Low-Level HW Digital Systems II

Κωτούλας Εμμανουήλ 9697

Module 1ο : Up and Down Counter

C:/Users/emkot/Desktop/univ/8thSem/Hw2/ergasia/counter.sv (/test_bindP_counter/c1) - Default

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

Παραπάνω βρίσκεται ο κώδικας για το up and down counter module.

Λειτουργία

Αρχικά ελέγχεται η κατάσταση του reset(rst_) και αν είναι ενεργό μηδενίζεται η είσοδος. Έπειτα αν δεν είναι ενεργό ελέγχεται η κατάσταση του load (ld_cnt) και αν είναι ενεργό περνάει η τιμή της εισόδου στην έξοδο. Τέλος αν δεν είναι ενεργό το load, ελέγχουμε την κατάσταση του count enable και αν είναι ενεργό κάνουμε count up ή count down ανάλογα με την τιμή του updn_cnt.

Testbench

```
1   `timescale 1ns/1ns
2
3   module tb_counter;
4
5           logic [15:0] tdata_in, tdata_out;
6           logic trst_, tld_cnt, tupdn_cnt, tcount_enb, tclk;
7
8           counter c1 (
9           .data_in(tdata_in),
10          .rst_(trst_),
11          .ld_cnt(tld_cnt),
12          .updn_cnt(tupdn_cnt),
13          .count_enb(tcount_enb),
14          .clk(tclk),
15          .data_out(tdata_out)
16          );
17
18          bind c1 counter_property cp1 (
19          .pdata_in(data_in),
20          .prst_(rst_),
21          .pld_cnt(ld_cnt),
22          .pupdn_cnt(updn_cnt),
23          .pcount_enb(count_enb),
24          .pclk(clk),
25          .pdata_out(data_out)
26          );
```

```
27
28      localparam CLK_PERIOD = 4;
29
30      always #(CLK_PERIOD/2) tclk=~tclk;
31
32      initial begin
33          // first all xs
34          #2  trst_=1'bx; tclk=1'bx; tdata_in=16'bx; tld_cnt=1'bx; tupdn_cnt=1'bx; tcount_enb=1'bx;
35          // activate reset
36          #10 trst_=1'b0; tclk=1'b1; tdata_in=16'b0; tld_cnt=1'b1; tupdn_cnt=1'b1; tcount_enb=1'b0;
37          // hold reset for 16, not enable counting
38          #16 trst_=1'b1; tdata_in=16'b0; tld_cnt=1'b1; tupdn_cnt=1'b1; tcount_enb=1'b0;
39          // enable counting upcounting
40          #8  trst_=1'b1; tdata_in=16'b0; tld_cnt=1'b1; tupdn_cnt=1'b1; tcount_enb=1'b1;
41          // do nothing for one
42          #20 trst_=1'b1; tdata_in=16'b0; tld_cnt=1'b1; tupdn_cnt=1'b0; tcount_enb=1'b0;
43          // downcount
44          #8  trst_=1'b1; tdata_in=16'b0; tld_cnt=1'b1; tupdn_cnt=1'b0; tcount_enb=1'b1;
45          // do nothing
46          #16 trst_=1'b1; tdata_in=16'b0; tld_cnt=1'b1; tupdn_cnt=1'b0; tcount_enb=1'b0;
47          // load value
48          #8 trst_=1'b1; tdata_in=16'd6942; tld_cnt=1'b0; tupdn_cnt=1'b1; tcount_enb=1'b0;
49          // do nothing
50          #4  trst_=1'b1; tdata_in=16'b0; tld_cnt=1'b1; tupdn_cnt=1'b1; tcount_enb=1'b0;
51          // count up till it resets
52          #4  trst_=1'b1; tdata_in=16'b0; tld_cnt=1'b1; tupdn_cnt=1'b1; tcount_enb=1'b1;
53          // do nothing
54          #32 trst_=1'b1; tdata_in=16'b0; tld_cnt=1'b1; tupdn_cnt=1'b1; tcount_enb=1'b0;
55
56          $stop(2);
57      end
58
59
60  endmodule
61
```

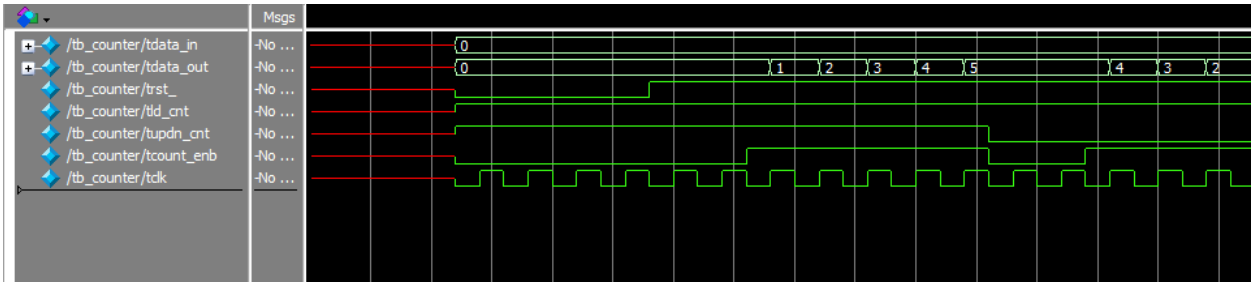
Το Testbench κάνει instantiate το module counter και το κάνει bind με τα assertions και έπειτα με ένα initial block δίνει διάφορες τιμές στις μεταβλητές εισόδου ώστε να ελεγχθεί η λειτουργικότητα του κυκλώματος. Το ρολόι υλοποιείται μέσω μίας always εναλλάσσοντας την τιμή του.

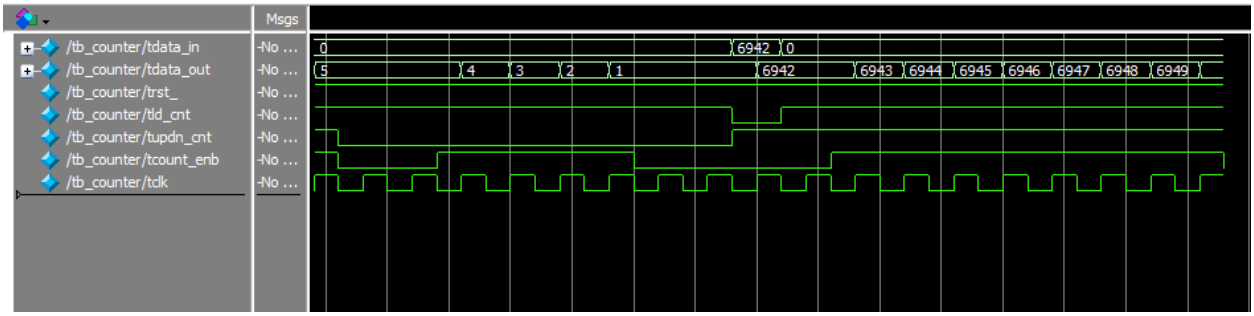
SVAs

```
Ln#
 1  `timescale 1ns/1ns
 2  module counter_property(pdata_in, prst_, pld_cnt, pupdn_cnt, pcount_enb, pclk, pdata_out);
 3
 4  input [15:0] pdata_in,pdata_out;
 5  input prst_, pld_cnt, pupdn_cnt, pcount_enb, pclk;
 6
 7  property pr1;
 8      @(posedge pclk) !prst_ |-> pdata_out == 0;
 9  endproperty
10
11  property pr2;
12      @(posedge pclk) disable iff(!prst_) (pld_cnt && !pcount_enb) |=> (pdata_out == $past(pdata_out) );
13  endproperty
14
15  property pr3;
16      @(posedge pclk) disable iff(!prst_)
17           (pld_cnt && pcount_enb) |=>
18          if($past(pupdn_cnt))
19          ( ($past(pdata_out) + 1)  == pdata_out)
20          else if( !($past(pupdn_cnt)) )
21          ( ($past(pdata_out) - 1)  == pdata_out);
22  endproperty
23
24  ap1 : assert property (pr1) else $display("pr1 FAIL time : %0t ", $time);
25  cp1 : cover property (pr1) $display("pr1 PASS time : %0t ", $time);
26
27  ap2 : assert property (pr2) else $display("pr2 FAIL time : %0t", $time);
28  cp2 : cover property (pr2) $display("pr2 PASS time : %0t", $time);
29
30  ap3 : assert property (pr3) else $display("pr3 FAIL time : %0t", $time);
31  cp3 : cover property (pr3) $display("pr3 PASS time : %0t", $time);
32
33
34  endmodule
35
```

Τα properties ορίζονται με την σειρά που υπάρχουν στην εκφώνηση της εργασίας. Εξαιτίας του τρόπου που λειτουργούν τα assertions τα properties 2 και 3 λειτουργούν ελέγχοντας την τιμή εξόδου του επόμενου κύκλου σε σύγκριση με την τιμή εξόδου στον κύκλο που βρισκόμαστε. Χρησιμοποιείται assert και cover ώστε να μην έχουμε vacuous passes.

Results






```
VSIM 39> run -all
# pr1 PASS time : 14
# pr1 PASS time : 18
# pr1 PASS time : 22
# pr1 PASS time : 26
# pr2 PASS time : 34
# pr2 PASS time : 38
# pr3 PASS time : 42
# pr3 PASS time : 46
# pr3 PASS time : 50
# pr3 PASS time : 54
# pr3 PASS time : 58
# pr2 PASS time : 62
# pr2 PASS time : 66
# pr3 PASS time : 70
# pr3 PASS time : 74
# pr3 PASS time : 78
# pr3 PASS time : 82
# pr2 PASS time : 86
# pr2 PASS time : 90
# pr2 PASS time : 98
# pr3 PASS time : 102
# pr3 PASS time : 106
# pr3 PASS time : 110
# pr3 PASS time : 114
# pr3 PASS time : 118
# pr3 PASS time : 122
# pr3 PASS time : 126
```

Τα waveforms παρουσιάζουν την σωστή λειτουργία του module καθώς και τα assertions τα οποία κάνουν PASS τις κατάλληλες στιγμές. Αρχικά κάνει reset και έπειτα κάνει κανονικά count up, σταματάει για λίγους κύκλους, κάνει count down, κάνει load μια τιμή και κάνει count up.

Module 2º : FIFO

```
module fifo
        #(parameter WIDTH = 16,
        parameter DEPTH = 16)
        (output logic [WIDTH-1:0] fifo_data_out,
        output logic fifo_full, fifo_empty,
        input logic [WIDTH-1:0] fifo_data_in,
        input logic rst_, clk, fifo_write, fifo_read);

        logic [$clog2(DEPTH)-1:0] wr_ptr, rd_ptr;
        logic [DEPTH-1:0][WIDTH-1:0] fifo_memory;
        logic [$clog2(DEPTH):0] cnt;

        always_ff @(posedge clk, negedge rst_) begin

        if(!rst_) begin
            for(bit[4:0] i = 0 ;i <= DEPTH - 1 ; i++)begin fifo_memory[i] <= 0 ; end
            wr_ptr <= 0;
            rd_ptr <= 0;
            cnt <= 0;
        end

        else begin

            if(fifo_write && !fifo_full) begin
                fifo_memory[wr_ptr] <= fifo_data_in;
                wr_ptr <= wr_ptr + 1 ;
                cnt <= cnt + 1 ;
            end

            else if(fifo_read && !fifo_empty)begin
                fifo_data_out <= fifo_memory[rd_ptr];
                rd_ptr <= rd_ptr + 1;
                cnt <= cnt - 1;
            end

        end
    end
```

```

    always_comb begin
        if(cnt == 0) begin
            fifo_empty = 1;
            fifo_full = 0;
        end
        else if(cnt == DEPTH) begin
            fifo_empty = 0;
            fifo_full = 1;
        end
        else begin
            fifo_empty = 0;
            fifo_full = 0;
        end
    end

endmodule
```

Λειτουργία

Αρχικά ελέγχεται η τιμή του reset(rst_) και αν είναι 0 μηδενίζονται όλες οι τιμές. Εάν δεν είναι ενεργό το reset τότε ελέγχεται αν το read ή write είναι ενεργό (δεν αναφέρεται κάπου προτεραιότητα οπότε θεωρώ ότι είναι illegal να είναι και τα δύο ενεργά ταυτόχρονα και έτσι η σειρά στην if δεν έχει πραγματικά σημασία) και αντίστοιχα διαβάζω από έξω την τιμή στην είσοδο και την αποθηκεύω στην μνήμη ή βγάζω μια θέση μνήμης στην έξοδο, κάνοντας increment τον αντίστοιχο pointer και αυξάνοντας ή μειώνοντας την τιμή του cnt. Τέλος υπάρχει ένα συνδυαστικό always για την τιμή των fifo_empty και fifo_full. Το μέγεθος του cnt επιλέχθηκε ώστε να μπορεί να μετρήσει τουλάχιστον από 0 μέχρι και 16 στην συγκεκριμένη περίπτωση.

Testbench

```
1   `timescale 1ns/1ns
2
3   module tb_fifo;
4       logic [15:0] tfifo_data_in, tfifo_data_out;
5
6       logic trst_, tclk, tfifo_write, tfifo_read, tfifo_full, tfifo_empty;
7
8       fifo f1 (
9           .fifo_data_in(tfifo_data_in),
10          .rst_(trst_),
11          .clk(tclk),
12          .fifo_write(tfifo_write),
13          .fifo_read(tfifo_read),
14          .fifo_data_out(tfifo_data_out),
15          .fifo_full(tfifo_full),
16          .fifo_empty(tfifo_empty)
17          );
18
19      bind f1 fifo_property fp1 (
20          .pfifo_data_in(fifo_data_in),
21          .prst_(rst_),
22          .pclk(clk),
23          .pfifo_write(fifo_write),
24          .pfifo_read(fifo_read),
25          .pfifo_data_out(fifo_data_out),
26          .pfifo_full(fifo_full),
27          .pfifo_empty(fifo_empty),
28          .prd_ptr(f1.rd_ptr),
29          .pwr_ptr(f1.wr_ptr),
30          .pcnt(f1.cnt)
31      );
32
33      localparam CLK_PERIOD = 4;
34
35      always #(CLK_PERIOD/2) tclk=~tclk;
36
37      initial begin
38          #1  trst_ = 1'bx; tclk = 1'bx; tfifo_write = 1'bx; tfifo_read = 1'bx; tfifo_data_in = 16'bx;
39
40          #10 trst_ = 1'b0; tclk = 1'b1; tfifo_write = 1'b0; tfifo_read = 1'b0; tfifo_data_in = 16'b0;
41
42          #16 trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd1;
43
44          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd2;
45
46          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd3;
47
48          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd4;
49
50          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd5;
```

```
50          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd5;
51
52          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd6;
53
54          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd7;
55
56          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd8;
57
58          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd9;
59
60          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd10;
61
62          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd11;
63
64          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd12;
65
66          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd13;
67
68          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd14;
69
70          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd15;
71
72          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd16;
73
74          #4  trst_ = 1'b1; tfifo_write = 1'b1; tfifo_read = 1'b0; tfifo_data_in = 16'd234;
75
76          #4  trst_ = 1'b1; tfifo_write = 1'b0; tfifo_read = 1'b1; tfifo_data_in = 16'd234;
77
78          #(16*5) trst_ = 1'b1; tfifo_write = 1'b0; tfifo_read = 1'b1; tfifo_data_in = 16'd234;
79
80          $stop(2);
81      end
82
83  endmodule
84
```

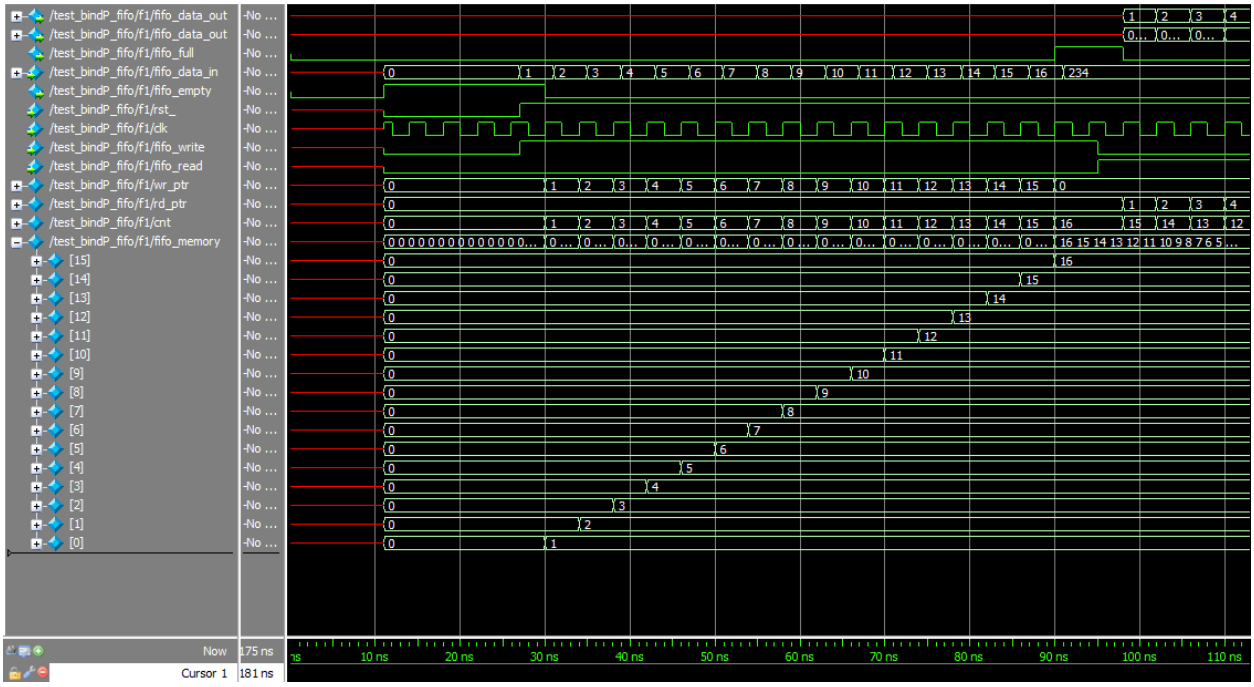
Αρχικά το Testbench κάνει instantiate το fifo module και κάνει το bind με τα assertions. Έπειτα ενεργοποιεί το reset και φορτώνεται κάθε κύκλο μια διαφορετική τιμή. Φορτώνονται περισσότερες τιμές απ' ότι χωράει η μνήμη για να δειχθεί ότι δεν εισάγονται περισσότερες τιμές από 16 και τις αγνοεί. Μετά διαβάζονται τιμές μέχρι να αδειάσει η μνήμη.

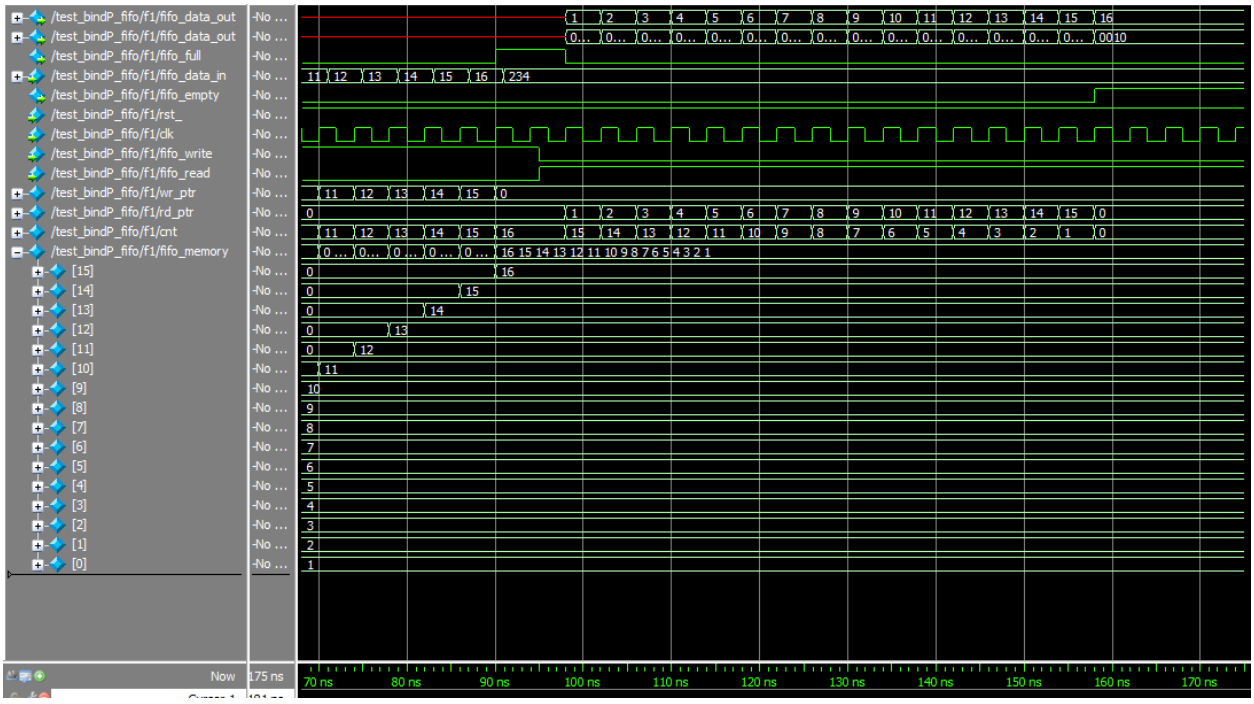
SVAs

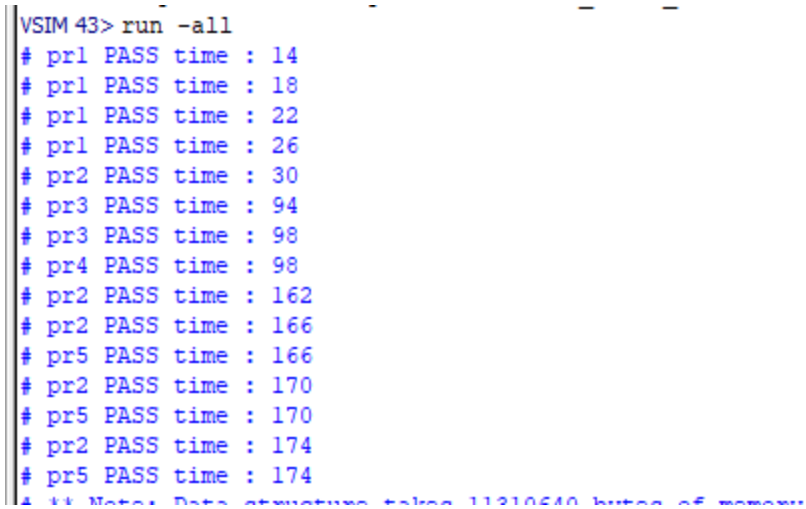
```
`timescale 1ns/1ns

module fifo_property(pfifo_data_in, prst_, pfifo_write, pfifo_read, pclk, pfifo_data_out, pfifo_full, pfifo_empty, prd_ptr, pwr_ptr, pcnt );

input [15:0] pfifo_data_in, pfifo_data_out;
input [3:0] prd_ptr, pwr_ptr;
input [4:0] pcnt;
input prst_, pfifo_write, pfifo_read, pclk, pfifo_full, pfifo_empty;

property pr1;
    @(posedge pclk) !prst_ |-> (pfifo_empty && !pfifo_full && !prd_ptr && !pwr_ptr && !pcnt) ;
endproperty

property pr2;
    @(posedge pclk) disable iff(!prst_) !pcnt |-> pfifo_empty ;
endproperty

property pr3;
    @(posedge pclk) disable iff(!prst_) pcnt >= 16 |-> pfifo_full ;
endproperty

property pr4;
    @(posedge pclk) (pfifo_full && pfifo_write && !pfifo_read) |=> (pwr_ptr == $past(pwr_ptr));
endproperty

property pr5;
    @(posedge pclk) (pfifo_empty && !pfifo_write && pfifo_read) |=> (prd_ptr == $past(prd_ptr));
endproperty


ap1 : assert property (pr1) else $display("pr1 FAIL time : %0t", $time);
cp1 : cover property (pr1) $display("pr1 PASS time : %0t", $time);

ap2 : assert property (pr2) else $display("pr2 FAIL time : %0t", $time);
cp2 : cover property (pr2) $display("pr2 PASS time : %0t", $time);

ap3 : assert property (pr3) else $display("pr3 FAIL time : %0t", $time);
cp3 : cover property (pr3) $display("pr3 PASS time : %0t", $time);

ap4 : assert property (pr4) else $display("pr4 FAIL time : %0t", $time);
cp4 : cover property (pr4) $display("pr4 PASS time : %0t", $time);

ap5 : assert property (pr5) else $display("pr5 FAIL time : %0t", $time);
cp5 : cover property (pr5) $display("pr5 PASS time : %0t", $time);


endmodule
```

Τα properties ορίζονται με την σειρά που υπάρχουν στην εκφώνηση της εργασίας. Χρησιμοποιείται assert και cover ώστε να μην έχουμε vacuous passes. Τα assertions 4 και 5 ελέγχουν τον επόμενο κύκλο αν η τιμή παρέμεινε σταθερή.

Results


/test_bindP_fifo/f1/fifo_data_out
/test_bindP_fifo/f1/fifo_data_out
/test_bindP_fifo/f1/fifo_full
/test_bindP_fifo/f1/fifo_data_in
/test_bindP_fifo/f1/fifo_empty
/test_bindP_fifo/f1/rst_
/test_bindP_fifo/f1/clk
/test_bindP_fifo/f1/fifo_write
/test_bindP_fifo/f1/fifo_read
/test_bindP_fifo/f1/wr_ptr
/test_bindP_fifo/f1/rd_ptr
/test_bindP_fifo/f1/cnt
/test_bindP_fifo/f1/fifo_memory
[15]
[14]
[13]
[12]
[11]
[10]
[9]
[8]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
Now
Cursor 1
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns


/test_bindP_fifo/f1/fifo_data_out
/test_bindP_fifo/f1/fifo_data_out
/test_bindP_fifo/f1/fifo_full
/test_bindP_fifo/f1/fifo_data_in
/test_bindP_fifo/f1/fifo_empty
/test_bindP_fifo/f1/rst_
/test_bindP_fifo/f1/clk
/test_bindP_fifo/f1/fifo_write
/test_bindP_fifo/f1/fifo_read
/test_bindP_fifo/f1/wr_ptr
/test_bindP_fifo/f1/rd_ptr
/test_bindP_fifo/f1/cnt
/test_bindP_fifo/f1/fifo_memory
[15]
[14]
[13]
[12]
[11]
[10]
[9]
[8]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
Now
70 ns
80 ns
90 ns
100 ns
110 ns
120 ns
130 ns
140 ns
150 ns
160 ns
170 ns



Όπως φαίνεται από τα waveforms αποθηκεύονται σωστά οι τιμές στη μνήμη και διαβάζονται σωστά με τους pointer να κάνουν tip over και να γίνονται ξανά μηδέν. Επίσης φαίνεται ότι δεν αποθηκεύτηκε η τιμή που βρίσκονταν στην είσοδο όταν η μνήμη ήταν γεμάτη. Τέλος τα assertions φαίνεται να λειτουργούν σωστά χωρίς να κάνει κάποιο fail.

Δεν χρησιμοποίησα macros και το compile και simulation έγινε με χρήση του UI του Questa.

- Το αρχείο counter.sv περιέχει την υλοποίηση του up down counter module.
- Το αρχείο tb_counter.sv περιέχει το testbench για τον counter module
- Το αρχείο counter_property.sv περιέχει το module με τα assertions

Με παρόμοιο τρόπο έχουν ονομαστεί τα αρχεία για την fifo.