**Ψηφιακά Συστήματα HW σε Χαμηλά Επίπεδα Λογικής II**

**Πίνακας περιεχομένων**

[1 Εισαγωγή 3](#_Toc110015941)

[2 Μετρητής 3](#_Toc110015942)

[2.1 Κύκλωμα Μετρητή 3](#_Toc110015943)

[2.2 Assertions Μετρητή 4](#_Toc110015944)

[2.3 Kώδικας TestBench 8](#_Toc110015945)

[3 FIFO 10](#_Toc110015946)

[3.1 Κύκλωμα FIFO 10](#_Toc110015947)

[3.2 Assertions FIFO 12](#_Toc110015948)

[3.3 Κώδικας Testbench 17](#_Toc110015949)

Όνομα: ΤΖΟΥΛΙΟ ΤΖΕΛΙΛΑΙ

ΑΕΜ: 9662

# 1 Εισαγωγή

Η εργασία αποτελείται από δύο κύρια κομμάτια τα οποία είναι και τα δύο ψηφιακά κυκλώματα που ζητήθηκε να σχεδιαστούν. Συγκεκριμένα το κάθε κομμάτι χωρίζεται σε τρία βασικά μέρη:

1. Μια σύντομη περιγραφή της σχεδιαστικής λογικής που ακολουθήθηκε για το εκάστοτε κύκλωμα καθώς και η παράθεση του κώδικα
2. Περιγραφή των assertions που υλοποιήθηκαν (αριθμημένα), παρουσίαση των αντίστοιχων screenshots καθώς και ο κώδικας των assertions.
3. Ο Κώδικας που χρησιμοποιήθηκε για το Testbench

Για την ευκολότερη κατανόηση των κωδίκων όσο αφορά τα κυκλώματα και των αντίστοιχων assertions, υπάρχουν επεξηγηματικά κομμάτια σχολίων στον ίδιο τον κώδικα.

# 2 Μετρητής

# 2.1 Κύκλωμα Μετρητή

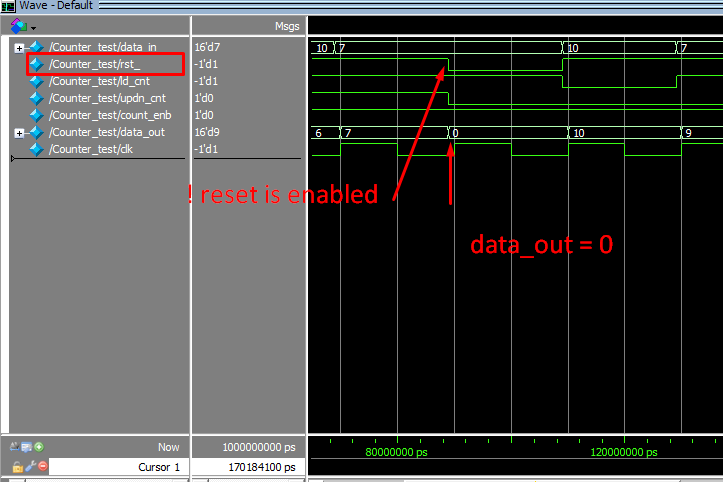
Ο παρακάτω μετρητής αποτελείται κυρίως από ένα always\_ff block το οποίο είναι ευαίσθητο στην θετική ακμή του ρολογιού (clk) καθώς και στην αρνητική ακμή του σήματος reset(rst\_). Η λογική που ακολουθείται στο if-else block είναι του ελέγχου σημάτων βάση προτεραιότητας. Συγκεκριμένα, πρωτίστως ελέγχεται το σήμα reset το οποίο και μηδενίζει τον μετρητή, έπειτα ακολουθεί σε προτεραιότητα το σήμα load counter (ld\_cnt) το οποίο ανανεώνει τον μετρητή με μια συγκεκριμένη αριθμητική τιμή και τέλος ακολουθεί το σήμα counter enable (count\_enb) το οποίο απλά ορίζει την αύξηση η μείωση της τιμής του μετρητή ανάλογα με το σήμα up-down counter (updn\_cnt).

1. `timescale 10us/10ps
2. module Counter
3. (input [15:0]data\_in,
4. input rst\_,ld\_cnt,updn\_cnt,count\_enb,clk,
5. output logic [15:0]data\_out);
7. logic [15:0] data;
9. always\_ff @(posedge clk,negedge rst\_) begin
11. priority if (!rst\_) data = 0;
13. else if (!ld\_cnt) data = data\_in;
15. else if(count\_enb)begin
16. if(updn\_cnt) data = data + 1;
17. else data = data - 1;
18. end else data = data;
19. end
21. assign data\_out = data;
23. endmodule

# 2.2 Assertions Μετρητή

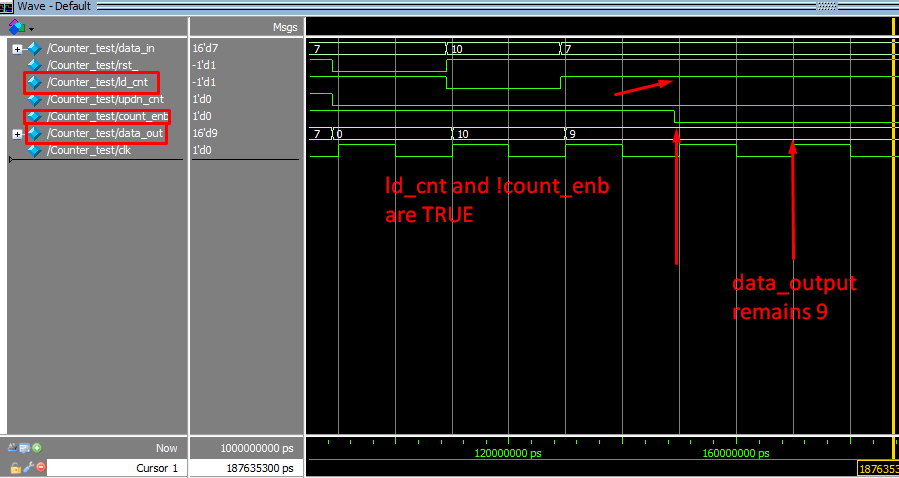
1. *Έλεγχος Σωστής Λειτουργίας reset (check\_reset)*

Αυτό το assertion θα πρέπει να αποτιμάται πάντα όταν εμφανίζεται η αρνητική ακμή του σήματος reset. Έπειτα, στην επόμενη θετική ακμή του ρολογιού θα πρέπει ήδη ο μετρητής να έχει μηδενιστεί.



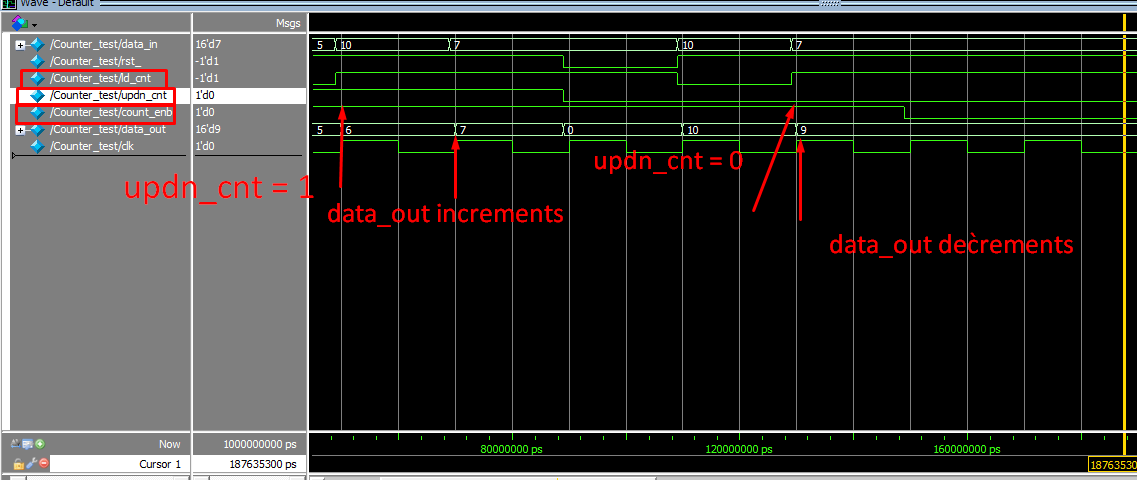
1. *Έλεγχος Σωστής Λειτουργίας της εξόδου του Μετρητή Όταν Κανένα σήμα δεν ενεργοποιεί κάποια if-else συνθήκη (check\_output)*

Σε κάθε θετική ακμή του ρολογιού, εφόσον το reset δεν είναι ενεργοποιημένο τότε το assertion αποτιμάται σε περίπτωση που τα σήματα load counter (ld\_counter) και counter enable (count\_enb) δεν ενεργοποιούν κάποια if-else συνθήκη. Τότε η έξοδος του μετρητή πρέπει να παραμείνει σταθερή σε σχέση με τον προηγούμενο κύκλο. Ο έλεγχος με την έτοιμη συνάρτηση $stable γίνεται στον δεύτερο κύκλο ρολογιού από τότε που άρχισε να αποτιμάται το assertion.

**

1. *Έλεγχος Σωστής Λειτουργίας της Αύξησης/Μείωσης του Μετρητή*

Σε κάθε θετική ακμή του ρολογιού, εφόσον το reset δεν είναι ενεργοποιημένο τότε το assertion αποτιμάται σε περίπτωση που load\_counter (ld\_cnt) και counter enable (cnt\_enb) είναι ενεργοποιημένα. Τότε ανάλογα την τιμή του σήματος up-down counter (updn\_cnt) είτε ο μετρητής αυξάνεται (updn\_cnt = 1) είτε μειώνεται (updn\_cnt = 0).



1. *Κώδικας Assertions*

`timescale 1us/1ns

module Counter\_assertions

   (input [15:0]data\_in,

    input rst\_,ld\_cnt,updn\_cnt,count\_enb,clk,

    input [15:0]data\_out);

    /\*

        --check\_reset--

        This property when it is asserted checks if reset=0 (active low)

        makes output 0 in the next positive clock edge

    \*/

    property check\_reset;

        @(negedge rst\_) 1'b1 |-> @(posedge clk)(data\_out == 16'd0);

    endproperty

    /\*

        --check\_output--

        This property when it is asserted checks if ld\_cnt=1 (active low) and count\_enb=1 (active high)

        make output stay stable for continous cycles

    \*/

    property check\_output;

        @(posedge clk) disable iff(!rst\_) (ld\_cnt && !count\_enb)|=> $stable(data\_out);

    endproperty

    /\*

        --check\_increment--

        --check\_decrements--

        Tis property when it is asserted checks if ld\_cnt=1 (active low) and count\_enb=1(active high)

        make output increases or decreases according to updn\_cnt(high-> increase, low->decrease).

    \*/

    property check\_increment;

        @(posedge clk) disable iff(!rst\_) (ld\_cnt && count\_enb && updn\_cnt)|=> ($past(data\_out)+1 == data\_out);

    endproperty

    property check\_dincrement;

        @(posedge clk)  disable iff(!rst\_) (ld\_cnt && count\_enb && !updn\_cnt)|=> ($past(data\_out,1) == data\_out+1);

    endproperty

    // ASSERTING PROPERTIES

    control\_reset: assert property (check\_reset) $display("1-PASS: RESET WORKS %t\n",$time);

                    else $display("1-FAIL: RESET DOES NOT WORK\n %t\n",$time);

    control\_ouput: assert property (check\_output) $display("2-PASS: STABLE OUTPUT WORKS %t\n",$time);

                    else $display("2-FAIL: STABLE OUTPUT DOES NOT WORK %t\n",$time);

    control\_increment: assert property (check\_increment) $display("3-PASS: INCREMENT WORKS %t\n",$time);

                    else $display("3-FAIL:INCREMENT DOES NOT WORK %t\n",$time);

    control\_dincrement: assert property (check\_dincrement) $display("4-PASS: DINCREMENT WORKS %t\n",$time);

                    else $display("4-FAIL:DINCREMENT DOES NOT WORK %t\n",$time);

endmodule

# 2.3 Kώδικας TestBench

`timescale 1us/1ns

module Counter\_test;

    logic [15:0]data\_in;

    logic rst\_,ld\_cnt,updn\_cnt,count\_enb,clk;

    logic [15:0]data\_out;

    Counter counter (data\_in,rst\_,ld\_cnt,updn\_cnt,count\_enb,clk,data\_out);

    bind Counter Counter\_assertions binded(data\_in,rst\_,ld\_cnt,updn\_cnt,count\_enb,clk,data\_out);

    always #10 clk = ~ clk;

    initial begin

        clk = 0;

        data\_in = 10;

        rst\_ = 0;

        ld\_cnt = 1;

        updn\_cnt = 1;

        count\_enb = 1;

        #29;

        data\_in = 5;

        rst\_ = 1;

        ld\_cnt = 0;

        updn\_cnt = 1;

        count\_enb = 1;

    @(posedge clk);

        #19

        data\_in = 10;

        rst\_ = 1;

        ld\_cnt = 1;

        updn\_cnt = 1;

        count\_enb = 1;

    @(posedge clk);

        #19

        data\_in = 7;

        rst\_ = 1;

        ld\_cnt = 1;

        updn\_cnt = 1;

        count\_enb = 1;

    @(posedge clk);

        #19

        data\_in = 7;

        rst\_ = 0;

        ld\_cnt = 1;

        updn\_cnt = 0;

        count\_enb = 1;

    @(posedge clk);

        #19

        data\_in = 10;

        rst\_ = 1;

        ld\_cnt = 0;

        updn\_cnt = 0;

        count\_enb = 1;

    @(posedge clk);

        #19

        data\_in = 7;

        rst\_ = 1;

        ld\_cnt = 1;

        updn\_cnt = 0;

        count\_enb = 1;

    @(posedge clk);

        #19

        data\_in = 7;

        rst\_ = 1;

        ld\_cnt = 1;

        updn\_cnt = 0;

        count\_enb = 0;

    @(posedge clk);

        #19

        data\_in = 7;

        rst\_ = 1;

        ld\_cnt = 1;

        updn\_cnt = 0;

        count\_enb = 0;

    end

endmodule

# 3 FIFO

# 3.1 Κύκλωμα FIFO

Το παρακάτω FIFO αποτελείται από ένα always\_ff block το οποίο είναι ευαίσθητο στην θετική ακμή του ρολογιού καθώς και στην αρνητική ακμή του σήματος reset (rst\_). Η λογική που ακολουθείται και σε αυτό το block είναι αυτής του ελέγχου των σημάτων βάση προτεραιότητας. Συγκεκριμένα, υψηλότερη προτεραιότητα έχει το σήμα reset το οποίο αδειάζει την FIFO. Έπειτα ακολουθεί το σήμα εγγραφής της FIFO (fifo\_write) και τέλος το σήμα ανάγνωσης της FIFO (fifo\_read). Στα if-else blocks που ενεργοποιούνται με τα δύο τελευταία σήματα, υπάρχουν οι κατάλληλοι έλεγχοι που εγγυόνται την ορθή λειτουργία της FIFO, είτε αυτό αφορά το όριο των εγγραφών που μπορεί να υποστηρίξει η συγκεκριμένη FIFO, είτε αφορά τα σήματα εξόδου (fifo\_full, fifo\_empty)που πληροφορούν για την εκάστοτε κατάσταση της.

`timescale 1us/1ps

module FIFO

    (input [15:0]fifo\_data\_in,

     input rst\_,fifo\_write,fifo\_read,clk,

     output logic [15:0]fifo\_data\_out,

     output logic fifo\_full,fifo\_empty);

    /\*

        WIDTH -> How bits of each memory slot

        DEPTH -> How many memory slots memory has in total

        MSB\_ptr-> Which is the most significant bit of the pointer

    \*/

    parameter WIDTH = 16,

              DEPTH = 16,

              MSB\_ptr = 4;

    /\*

        Memory: 16bit x 16 slots memory

        wr\_ptr: Writing pointer in memory

        rd\_ptr: Reading pointer in memory

        data: Temprary variable for data that must be an output

        counter: a viarable that is used to determine if FIFO is full or not

    \*/

    logic [WIDTH-1:0]Memory[DEPTH-1:0];

    logic [MSB\_ptr - 1:0]wr\_ptr,rd\_ptr;

    logic [WIDTH - 1:0]data;

    int counter;

    always\_ff @(posedge clk,negedge rst\_)begin: loop

        priority if (!rst\_)begin                 // Reseting circuit

            wr\_ptr = 0;

            rd\_ptr = 0;

            fifo\_full = 0;

            fifo\_empty = 1;

            data = 0;

            counter = 0;

        end else if (fifo\_write)begin            // If fifo\_write is TRUE

            if(counter < DEPTH)begin             // If FIFO is not full

                Memory[wr\_ptr] = fifo\_data\_in;   // Writing data\_input in the proper address

                counter++;

                wr\_ptr++;

                fifo\_empty = 0;                  // If at least one element is added, FIFO is not empty

                if(counter == 16)fifo\_full = 1;  // If counter == 16 then fifo is FULL

            end else begin

                fifo\_full = 1;                   // else FIFO is full

                fifo\_empty = 0;

            end

        end else if (fifo\_read)begin             // If fifo\_read is TRUE

            if(counter > 0)begin                 // If FIFO is not empty

                data = Memory[rd\_ptr];           // Reading data\_ouput from the proper address

                rd\_ptr++;

                counter--;

                fifo\_full = 0;                   // if at least one element is taken, FIFO is not full

                if (counter == 0)fifo\_empty = 1; // If counter == 0 then FIFO is empty

            end else begin

                fifo\_empty = 1;                  // else FIFO is empty

                fifo\_full = 0;

            end

        end

    end: loop

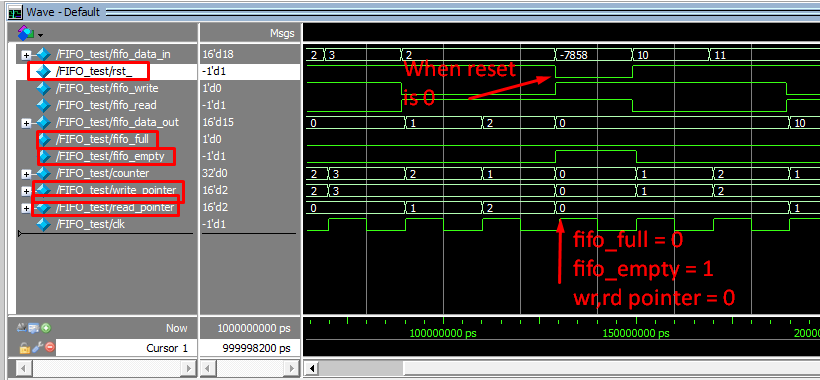
    assign fifo\_data\_out = data;

endmodule

# 3.2 Assertions FIFO

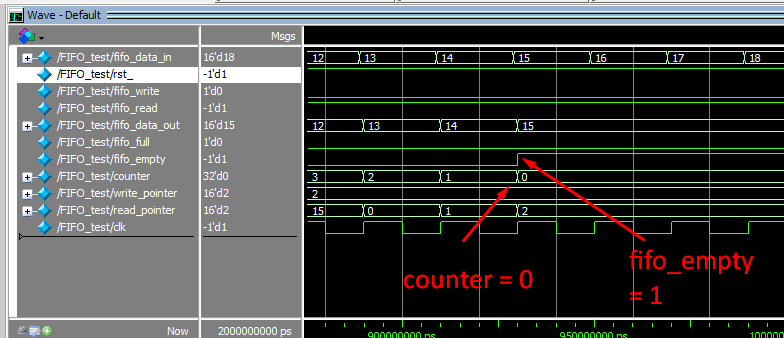
1. *Έλεγχος Σωστής Λειτουργίας Σήματος reset (check\_reset)*

Αυτό το assertion θα πρέπει να αποτιμάται πάντα όταν το σήμα reset ενεργοποιείται. Ως αποτέλεσμα, το σήμα εξόδου fifo\_full πρέπει να ισούται με μηδέν ενώ το σήμα fifo\_empty πρέπει να ισούται με 1 καθώς η FIFO πλέον είναι άδεια.

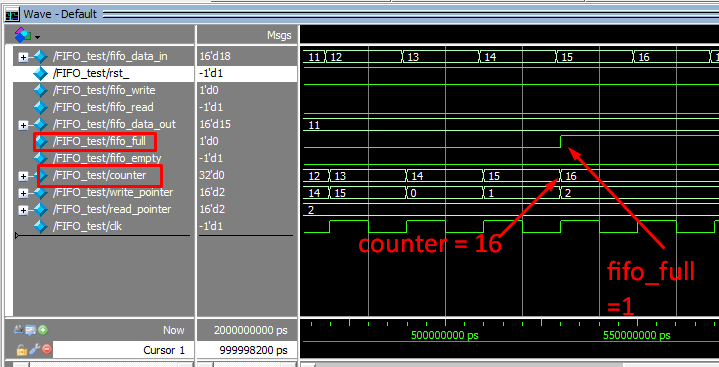
**

1. *Έλεγχος Σωστής Λειτουργίας για το Άδειασμα της FIFO (check\_fifo\_empty)*

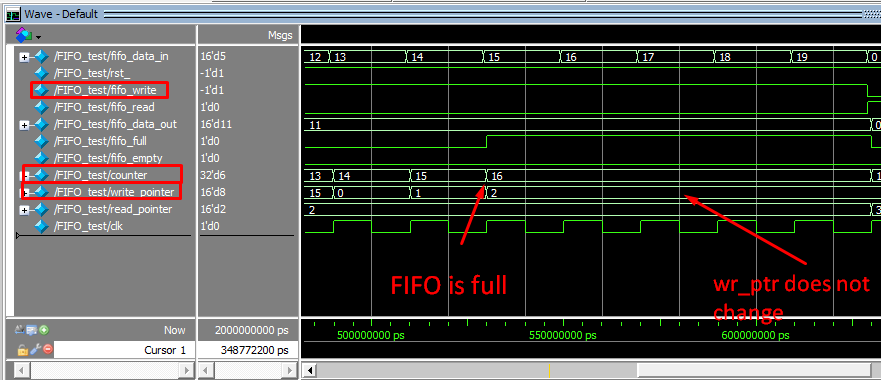
Σε κάθε θετική ακμή του ρολογιού, εφόσον το reset δεν είναι ενεργοποιημένο τότε το assertion αποτιμάται σε περίπτωση που ο counter (ο καταχωρητής που αποθηκεύει πόσες εγγραφές έχουν γίνει στην FIFO και δεν έχουν διαβαστεί ακόμη ) γίνει 0 τότε θα πρέπει το σήμα fifo\_empty να γίνει 1.



1. *Έλεγχος Σωστής Λειτουργίας για το γέμισμα της FIFO (check\_fifo\_full)*

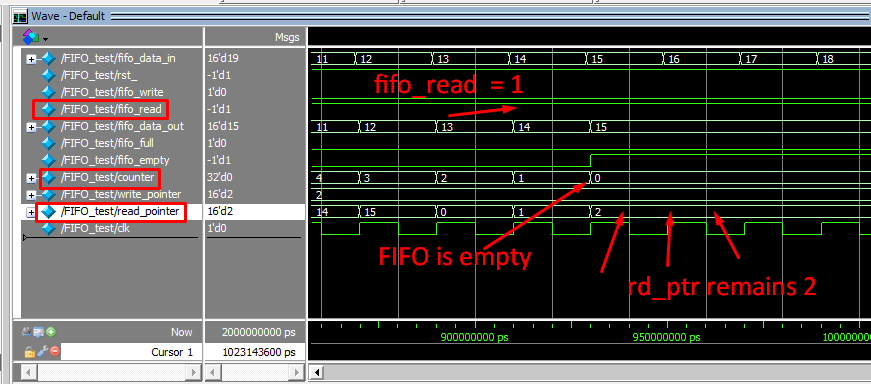
Σε κάθε θετική ακμή του ρολογιού, εφόσον το reset δεν είναι ενεργοποιημένο τότε το assertion αποτιμάται σε περίπτωση που ο counter (ο καταχωρητής που αποθηκεύει πόσες εγγραφές έχουν γίνει στη FIFO και δεν έχουν διαβαστεί ακόμη) γίνει 16 τότε θα πρέπει το σήμα fifo\_full να γίνει 1. 

1. *Έλεγχος Σωστής Λειτουργίας της Αίτησης Εγγραφής (check\_write\_pointer)*

Σε κάθε θετική ακμή του ρολογιού, εφόσον το reset δεν είναι ενεργοποιημένο τότε το assertion αποτιμάται όταν η FIFO έχει γεμίσει (counter = 16) και ζητείται εκ νέου εγγραφή (!fifo\_write = 1) και δεν ζητείται ανάγνωση (fifo\_read = 1). Τότε, για όσο συμβαίνει αυτός, θα πρέπει το write\_pointer (wr\_ptr) να παραμείνει σταθερός με την τιμή του προηγούμενου κύκλου καθώς δεν υπάρχουν ελεύθερες θέσεις για εγγραφή νέας πληροφορίας. 

1. *Έλεγχος Σωστής Λειτουργίας της Αίτησης Ανάγνωσης (check\_read\_pointer)*

Σε κάθε θετική ακμή του ρολογιού, εφόσον το reset δεν είναι ενεργοποιημένο τότε το assertion αποτιμάται όταν η FIFO έχει αδειάσει (counter = 0) και ζητείται εκ νέου ανάγνωση (!fifo\_read = 1) και δεν ζητείται εγγραφή (fifo\_read = 1). Τότε, για όσο συμβαίνει αυτός, θα πρέπει ο read\_pointer (rd\_ptr) να παραμείνει σταθερός με την τιμή του προηγούμενου κύκλου καθώς δεν εγγεγραμμένη πληροφορία στην FIFO.



1. *Κώδικας Assertios*
2. `timescale 1us/1ps
3. module FIFO\_assertions
4. (input int counter,
5. input [3:0]wr\_ptr,[3:0]rd\_ptr,
6. input [15:0]fifo\_data\_in,
7. input rst\_,fifo\_write,fifo\_read,clk,
8. input [15:0]fifo\_data\_out,
9. input fifo\_full,fifo\_empty);
11. /\*
12. --check\_reset--
13. This property checks if reset is working properly.
14. Reset should make writing,reading pointer 0 and inform that FIFO is not full (fifo\_full = 0)
15. and FIFO is empty (fifo\_empty = 1)
16. \*/
17. property check\_reset;
18. @(negedge rst\_) 1'b1 |-> @(posedge clk) (wr\_ptr == 0 && rd\_ptr == 0 && fifo\_full == 0 && fifo\_empty == 1);
19. endproperty

22. /\*
23. --check\_fifo\_empty--
24. This property checks if the fifo empty is activated (fifo\_empty = 1) when counter = 0
26. \*/
27. property check\_fifo\_empty;
28. @(posedge clk) disable iff(!rst\_) (counter == 0) |-> (fifo\_empty == 1);
29. endproperty

32. /\*
33. --check\_fifo\_full--
34. This property checks if the fifo\_full is activated (fifo\_full = 1) when counter = 16
35. \*/
36. property check\_fifo\_full;
37. @(posedge clk) disable iff(!rst\_) (counter == 16) |-> (fifo\_full == 1);
38. endproperty

41. /\*
42. --check\_write\_pointer--
43. This property checks if writing pointer stays stable when the counter = 16 => FIFO is full
44. \*/
45. property check\_write\_pointer;
46. @(posedge clk) disable iff(!rst\_) (counter == 16 && fifo\_write && !fifo\_read) |=> $stable(wr\_ptr);
47. endproperty

50. /\*
51. --check\_read\_pointer--
52. This property checks if reading pointer stays stabel when the counter = 0 => FIFO is empty
53. \*/
54. property check\_read\_pointer;
55. @(posedge clk) disable iff(!rst\_) (counter == 0 && !fifo\_write && fifo\_read) |=> $stable(rd\_ptr);
56. endproperty

59. // ASSERTING PROPERTIES
60. control\_reset: assert property (check\_reset) $display("1-PASS: RESET WORKS %t\n",$time);
61. else $display("1-FAIL: RESET DOES NOT WORK %t\n",$time);
62. control\_fifo\_empty: assert property (check\_fifo\_empty) $display("2-PASS: FIFO EMPTY WORKS %t\n",$time);
63. else $display("2-FAIL: FIFO EMPTY DOES NOT WORK %t\n",$time);
65. control\_check\_fifo\_full: assert property (check\_fifo\_full) $display("3-PASS: FIFO FULL WORKS %t\n",$time);
66. else $display("3-FAIL: FIFO FULL DOES NOT WORK %t \n",$time);
68. control\_check\_write\_pointer: assert property(check\_write\_pointer) $display("4-PASS: WRITE POINTER WORKS %t\n",$time);
69. else $display("4-FAIL: WRITE POINTER DOES NOT WORK %t\n",$time);
71. control\_chekc\_read\_pointer: assert property(check\_read\_pointer) $display("5-PASS: READ POINTER WORKS %t\n",$time);
72. else $display("5-FAIL: READ POINTER DOES NOT WORK %t\n",$time);
73. endmodule

# 3.3 Κώδικας Testbench

`timescale 1us/1ps

module FIFO\_test;

    logic [15:0]fifo\_data\_in;

    logic rst\_,fifo\_write,fifo\_read,clk;

    logic [15:0]fifo\_data\_out;

    logic fifo\_full,fifo\_empty;

    int counter;

    logic [15:0]write\_pointer;

    logic [15:0]read\_pointer;

    assign counter = fifo.counter;

    assign write\_pointer = fifo.wr\_ptr;

    assign read\_pointer = fifo.rd\_ptr;

    FIFO fifo(fifo\_data\_in,rst\_,fifo\_write,fifo\_read,clk,fifo\_data\_out,fifo\_full,fifo\_empty);

    bind fifo FIFO\_assertions binded(fifo.counter,fifo.wr\_ptr,fifo.rd\_ptr,fifo\_data\_in,rst\_,fifo\_write,fifo\_read,clk,fifo\_data\_out,fifo\_full,fifo\_empty);

    always #10 clk = ~clk;

    initial begin

        clk = 0;

        rst\_ = 0;

        fifo\_write = 1;

        fifo\_read = 1;

        fifo\_data\_in = 123214;

        #29               // WRITING

        rst\_ = 1;

        fifo\_write = 1;

        fifo\_read = 0;

        fifo\_data\_in = 1;

        @(posedge clk);

        #19               // WRITING

        fifo\_write = 1;

        fifo\_read = 0;

        fifo\_data\_in = 2;

        @(posedge clk);

        #19               // WRITING

        fifo\_write = 1;

        fifo\_read = 0;

        fifo\_data\_in = 3;

        @(posedge clk);

        #19                // READING

        fifo\_write = 0;

        fifo\_read = 1;

        fifo\_data\_in = 2;

        @(posedge clk);

        #19                // READING

        fifo\_write = 0;

        fifo\_read = 1;

        fifo\_data\_in = 2;

        @(posedge clk);

        #19                // RESETING

        rst\_ = 0;

        fifo\_write = 1;

        fifo\_read = 1;

        fifo\_data\_in = 123214;

        @(posedge clk);

        #19                // WRITING

        rst\_ = 1;

        fifo\_write = 1;

        fifo\_read = 0;

        fifo\_data\_in = 10;

        @(posedge clk);

        #19                // WRITING

        fifo\_write = 1;

        fifo\_read = 0;

        fifo\_data\_in = 11;

        @(posedge clk);

        #19                // READING

        fifo\_write = 0;

        fifo\_read = 1;

        fifo\_data\_in = 11;

        @(posedge clk);

        #19

        fifo\_write = 0;

        fifo\_read = 1;

        @(posedge clk);

        // This loop is used to full FIFO

        for(int i = 0; i < 20; i++)begin

            #19

            fifo\_write = 1;

            fifo\_read = 0;

            fifo\_data\_in = i;

            @(posedge clk);

        end

        // This loop is used to empty FIFO

        for(int i = 0; i < 20; i++)begin

            #19

            fifo\_write = 0;

            fifo\_read = 1;

            fifo\_data\_in = i;

            @(posedge clk);

        end

    end

endmodule