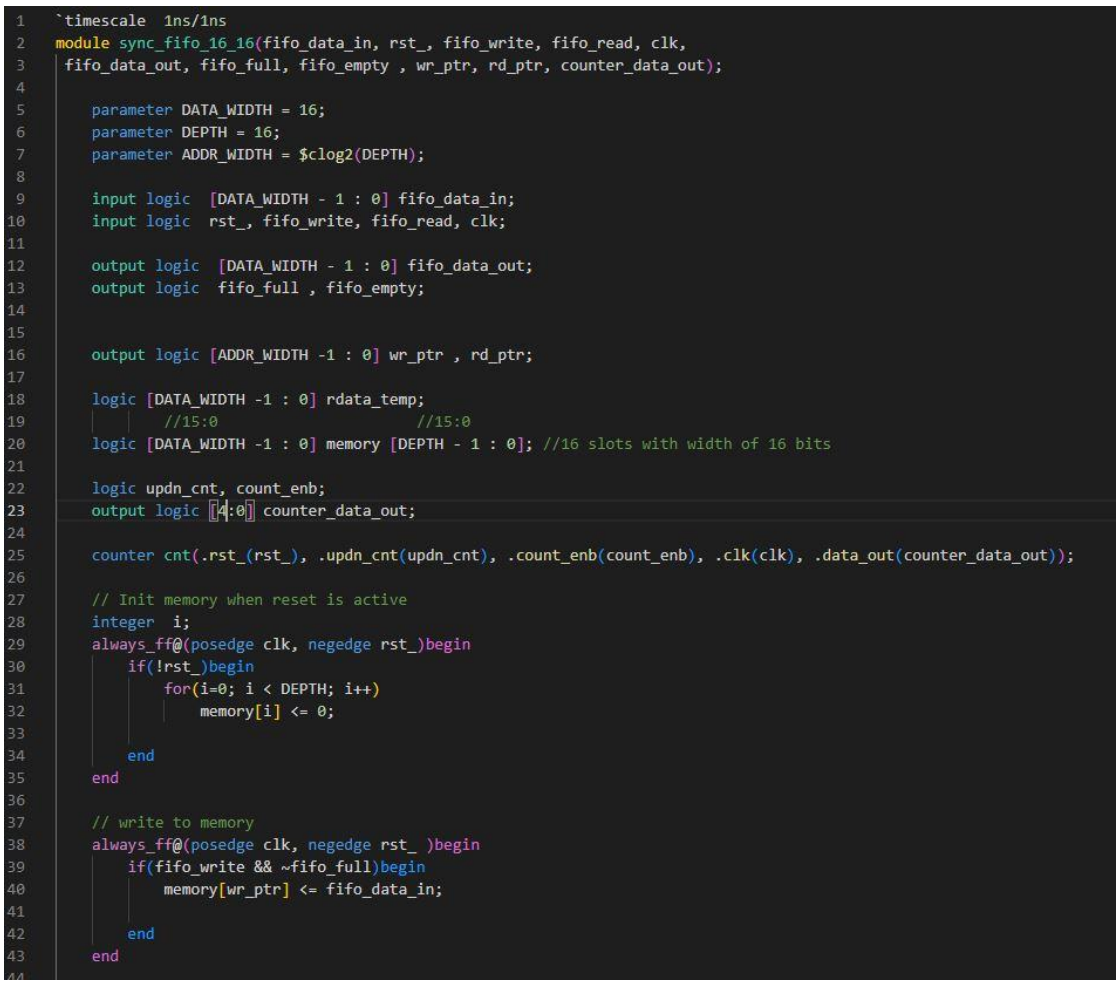
Όπως φαίνεται από την παραπάνω εικόνα καθώς στην αρχή ενεργοποιούμε το rst_ = 0 (active low) και μετά από 10 t.u. το updn_cnt γίνεται 1 και έτσι θα μετράει προς τα πάνω κατά ένα σε συνδυασμό με το count_enb. Μετά από 30 t.u. απενεργοποιείται ο μετρητής φαίνονται οι κυματομορφές του testbench και η έξοδος του transcript.



time	clk	rst_	updn_cnt	count_enb	data_out
0	0	0	x	0	0000
5	1	0	x	0	0000
10	0	1	1	1	0000
15	1	1	1	1	0001
20	0	1	1	1	0001
25	1	1	1	1	0010
30	0	1	1	1	0010
35	1	1	1	1	0011
40	0	1	1	1	0011
45	1	1	1	1	0100
50	0	1	1	0	0100
55	1	1	1	0	0100
60	0	1	1	0	0100
65	1	1	1	0	0100
70	0	1	1	0	0100
75	1	1	1	0	0100
80	0	1	1	0	0100
85	1	1	1	0	0100
90	0	1	1	0	0100
95	1	1	1	0	0100

Φαίνεται η ορθή λειτουργία του counter module.

RTL-Code sync FIFO

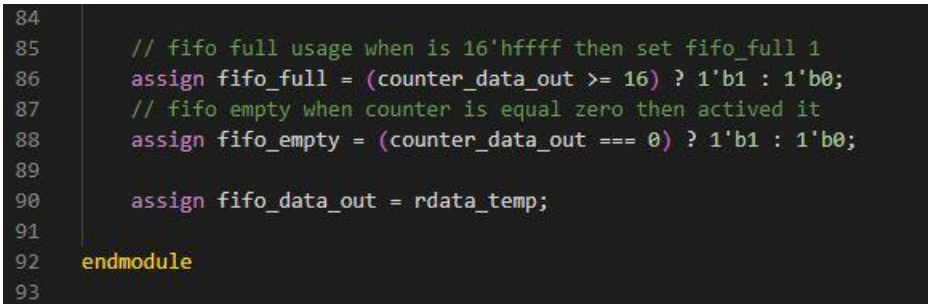


```
1  `timescale  1ns/1ns
2  module sync_fifo_16_16(fifo_data_in, rst_, fifo_write, fifo_read, clk,
3   fifo_data_out, fifo_full, fifo_empty , wr_ptr, rd_ptr, counter_data_out);
4
5      parameter DATA_WIDTH = 16;
6      parameter DEPTH = 16;
7      parameter ADDR_WIDTH = $clog2(DEPTH);
8
9      input logic  [DATA_WIDTH - 1 : 0] fifo_data_in;
10     input logic  rst_, fifo_write, fifo_read, clk;
11
12     output logic  [DATA_WIDTH - 1 : 0] fifo_data_out;
13     output logic  fifo_full , fifo_empty;
14
15
16     output logic [ADDR_WIDTH -1 : 0] wr_ptr , rd_ptr;
17
18     logic [DATA_WIDTH -1 : 0] rdata_temp;
19           //15:0                    //15:0
20     logic [DATA_WIDTH -1 : 0] memory [DEPTH - 1 : 0]; //16 slots with width of 16 bits
21
22     logic updn_cnt, count_enb;
23     output logic [4:0] counter_data_out;
24
25     counter cnt(.rst_(rst_), .updn_cnt(updn_cnt), .count_enb(count_enb), .clk(clk), .data_out(counter_data_out));
26
27     // Init memory when reset is active
28     integer  i;
29     always_ff@(posedge clk, negedge rst_)begin
30         if(!rst_)begin
31             for(i=0; i < DEPTH; i++)
32                 memory[i] <= 0;
33
34         end
35     end
36
37     // write to memory
38     always_ff@(posedge clk, negedge rst_ )begin
39         if(fifo_write && ~fifo_full)begin
40             memory[wr_ptr] <= fifo_data_in;
41
42         end
43     end
44
```

Έχω δηλώση ως parameter τα μεγέθη του πλάτους των δεδομένων και το βάθος της μνήμης καθώς και το μέγεθος για το pointer με της διεύθνσης με την εντολή $clog2(DEPTH) είναι ο λογάριθμος με βάση του 2 και ως όρισμα το DEPTH όπου είναι $\log_2 16 = 4$ που είναι το μέγεθος του pointer για να μας δείξουν κατάλληλα τις θέσεις των δεδομένων μας. Στην γραμμή 20 φαίνεται η δήλωση της μνήμης memory η οποία περιέχει λέξεις τον 16-bit και βάθος 16 θέσεων

Στην γραμμή 25 φαίνεται η δήλωση του module counter που αναλύθηκε πριν με όνομα cnt

```
45      //Default val of read data when reset
46      always_ff@(posedge clk, negedge rst_)begin
47          if(!rst_)begin
48              rdata_temp <= 0;
49          end
50      end
51
52      //Read from Fifo
53      always_ff@(posedge clk, negedge rst_ )begin
54          if(fifo_read && ~fifo_empty)begin
55              rdata_temp <= memory[rd_ptr];
56          end
57      end
58
59      // write pointer handler
60      always_ff@(posedge clk, negedge rst_)begin
61          if (!rst_)begin
62              wr_ptr <= 0;
63              count_enb <= 0;
64          end
65          else if(fifo_write && ~fifo_full)begin
66              wr_ptr <= wr_ptr + 1;
67              updn_cnt <= 1;
68              count_enb <= 1;
69          end
70      end
71       // read pointer handler
72      always_ff@(posedge clk, negedge rst_)begin
73          if(!rst_)begin
74              rd_ptr <= 0;
75              count_enb <= 0;
76          end
77          else if(fifo_read && ~(fifo_empty))begin
78              rd_ptr <= rd_ptr + 1;
79              updn_cnt <= 0;
80              count_enb <= 1;
81          end
82      end
83
```



```
84
85      // fifo full usage when is 16'hffff then set fifo_full 1
86      assign fifo_full = (counter_data_out >= 16) ? 1'b1 : 1'b0;
87      // fifo empty when counter is equal zero then actived it
88      assign fifo_empty = (counter_data_out === 0) ? 1'b1 : 1'b0;
89
90      assign fifo_data_out = rdata_temp;
91
92  endmodule
93
```

Στην γραμμή 28– 35 αποτελείται από ένα always_ff block το οποίο είναι υπεύθυνο για όταν συμβεί το reset(rst_ active low) να γίνει η κατάλληλη επαναφορά της μνήμης memory της FIFO.

Στην γραμμή 38 –43 αποτελείται από ένα always_ff block το οποίο είναι υπεύθυνο για να γραφεί στην μνήμη καθώς αν είναι ενεργό το σήμα εισόδου fifo_write και δεν είναι γεμάτη η μνήμη.

Στην γραμμή 46 – 50 αποτελείται από ένα always_ff block το οποίο είναι υπεύθυνο για να παρέχει την κατάλληλη ανάθεση στη προσωρινή variable τύπου logic με όνομα rdata_temp

Στην γραμμή 53 – 57 αποτελείται από ένα always_ff block το οποίο είναι υπεύθυνο για να γίνεται το read από την FIFO καθώς παρέχει στην μεταβλητή τύπου logic με όνομα rdata_temp τα δεδομένα από την μνήμη στην θέση που δείχνει ο rd_ptr πρώτου όμως γίνει ανάθεση γίνεται έλεγχος αν το σήμα fifo_read είναι ενεργό και η μνήμη δεν είναι άδεια.

Στην γραμμή 60 – 70 αποτελείται από ένα always_ff block το οποίο είναι υπεύθυνο για να διαχειρίζεται τις αλλαγές του wr_ptr. Δηλαδή όταν το rst_ είναι ενεργό (active low) τότε ο wr_ptr δείχνει την θέση μηδέν στην μνήμη διαφορετικά όταν το rst_ είναι ανενεργό ελέγχουμε αν το σήμα fifo_write είναι ενεργό και αν η μνήμη δεν είναι γεμάτη fifo_full οπότε με την κάθε νέα εγγραφή αυξάνετε ο δείκτης εγγραφής wr_ptr. Επιπλέον διαχειρίζεται τον μετρητή ώστε να τον ενεργοποιεί και να μετράει προς τα πάνω

Στην γραμμή 72 – 82 αποτελείται από ένα always_ff block το οποίο είναι υπεύθυνο για να διαχειρίζεται τις αλλαγές του rd_ptr. Δηλαδή όταν το rst_ είναι ενεργό (active low) τότε ο rd_ptr δείχνει την θέση μηδέν στην μνήμη διαφορετικά όταν το rst_ είναι ανενεργό ελέγχουμε αν το σήμα fifo_read είναι ενεργό και αν η μνήμη δεν είναι άδεια fifo_empty οπότε με την κάθε νέα ανάγνωση αυξάνετε ο δείκτης ανάγνωσης rd_ptr. Επιπλέον διαχειρίζεται τον μετρητή ώστε να τον ενεργοποιεί και να μετράει προς τα κάτω

Στην γραμμή 86 γίνεται μια συνεχής ανάθεση assign για το flag fifo_full καθώς αν ο μετρητής cnt φτάσει στην μέγιστη τιμή (16_{10}) τότε το σήμα αυτό ενεργοποιείται

Στην γραμμή 88 γίνεται μια συνεχής ανάθεση assign για το flag fifo_empty καθώς αν ο μετρητής cnt φτάσει στην ελάχιστη τιμή (0_{10}) τότε το σήμα αυτό ενεργοποιείται.

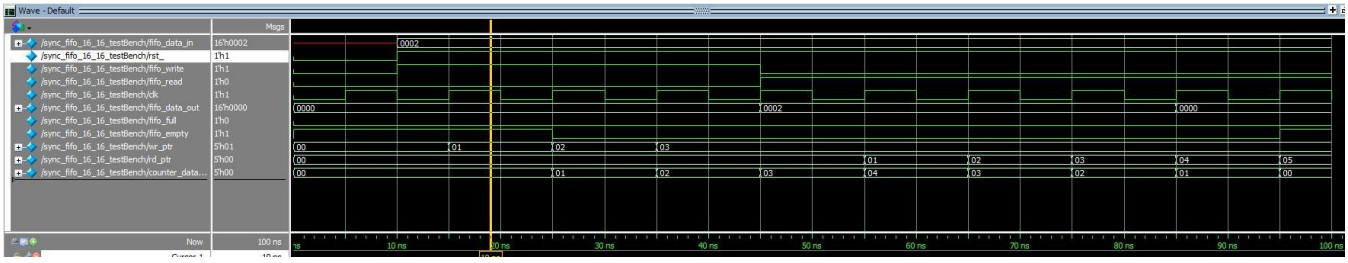
Στην γραμμή 90 γίνεται η συνεχής ανάθεση assign για να παρέχει την έξοδο της FIFO.

Testbench sync FIFO

```
1  `timescale  1ns/1ns
2  module sync_fifo_16_16_testBench;
3
4      parameter DATA_WIDTH = 16;
5      parameter DEPTH = 16;
6      parameter ADDR_WIDTH = $clog2(DEPTH) +1;
7
8      logic  [DATA_WIDTH - 1 : 0] fifo_data_in;   // input
9      logic  rst_, fifo_write, fifo_read, clk;    // input
10
11     logic  [DATA_WIDTH - 1 : 0] fifo_data_out;  // output
12
13     logic fifo_full, fifo_empty;
14
15     logic [ADDR_WIDTH -1 : 0] wr_ptr , rd_ptr;
16     logic [4:0] counter_data_out;
17
18     initial begin
19
20     $display ("time\t clk\t rst_\t fifo_write\t fifo_read\t fifo_full\t fifo_empty\t\t fifo_data_in\t\t\tfifo_data_out");
21     $monitor ("%g\t %b\t %b\t\t%b\t%b\t %b\t\t%b\t%b\t\t\t%b ", $time, clk, rst_, fifo_write, fifo_read, fifo_full, fifo_empty, fifo_data_in, fifo_data_out);
22
23     clk = 0;
24     rst_ = 1'b0; // start with reset
25     fifo_read = 1'b0;
26     fifo_write = 1'b0;
27
28     #10
29
30     rst_ = 1'b1; //reset off
31     fifo_data_in = 2;
32     fifo_write = 1'b1;
33     fifo_read = 1'b0;
34
35     #35
36
37     rst_ = 1'b1; //reset off
38     fifo_write = 1'b0;  //write off
39     fifo_read = 1'b1; // start reading
40
41     end
42
43 always #5 clk = ~clk;
44 sync_fifo_16_16 u0(.*);
45 endmodule
46
```

Παρόμοια δήλωση όπως πριν από την γραμμή 20 έως 21 έπειτα χρησιμοποιούμε το $display για να παρέχουμε μια έξοδο στο transcript σε συνδυασμό με το $monitor που εκτυπώνει κάθε φορά που γίνεται μια αλλαγή στα σήματα που βρίσκονται μέσα στην παρένθεση. Στην συνέχεια ξεκινάμε από το rst_ να είναι ενεργό (active low) και μετά από 10 t.u. γράφουμε το 2_{10} και μετά από 25 t.u. διαβάζουμε αυτό το δεδομένο.

Στην παρακάτω εικόνα φαίνονται οι κυματομορφές της προσομοίωσης αυτής



Καθώς την χρονική στιγμή τον 10ns παρέχετε στην είσοδο fifo_data_in = 2_{10} την χρονική στιγμή των 45ns ενεργοποιείται η ανάγνωση και έτσι βλέπουμε στην έξοδο fifo_data_out = 2_{10} και μετά από 2 κύκλους ρολογιού 65ns βλέπουμε το flag fifo_empty να ενεργοποιείται.

Στην παρακάτω εικόνα φαίνεται η έξοδος του transcript

# time	clk	rst_	fifo_write	fifo_read	fifo_full	fifo_empty	fifo_data_in	fifo_data_out
# 0	0	0	0	0	0	1	xxxxxxxxxxxxxxxx	0000000000000000
# 5	1	0	0	0	0	1	xxxxxxxxxxxxxxxx	0000000000000000
# 10	0	1	1	0	0	1	0000000000000010	0000000000000000
# 15	1	1	1	0	0	1	0000000000000010	0000000000000000
# 20	0	1	1	0	0	1	0000000000000010	0000000000000000
# 25	1	1	1	0	0	0	0000000000000010	0000000000000000
# 30	0	1	1	0	0	0	0000000000000010	0000000000000000
# 35	1	1	1	0	0	0	0000000000000010	0000000000000000
# 40	0	1	1	0	0	0	0000000000000010	0000000000000000
# 45	1	1	0	1	0	0	0000000000000010	0000000000000010
# 50	0	1	0	1	0	0	0000000000000010	0000000000000010
# 55	1	1	0	1	0	0	0000000000000010	0000000000000010
# 60	0	1	0	1	0	0	0000000000000010	0000000000000010
# 65	1	1	0	1	0	0	0000000000000010	0000000000000010
# 70	0	1	0	1	0	0	0000000000000010	0000000000000010
# 75	1	1	0	1	0	0	0000000000000010	0000000000000010
# 80	0	1	0	1	0	0	0000000000000010	0000000000000010
# 85	1	1	0	1	0	0	0000000000000010	0000000000000000
# 90	0	1	0	1	0	0	0000000000000010	0000000000000000
# 95	1	1	0	1	0	1	0000000000000010	0000000000000000

Testing των property sync FIFO

Στην παρακάτω εικόνα φαίνεται η υλοποίηση του αρχείου όπου κάνουμε simulation με τα property καθώς και τα macro.

sync_fifo_16_16_property.sv > sync_fifo_16_16_property

```
`timescale 1ns/1ns
module sync_fifo_16_16_property(fifo_data_in, rst_, fifo_write, fifo_read, clk,
  fifo_full, fifo_empty, wr_ptr, rd_ptr, counter_data_out);
    parameter DATA_WIDTH = 16;
    parameter DEPTH = 16;
    parameter ADDR_WIDTH = $clog2(DEPTH);
    input logic  [DATA_WIDTH - 1 : 0] fifo_data_in;
    input logic  rst_, fifo_write, fifo_read, clk;
    input logic  fifo_full , fifo_empty;
    input logic [ADDR_WIDTH -1 : 0] wr_ptr , rd_ptr;
    input logic [4:0] counter_data_out;

`ifdef check_rst
    property pr1;
        @(posedge clk) (!rst_) |-> (rd_ptr===0) && (wr_ptr===0) && (counter_data_out===0) && (fifo_empty===1) && (fifo_full===0);
    endproperty

    rstPr1: assert property(pr1) $display($stime,,,"\t\t %m PASS");
            else $display($stime,,,"\t\t %m FAIL");

`elsif check_empty

    property pr2;
        @(posedge clk) disable iff(!rst_) counter_data_out === 0 |-> fifo_empty;
    endproperty

    fifoEmpty: assert property(pr2) $display($stime,,,"\t\t %m PASS");
            else $display($stime,,,"\t\t %m FAIL");

`elsif check_full
    property pr3;
        @(posedge clk) disable iff(!rst_) counter_data_out >= 16 |-> fifo_full;
    endproperty

    checkFull: assert property(pr3) $display($stime,,,"\t\t %m PASS");
            else $display($stime,,,"\t\t %m FAIL");

`elsif check_writePtr
    property pr4;
        @(posedge clk) disable iff(!rst_) fifo_full && fifo_write && !fifo_read |-> $stable(wr_ptr);
    endproperty

    checkWritePtr: assert property (pr4) $display($stime,,,"\t\t %m PASS");
            else $display($stime,,,"\t\t %m FAIL");
```

```
45
46  `elsif check_readPtr
47      property pr5;
48          @(posedge clk) disable iff(!rst_) fifo_empty && fifo_read && !fifo_write |-> $stable(rd_ptr);
49      endproperty
50
51      checkReadPtr: assert property (pr5) $display($stime,,,"\t\t %m PASS");
52                  else $display($stime,,,"\t\t %m FAIL");
53  `endif
54  endmodule
```

Στην παρακάτω εικόνα φαίνεται η υλοποίηση του αρχείου που γίνεται το bind και το testing τον property

```
1   `timescale 1ns/1ns
2   module sync_fifo_16_16_test_with_bind;
3
4       parameter DATA_WIDTH = 16;
5       parameter DEPTH = 16;
6       parameter ADDR_WIDTH = $clog2(DEPTH);
7       logic  [DATA_WIDTH - 1 : 0] fifo_data_in;
8       logic  [DATA_WIDTH - 1 : 0] fifo_data_out;
9       logic  rst_, fifo_write, fifo_read, clk;
10      logic  fifo_full , fifo_empty;
11      logic [ADDR_WIDTH -1 : 0] wr_ptr , rd_ptr;
12      logic [4:0] counter_data_out;
13      sync_fifo_16_16 fifo(.fifo_data_in(fifo_data_in), .rst_(rst_), .fifo_write(fifo_write), .fifo_read(fifo_read), .clk(clk),
14       .fifo_data_out(fifo_data_out), .fifo_full(fifo_full), .fifo_empty(fifo_empty), .wr_ptr(wr_ptr), .rd_ptr(rd_ptr), .counter_data_out(counter_data_out));
15      bind sync_fifo_16_16 sync_fifo_16_16_property dut(fifo_data_in, rst_, fifo_write, fifo_read, clk, fifo_full, fifo_empty, wr_ptr, rd_ptr, counter_data_out);
16      always #5 clk = ~clk;
17      initial begin
18          $display ("time\t clk\t rst_\t fifo_write\t fifo_read\t fifo_full\t fifo_empty\t\t   fifo_data_in\t\t\tfifo_data_out\t\t   wr_ptr\t\t rd_ptr\t\tcounter_data_out");
19          $monitor ("%g\t %b\t %b\t\t%b\t%b\t %b\t\t%b\t%b\t\t%b\t\t%b\t\t%b\t\t%b", $time, clk, rst_, fifo_write, fifo_read, fifo_full, fifo_empty, fifo_data_in, fifo_data_out, wr_ptr,
20           rd_ptr, counter_data_out);
21
22          rst_ = 1'b1;
23          clk = 1'b0;
24          rst_ = 1'b0;
25
26          @(posedge clk); // check propert pr1
27          @(posedge clk) rst_ = 1'b1;  fifo_write = 0; fifo_read = 0;     // check property pr2
28          @(posedge clk);
29
30          @(posedge clk) rst_ = 1'b1;  fifo_write = 0; fifo_read = 1;           // check property pr5
31          @(posedge clk) rst_ = 1'b1; fifo_write = 0; fifo_read = 0;
32
33          @(posedge clk) for(int i = 0; i < 16; i++) begin                  // write all over 16 slots to check property pr3
34                  fifo_write = 1; fifo_read = 0; fifo_data_in = 1;
35          end
36          @(posedge clk) rst_ = 1'b1; fifo_write = 1; fifo_read = 0;     // check property pr4
37      end
38      initial  // Dump waveform for debug
39      $dumpvars;
40
41  endmodule
```

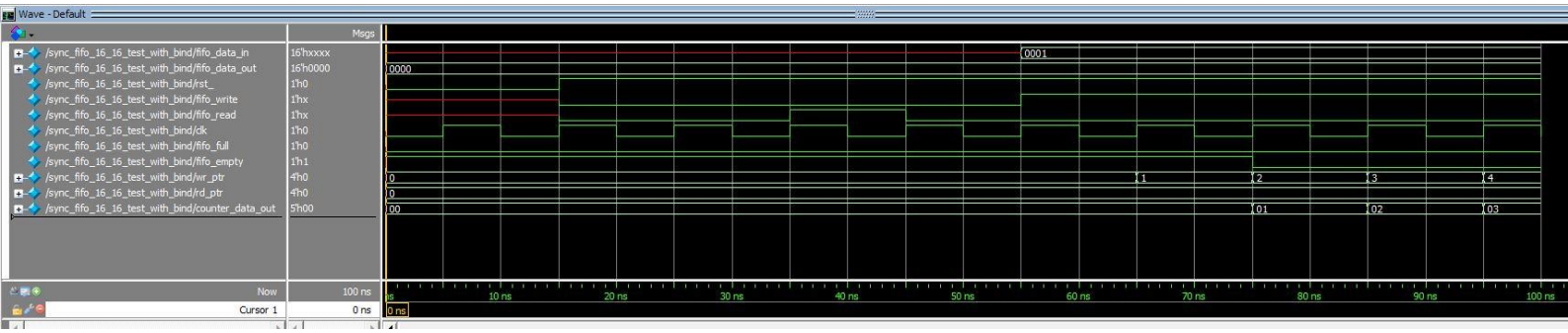
Στην γραμμή 13 γίνεται η δήλωση του module που είναι η fifo μας καθώς στην γραμμή 15 γίνεται το bind μεταξύ fifo module & property module.

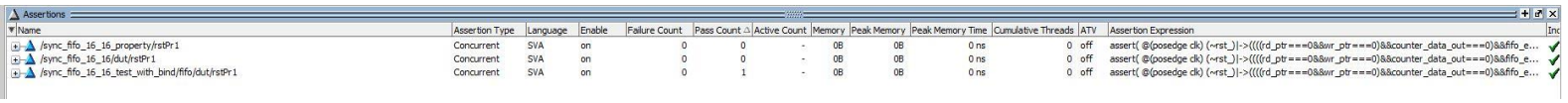
Στην αρχή ξεκινάμε με την αρχικοποίηση των rst_, clk και στην συνέχεια την ενεργοποίηση του rst_ . Μετά από έναν κύκλο ρολογιού θέτουμε το rst_ off και το fifo_write = 0, fifo_read = 0. Στην γραμμή 30 δηλώνουμε ενεργό το fifo_read ώστε να δούμε πως λειτουργεί το κατάλληλο property pr5 έπειτα στην γραμμή 33 κάνουμε 16 εγγραφές στην μνήμη για να τσεκάρουμε αν λειτουργεί σωστά το property pr3 και τέλος για να δούμε το property pr4 πάμε να κάνουμε write σε γεμάτη FIFO.

Για το 1ο Property γράφουμε στο Transcript την εξής εντολή

vlog -sv sync_fifo_16_16_property.sv +define+check_rst

προκύπτουν οι ίδιες κυματομορφές όπως παραπάνω αλλά και τα property και την έξοδο του transcript





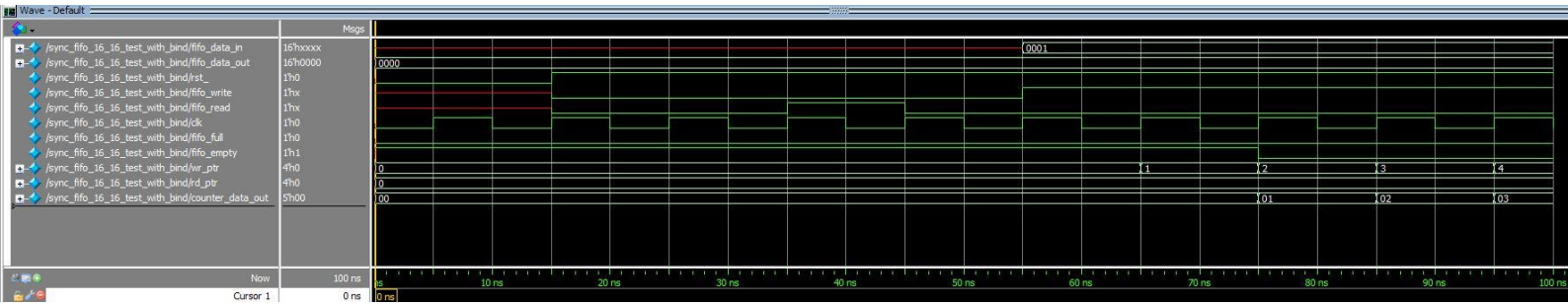

```
# time    clk     rst_    fifo_write  fifo_read  fifo_full  fifo_empty    fifo_data_in      fifo_data_out     wr_ptr    rd_ptr    counter_data_out
# 0       0       0       x           x          0          1             xxxxxxxxxxxxxxxx  0000000000000000  0000      0000      00000
#          5      sync_fifo_16_16_test_with_bind.fifo.dut.rstPr1  PASS
# 5       1       0       x           x          0          1             xxxxxxxxxxxxxxxx  0000000000000000  0000      0000      00000
# 10      0       0       x           x          0          1             xxxxxxxxxxxxxxxx  0000000000000000  0000      0000      00000
#          15     sync_fifo_16_16_test_with_bind.fifo.dut.rstPr1  PASS
# 15      1       1       0           0          0          1             xxxxxxxxxxxxxxxx  0000000000000000  0000      0000      00000
# 20      0       1       0           0          0          1             xxxxxxxxxxxxxxxx  0000000000000000  0000      0000      00000
# 25      1       1       0           0          0          1             xxxxxxxxxxxxxxxx  0000000000000000  0000      0000      00000
# 30      0       1       0           0          0          1             xxxxxxxxxxxxxxxx  0000000000000000  0000      0000      00000
# 35      1       1       0           1          0          1             xxxxxxxxxxxxxxxx  0000000000000000  0000      0000      00000
# 40      0       1       0           1          0          1             xxxxxxxxxxxxxxxx  0000000000000000  0000      0000      00000
# 45      1       1       0           0          0          1             xxxxxxxxxxxxxxxx  0000000000000000  0000      0000      00000
# 50      0       1       0           0          0          1             xxxxxxxxxxxxxxxx  0000000000000000  0000      0000      00000
# 55      1       1       1           0          0          1             0000000000000001  0000000000000000  0000      0000      00000
# 60      0       1       1           0          0          1             0000000000000001  0000000000000000  0000      0000      00000
# 65      1       1       1           0          0          1             0000000000000001  0000000000000000  0001      0000      00000
# 70      0       1       1           0          0          1             0000000000000001  0000000000000000  0001      0000      00000
# 75      1       1       1           0          0          0             0000000000000001  0000000000000000  0010      0000      00001
# 80      0       1       1           0          0          0             0000000000000001  0000000000000000  0010      0000      00001
# 85      1       1       1           0          0          0             0000000000000001  0000000000000000  0011      0000      00010
# 90      0       1       1           0          0          0             0000000000000001  0000000000000000  0011      0000      00010
# 95      1       1       1           0          0          0             0000000000000001  0000000000000000  0100      0000      00011
```

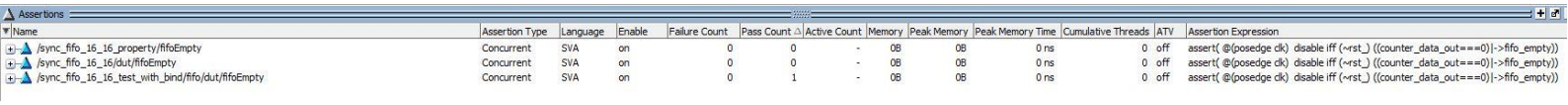
Καθώς την χρονική στιγμή 0 t.u το rst_ είναι ενεργό (active low) και το fifo_full = 0 , fifo_empty = 1 ,wr_ptr = 0 , rd_ptr = 0, counter_data_out = 0 στην θετική ακμή του ρολογιού 5 t.u το property pr1 κάνει PASS εμφανίζεται ένα δεύτερο PASS καθώς η τιμή του rst_ παρέμεινε ενεργή πάνω από έναν κύκλο ρολογιού.

Για το 2ο Property γράφουμε στο Transcript την εξής εντολή

vlog -sv sync_fifo_16_16_property.sv +define+check_empty

και προκύπτουν τα εξής





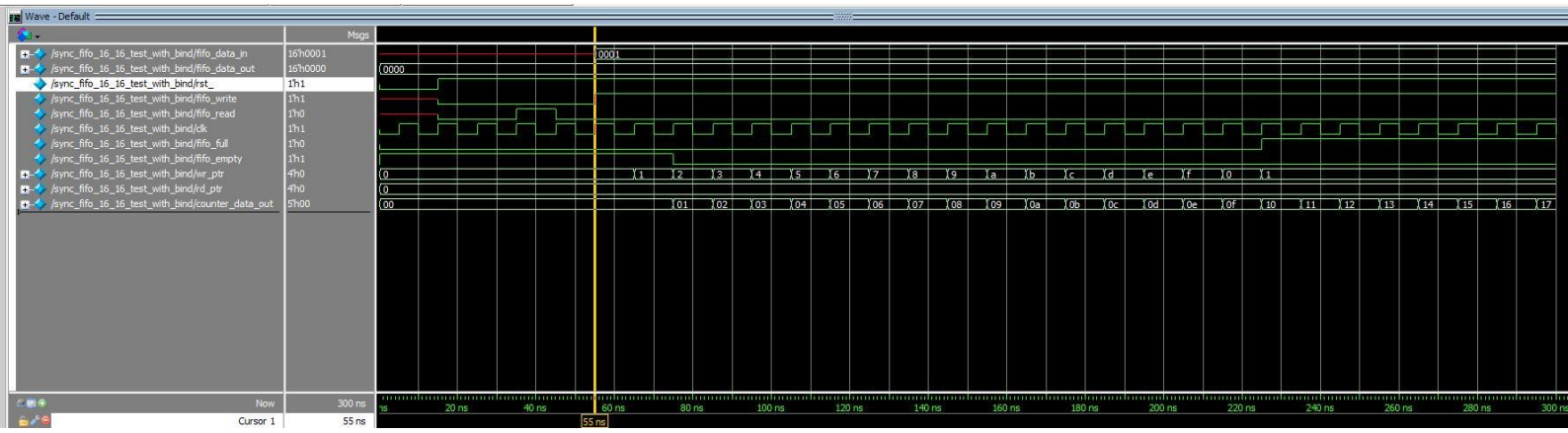

```
# time   clk    rst_    fifo_write  fifo_read  fifo_full  fifo_empty     fifo_data_in     fifo_data_out     wr_ptr        rd_ptr        counter_data_out
# 0      0      0             x          x          0           1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 5      1      0             x          x          0           1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 10     0      0             x          x          0           1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
#        15     sync_fifo_16_16_test_with_bind.fifo.dut.fifoEmpty PASS
# 15     1      1             0          0          0           1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 20     0      1             0          0          0           1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
#        25     sync_fifo_16_16_test_with_bind.fifo.dut.fifoEmpty PASS
# 25     1      1             0          0          0           1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 30     0      1             0          0          0           1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
#        35     sync_fifo_16_16_test_with_bind.fifo.dut.fifoEmpty PASS
# 35     1      1             0          1          0           1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 40     0      1             0          1          0           1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
#        45     sync_fifo_16_16_test_with_bind.fifo.dut.fifoEmpty PASS
# 45     1      1             0          0          0           1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 50     0      1             0          0          0           1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
#        55     sync_fifo_16_16_test_with_bind.fifo.dut.fifoEmpty PASS
# 55     1      1             1          0          0           1      0000000000000001  0000000000000000  0000          0000          00000
# 60     0      1             1          0          0           1      0000000000000001  0000000000000000  0000          0000          00000
#        65     sync_fifo_16_16_test_with_bind.fifo.dut.fifoEmpty PASS
# 65     1      1             1          0          0           1      0000000000000001  0000000000000000  0001          0000          00000
# 70     0      1             1          0          0           1      0000000000000001  0000000000000000  0001          0000          00000
#        75     sync_fifo_16_16_test_with_bind.fifo.dut.fifoEmpty PASS
# 75     1      1             1          0          0           0      0000000000000001  0000000000000000  0010          0000          00001
# 80     0      1             1          0          0           0      0000000000000001  0000000000000000  0010          0000          00001
# 85     1      1             1          0          0           0      0000000000000001  0000000000000000  0011          0000          00010
# 90     0      1             1          0          0           0      0000000000000001  0000000000000000  0011          0000          00010
# 95     1      1             1          0          0           0      0000000000000001  0000000000000000  0100          0000          00011
```

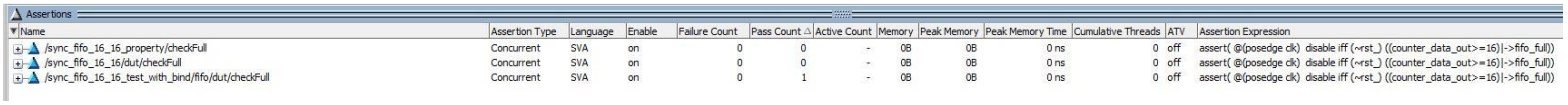
Καθώς τα counter_data_out === 0 και στον ίδιο κύκλο ρολογιού το flag fifo_empty είναι ενεργό και έτσι το property κάνει PASS

Για το 3ο Property γράφουμε στο Transcript την εξής εντολή

vlog -sv sync_fifo_16_16_property.sv +define+check_full

και προκύπτουν τα εξής





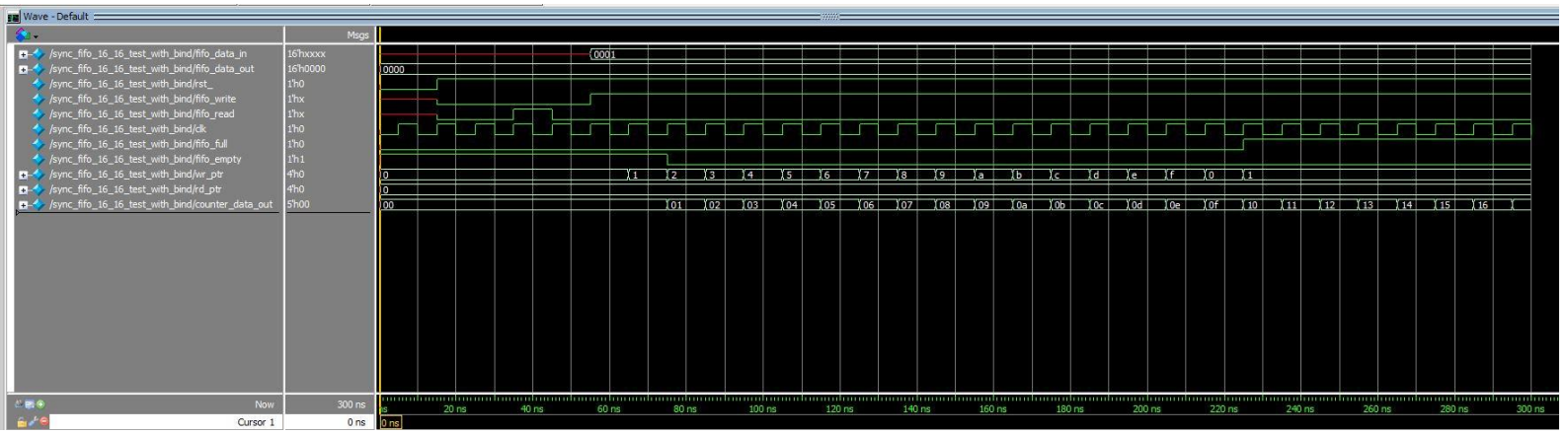
# time	clk	rst_	fifo_write	fifo_read	fifo_full	fifo_empty	fifo_data_in	fifo_data_out	wr_ptr	rd_ptr	counter_data_out
# 0	0	0	x	x	0	1	xxxxxxxxxxxxxxxx	0000000000000000	0000	0000	00000
# 5	1	0	x	x	0	1	xxxxxxxxxxxxxxxx	0000000000000000	0000	0000	00000
# 10	0	0	x	x	0	1	xxxxxxxxxxxxxxxx	0000000000000000	0000	0000	00000
# 15	1	1	0	0	0	1	xxxxxxxxxxxxxxxx	0000000000000000	0000	0000	00000
# 20	0	1	0	0	0	1	xxxxxxxxxxxxxxxx	0000000000000000	0000	0000	00000
# 25	1	1	0	0	0	1	xxxxxxxxxxxxxxxx	0000000000000000	0000	0000	00000
# 30	0	1	0	0	0	1	xxxxxxxxxxxxxxxx	0000000000000000	0000	0000	00000
# 35	1	1	0	1	0	1	xxxxxxxxxxxxxxxx	0000000000000000	0000	0000	00000
# 40	0	1	0	1	0	1	xxxxxxxxxxxxxxxx	0000000000000000	0000	0000	00000
# 45	1	1	0	0	0	1	xxxxxxxxxxxxxxxx	0000000000000000	0000	0000	00000
# 50	0	1	0	0	0	1	xxxxxxxxxxxxxxxx	0000000000000000	0000	0000	00000
# 55	1	1	1	0	0	1	0000000000000001	0000000000000000	0000	0000	00000
# 60	0	1	1	0	0	1	0000000000000001	0000000000000000	0000	0000	00000
# 65	1	1	1	0	0	1	0000000000000001	0000000000000000	0001	0000	00000
# 70	0	1	1	0	0	1	0000000000000001	0000000000000000	0001	0000	00000
# 75	1	1	1	0	0	0	0000000000000001	0000000000000000	0010	0000	00001
# 80	0	1	1	0	0	0	0000000000000001	0000000000000000	0010	0000	00001
# 85	1	1	1	0	0	0	0000000000000001	0000000000000000	0011	0000	00010
# 90	0	1	1	0	0	0	0000000000000001	0000000000000000	0011	0000	00010
# 95	1	1	1	0	0	0	0000000000000001	0000000000000000	0100	0000	00011
# 100	0	1	1	0	0	0	0000000000000001	0000000000000000	0100	0000	00011
# 105	1	1	1	0	0	0	0000000000000001	0000000000000000	0101	0000	00100
# 110	0	1	1	0	0	0	0000000000000001	0000000000000000	0101	0000	00100
# 115	1	1	1	0	0	0	0000000000000001	0000000000000000	0110	0000	00101
# 120	0	1	1	0	0	0	0000000000000001	0000000000000000	0110	0000	00101
# 125	1	1	1	0	0	0	0000000000000001	0000000000000000	0111	0000	00110
# 130	0	1	1	0	0	0	0000000000000001	0000000000000000	0111	0000	00110

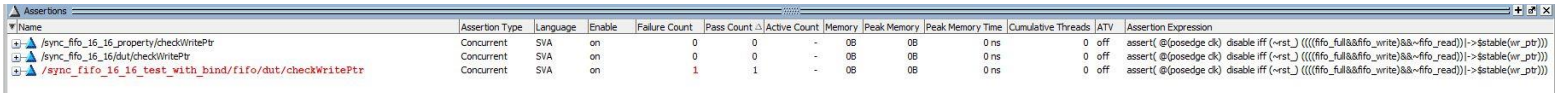
```
# 135    1       1               1       0       0               0       0000000000000001  0000000000000000  1000        0000              00111
# 140    0       1               1       0       0               0       0000000000000001  0000000000000000  1000        0000              00111
# 145    1       1               1       0       0               0       0000000000000001  0000000000000000  1001        0000              01000
# 150    0       1               1       0       0               0       0000000000000001  0000000000000000  1001        0000              01000
# 155    1       1               1       0       0               0       0000000000000001  0000000000000000  1010        0000              01001
# 160    0       1               1       0       0               0       0000000000000001  0000000000000000  1010        0000              01001
# 165    1       1               1       0       0               0       0000000000000001  0000000000000000  1011        0000              01010
# 170    0       1               1       0       0               0       0000000000000001  0000000000000000  1011        0000              01010
# 175    1       1               1       0       0               0       0000000000000001  0000000000000000  1100        0000              01011
# 180    0       1               1       0       0               0       0000000000000001  0000000000000000  1100        0000              01011
# 185    1       1               1       0       0               0       0000000000000001  0000000000000000  1101        0000              01100
# 190    0       1               1       0       0               0       0000000000000001  0000000000000000  1101        0000              01100
# 195    1       1               1       0       0               0       0000000000000001  0000000000000000  1110        0000              01101
VSIM 183> run
# 200    0       1               1       0       0               0       0000000000000001  0000000000000000  1110        0000              01101
# 205    1       1               1       0       0               0       0000000000000001  0000000000000000  1111        0000              01110
# 210    0       1               1       0       0               0       0000000000000001  0000000000000000  1111        0000              01110
# 215    1       1               1       0       0               0       0000000000000001  0000000000000000  0000        0000              01111
# 220    0       1               1       0       0               0       0000000000000001  0000000000000000  0000        0000              01111
# 225    1       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10000
# 230    0       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10000
#        235     sync_fifo_16_16_test_with_bind.fifo.dut.checkFull PASS
# 235    1       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10001
# 240    0       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10001
#        245     sync_fifo_16_16_test_with_bind.fifo.dut.checkFull PASS
# 245    1       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10010
# 250    0       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10010
#        255     sync_fifo_16_16_test_with_bind.fifo.dut.checkFull PASS
# 255    1       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10011
# 260    0       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10011
#        265     sync_fifo_16_16_test_with_bind.fifo.dut.checkFull PASS
# 265    1       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10100
# 270    0       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10100
#        275     sync_fifo_16_16_test_with_bind.fifo.dut.checkFull PASS
# 275    1       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10101

# 280    0       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10101
#        285     sync_fifo_16_16_test_with_bind.fifo.dut.checkFull PASS
# 285    1       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10110
# 290    0       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10110
#        295     sync_fifo_16_16_test_with_bind.fifo.dut.checkFull PASS
# 295    1       1               1       0       1               0       0000000000000001  0000000000000000  0001        0000              10111
```

Η λογική του counter δεν μας εξυπηρετεί καλά γιατί υπάρχουν διαφορές χρόνου μέχρι να γίνει η κατάλληλη αλλαγή και έτσι δεν μπορούμε γρήγορα να αναφέρουμε αν έγινε κάποιο flag ενεργό πιο βολικό θα ήταν να ελέγχαμε τους pointers για να έχουμε μια πιο άμεση απόδοση τον flags full , empty.

Για το 4ο Property γράφουμε στο Transcript την εξής εντολή

vlog -sv sync_fifo_16_16_property.sv +define+check_writePtr





στις προηγούμενες χρονικές στιγμές δεν υπάρχει κάποιο PASS/FAIL από 0 έως 200 t.u.

```
# 200    0    1    1    0    0    0    0000000000000001 0000000000000000 1110 0000 01101
# 205    1    1    1    0    0    0    0000000000000001 0000000000000000 1111 0000 01110
# 210    0    1    1    0    0    0    0000000000000001 0000000000000000 1111 0000 01110
# 215    1    1    1    0    0    0    0000000000000001 0000000000000000 0000 0000 01111
# 220    0    1    1    0    0    0    0000000000000001 0000000000000000 0000 0000 01111
# 225    1    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10000
# 230    0    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10000
#        235  sync_fifo_16_16_test_with_bind.fifo.dut.checkWritePtr FAIL
# 235    1    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10001
# 240    0    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10001
#        245  sync_fifo_16_16_test_with_bind.fifo.dut.checkWritePtr PASS
# 245    1    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10010
# 250    0    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10010
#        255  sync_fifo_16_16_test_with_bind.fifo.dut.checkWritePtr PASS
# 255    1    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10011
# 260    0    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10011
#        265  sync_fifo_16_16_test_with_bind.fifo.dut.checkWritePtr PASS
# 265    1    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10100
# 270    0    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10100
#        275  sync_fifo_16_16_test_with_bind.fifo.dut.checkWritePtr PASS
# 275    1    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10101
# 280    0    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10101
#        285  sync_fifo_16_16_test_with_bind.fifo.dut.checkWritePtr PASS
# 285    1    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10110
# 290    0    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10110
#        295  sync_fifo_16_16_test_with_bind.fifo.dut.checkWritePtr PASS
# 295    1    1    1    0    1    0    0000000000000001 0000000000000000 0001 0000 10111
```

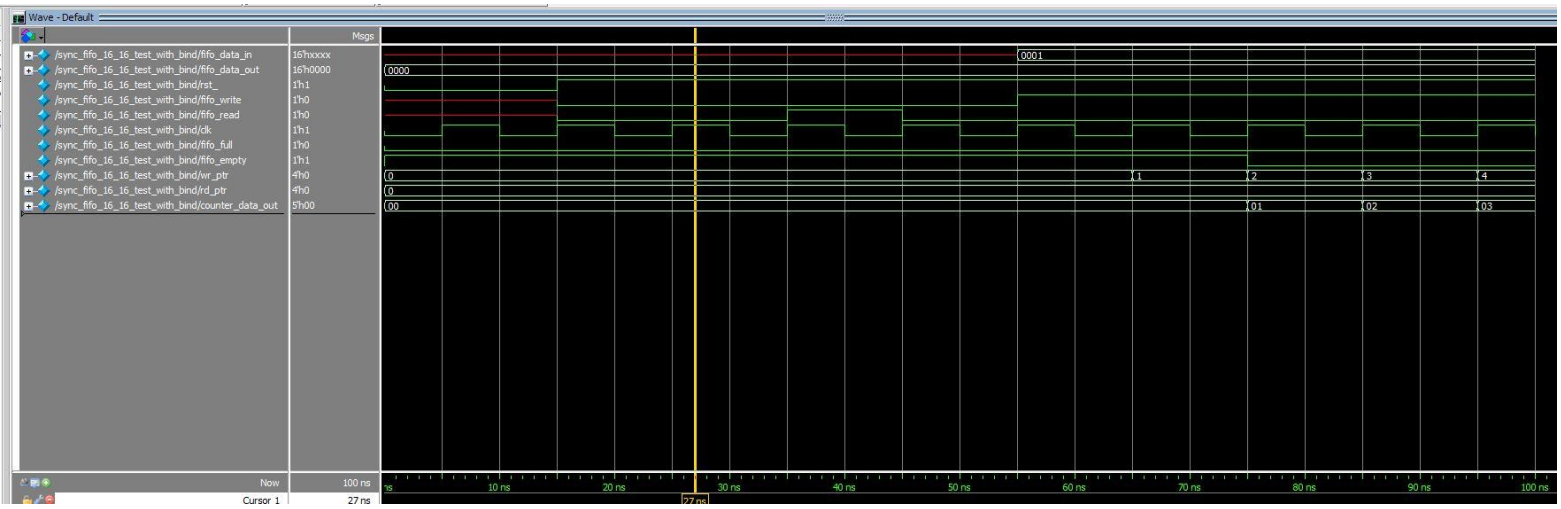
Φαίνεται ότι την χρονική στιγμή 220 με 230 γίνεται αλλαγή του wr_ptr και έτσι το property κάνει FAIL αυτό οφείλεται στην χρονική διαφορά που έχουν τα 2 modules. Στην

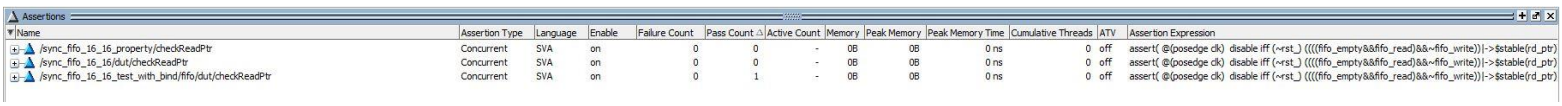
χρονική στιγμή 245 κάνει PASS γιατί δεν αλλάζει το wr_ptr εννοώ και στις 2 περιπτώσεις το fifo_full είναι ενεργό.

Για το 5ο Property γράφουμε στο Transcript την εξής εντολή

vlog -sv sync_fifo_16_16_property.sv +define+check_readPtr

προκύπτουν τα εξής






```
# time   clk    rst_   fifo_write  fifo_read  fifo_full  fifo_empty     fifo_data_in     fifo_data_out     wr_ptr        rd_ptr        counter_data_out
# 0      0      0            x          x          0          1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 5      1      0            x          x          0          1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 10     0      0            x          x          0          1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 15     1      1            0          0          0          1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 20     0      1            0          0          0          1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 25     1      1            0          0          0          1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 30     0      1            0          0          0          1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 35     1      1            0          1          0          1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 40     0      1            0          1          0          1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
#        45     sync_fifo_16_16_test_with_bind.fifo.dut.checkReadPtr PASS
# 45     1      1            0          0          0          1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 50     0      1            0          0          0          1      xxxxxxxxxxxxxxxx  0000000000000000  0000          0000          00000
# 55     1      1            1          0          0          1      0000000000000001  0000000000000000  0000          0000          00000
# 60     0      1            1          0          0          1      0000000000000001  0000000000000000  0000          0000          00000
# 65     1      1            1          0          0          1      0000000000000001  0000000000000000  0001          0000          00000
# 70     0      1            1          0          0          1      0000000000000001  0000000000000000  0001          0000          00000
# 75     1      1            1          0          0          0      0000000000000001  0000000000000000  0010          0000          00001
# 80     0      1            1          0          0          0      0000000000000001  0000000000000000  0010          0000          00001
# 85     1      1            1          0          0          0      0000000000000001  0000000000000000  0011          0000          00010
# 90     0      1            1          0          0          0      0000000000000001  0000000000000000  0011          0000          00010
# 95     1      1            1          0          0          0      0000000000000001  0000000000000000  0100          0000          00011
```

Φαίνεται ότι καθώς έχουμε το fifo_read = 1 και βλέπουμε ότι ο rd_ptr δεν αυξάνεται και επίσης το flag fifo_empty = 1 και έτσι κάνει pass το property.