Εργασία HW-2

Τζατζίδης Αντώνης

AEM:9938

Σύνοψη

Σκοπός της εργασίας ήταν η σχεδίαση και προσομοίωση 2 κυκλωμάτων, ενός FIFO και ενός counter στη SystemVerilog μαζί με κάποια assertions που μας δώθηκαν καθώς και από ενα testbench στο καθένα για τη βεβαίωση της σωστής λειτουργίας του κυκλώματος.

Counter Module

Για τη σχεδίαση χρησιμοποίησα μία εσωτερική μεταβλητή που "μαζεύει" τα 3 σήματα που καθορίζουν την λειτουργία του μετρητή (ld_cnt,count_enb,updn_cnt), η οποία ανανεώνεται σε ένα always_comb block.

```
logic [2:0] state_info;
...
always_comb begin : set_state
    state_info = {ld_cnt,count_enb,updn_cnt};
end
```

Το always_ff block ενεργοποιείται στη θετική ακμή του ρολογιού ή στην αρνητική του rst αφού λειτουργεί ασύγχρονα. Αν δεν είναι ενεργοποιημένο το rst τότε ο μετρητής ανανεώνει την τιμή του με βάση την μεταβλητή state_info. Χρησιμοποίησα priority case ώστε να τηρείται η προτεραιότητα μεταξύ των 3 σημάτων όπως αναφέρονται στις οδηγίες.

```
always_ff @ (posedge clk, negedge rst)
begin
    if(!rst) data_out <= 16'b0;
    else
    begin
        priority case(state_info)
            3'b 000: data_out <= data_in;
            3'b 001: data_out <= data_in;
            3'b 010: data_out <= data_in;
            3'b 011: data_out <= data_in;
            3'b 100: data_out <= data_out;
            3'b 101: data_out <= data_out;
            3'b 110: data_out <= data_out - 1;
            3'b 111: data_out <= data_out + 1;
        endcase
    end
end
```

Counter Assertions

Τα assertions έχουν ίδια δομή όπως στο παράδειγμα. Με τη χρήση του ifdef επιλέγουμε ποιό θέλουμε να ενεργοποιήσουμε ώστε να είναι πιο εύκολος ο έλεγχος τους. Το πρώτο assertion ελέγχει αν η έξοδος είναι 0 στη θετική ακμή του ρολογιού όταν ενεργοποιείται το reset.

```
`ifdef reset
property pr1;
    @(negedge prst) 1'b1  |-> @(posedge pclk) pdata_out == 16'd0;
endproperty
```

Το δεύτερο ελέγχει αν η έξοδος παραμένει σταθερή όταν ενεργοποιούμε το pcount_enb (και το rst δεν είναι ενεργό) χρησιμοποιώντας το $stable.

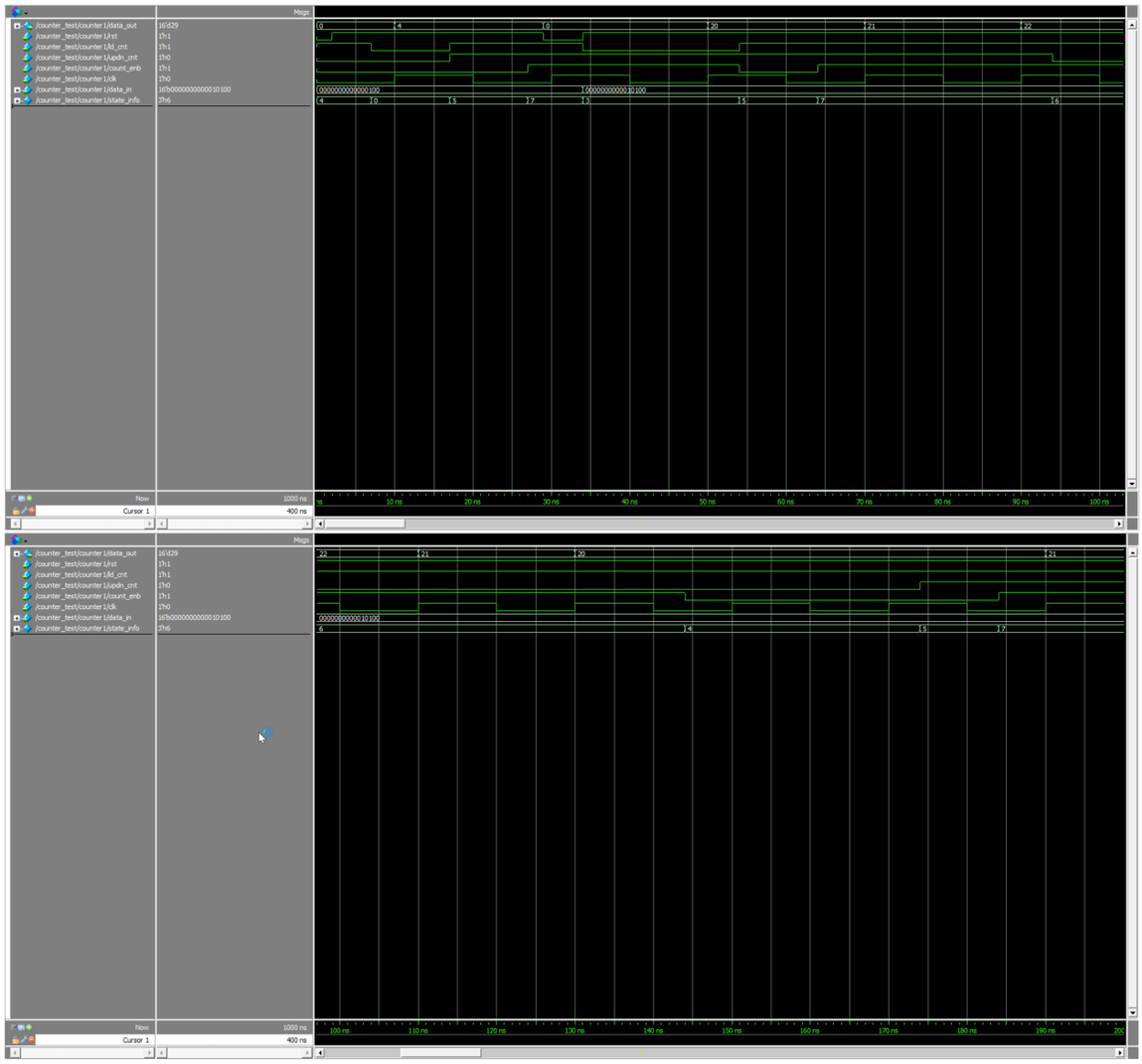
```
`elsif count_stable
property pr2;
    @(posedge pclk) disable iff(!prst) (pld_cnt && !pcount_enb) |->
    ↪  @(negedge pclk) $stable(pdata_out);
endproperty
```

Το τρίτο ελέγχει αν τα υπόλοιπα 2 σήματα που καθορίζουν την έξοδο έχουν τις κατάλληλες τιμές (και το rst δεν είναι ενεργό) ανάλογα με την τιμή του updn_cnt ο μετρητής αυξάνει ή μειώνει την έξοδο κατά 1. Επειδή το δεξί μέρος του assertion ελέγχεται στην αρνητική ακμή του ρολογιού θέλουμε την τιμή του updn_cnt στον προηγούμενο κύκλο καθώς τότε καθορίζεται αν θα αυξηθεί η θα μειωθεί το data_out γι' αυτό χρησιμοποίησα το $past(pupdn_cnt).

```
`elsif count_updown
property pr3;
    @(posedge pclk) disable iff(!prst) (pld_cnt && pcount_enb) |->
    ↪  @(negedge pclk) ($past(pupdn_cnt)) ? (pdata_out ==
    ↪  $past(pdata_out) + 16'd1) : (pdata_out == $past(pdata_out) -
    ↪  16'd1);
endproperty
```

Counter Testbench

Το testbench ακολουθεί τη δομή του παραδείγματος και δοκιμάζει διάφορες περιπτώσεις που κάνουν trigger τα assertions.



FIFO Module

Το FIFO δέχεται δύο παραμέτρους: DATA_WIDTH και FIFO_DEPTH οι οποίες έχουν default τιμές 16. Για τον υπολογισμό των bits που απαιτούνται για τα τους write και read pointers χρησιμοποίητει η μέθοδος $clog2 ώστε να έχουν log_2 FIFO_DEPTH bits. Επειδή ο counter μπορεί να γίνει ίσος με το FIFO_DEPTH προσθέτουμε ένα έξτρα bit για να μην έχουμε overflow. Επιπλέον όπως πριν το state_info "μαζεύει" τα σήματα fifo_write, fifo_read. Τέλος το regs είναι ένα array μήκους FIFO_DEPTH και κάθε στοιχείο αποτελείται από DATA_WIDTH bits, σ'αυτό θα αποθηκεύονται και θα διαβάζονται τα δεδομένα.

```
logic [$clog2(FIFO_DEPTH)-1:0] wr_ptr,rd_ptr;
logic [$clog2(FIFO_DEPTH):0] cnt;
logic [DATA_WIDTH-1:0] regs [0:FIFO_DEPTH-1];
logic [1:0] state_info;
```

Το state_info και το ασύγχρονο reset λειτουργούν όπως στο προηγούμενο κύκλωμα. Αν το reset δεν είναι ενεργό, στη θετική ακμή του ρολογιού διακρίνονται 3 περιπτώσεις ανάλογα με τα σήματα read και write (αν κανένα από τα 2 δεν είναι ενεργό, τότε το FIFO δεν κάνει τίποτα).

- Αν και τα 2 σήματα είναι ενεργά τότε ελέγχει αν το FIFO είναι γεμάτο για να διαβάσει πριν γράψει, διαφορετικά πρώτα διαβάζει και μετά γράφει, αυξάνει τους write και read pointers (το cnt δεν αλλάζει αφού έχουμε μια ανάγνωση και μία εγγραφή) και ελέγχει με βάση τη συνθήκη που δώθηκε αν είναι γεμάτο η άδειο.

- Αν μόνο το fifo_read είναι ενεργό ελέγχει πρώτα αν το FIFO είναι άδειο. Αν δεν είναι διαβάζει, αυξάνει το read_ptr, μειώνει το cnt και ελέγχει αν είναι άδειο (προφανώς δεν μπορεί να είναι γεμάτο). Επειδή οι εντολές θα εκτελεστούν ταυτόγχρονα ο έλεγχος για το αν είναι άδειο θα γίνει με το cnt−1.
- Το αντίστοιχο συμβαίνει αν μόνο το fifo_write είναι ενεργό.

```
case(state_info)
    2'b11: begin
            if(cnt >= FIFO_DEPTH)
            begin
                fifo_data_out <= regs[rd_ptr];
                regs[wr_ptr] <= fifo_data_in;
                rd_ptr <= rd_ptr + 1;
                wr_ptr <= wr_ptr + 1;

            end
            else
            begin
                regs[wr_ptr] <= fifo_data_in;
                fifo_data_out <= regs[rd_ptr];
                wr_ptr <= wr_ptr + 1;
                rd_ptr <= rd_ptr + 1;
            end
            fifo_empty <= (cnt == 0) ? 1:0;
            fifo_full  <=  (cnt >= FIFO_DEPTH) ? 1:0;


    end
    2'b01: begin
            if(cnt == 0) fifo_empty <= 1;
            else
            begin
                fifo_data_out <= regs[rd_ptr];
                rd_ptr <= rd_ptr + 1;
                cnt <= cnt - 1;
                fifo_empty <= (cnt-1 == 0) ? 1:0;
                fifo_full <= 0;
            end

    end
    2'b10: begin
            if(cnt >= FIFO_DEPTH) fifo_full  <= 1;
            else
            begin
                regs[wr_ptr] <= fifo_data_in;
                wr_ptr <= wr_ptr + 1;
                cnt <= cnt + 1;
```

```
                fifo_full  <=  (cnt+1 >= FIFO_DEPTH) ? 1:0;
                fifo_empty <= 0;
            end
    end
    default: continue;
endcase
```

FIFO Assertions

Τα assertions χρησιμοποιούν εκτός από τα inputs και outputs του module τα εσωτερικά σήματα read_pointer, write_pointer και cnt. Χωρίζονται σε 3 ομάδες με τα ifdef όπως στο προηγούμενο παράδειγμα.

- Στην πρώτη ελέγχεται η λειτουργία του reset όπως στον counter.

```
`ifdef reset
property pr1;
    @(negedge prst) 1'b1  |-> @(posedge pclk) (!pfifo_full &&
    ↪  pfifo_empty && pcnt == 0);
endproperty
```

- Στην δεύτερη ελέγχεται αν λειτουργούν σωστά οι συνθήκες για τα output fifo_full, fifo_empty

```
`elsif empty_full
property pr1;
    @(posedge pclk) disable iff(!prst) pcnt == 0 |-> pfifo_empty ==
    ↪  1;
endproperty

property pr2;
    @(posedge pclk) disable iff(!prst) pcnt >= FIFO_DEPTH |->
    ↪  pfifo_full == 1;
endproperty
```

- Στην τρίτη ελέγχεται αν το κύκλωμα δεν επιτρέπει τις εγγραφές σε γεμάτο FIFO και τις αναγώσεις σε άδειο βλέποντας αν ο write_ptr ή ο read_ptr αντίστοιχα δεν μεταβάλλεται.

```
`elsif read_write_invalid
property pr1;
    @(posedge pclk) disable iff(!prst) ((pcnt >= FIFO_DEPTH) &&
    ↪  pfifo_write && !pfifo_read) |-> @(negedge pclk)
    ↪  $stable(pwr_ptr);
endproperty
```

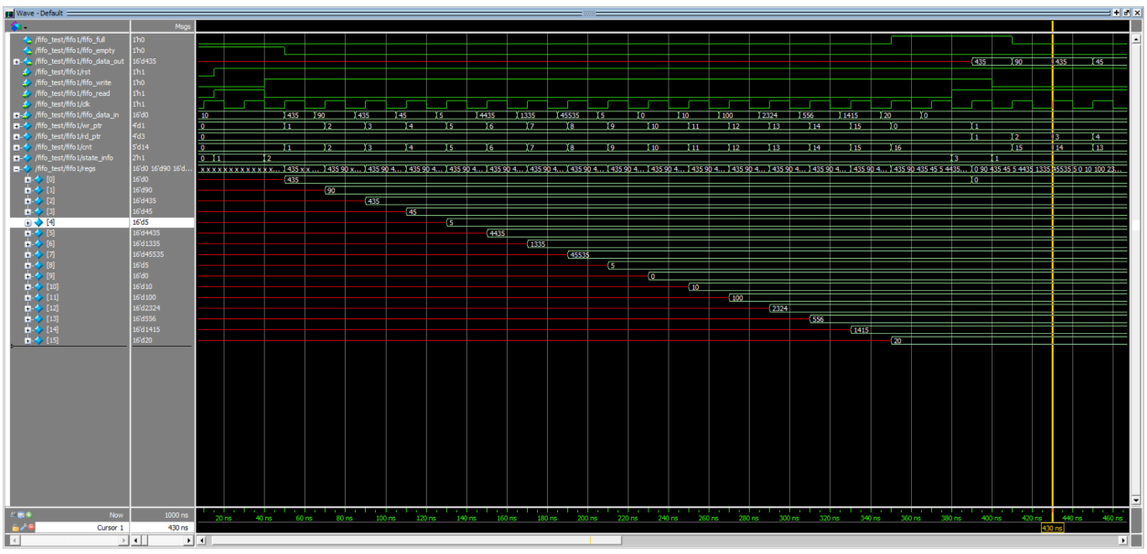
```
property pr2;
   @(posedge pclk) disable iff(!prst)  ((pcnt == 0) && !pfifo_write
    ↪  && pfifo_read) |-> @(negedge pclk) $stable(prd_ptr);
endproperty
```

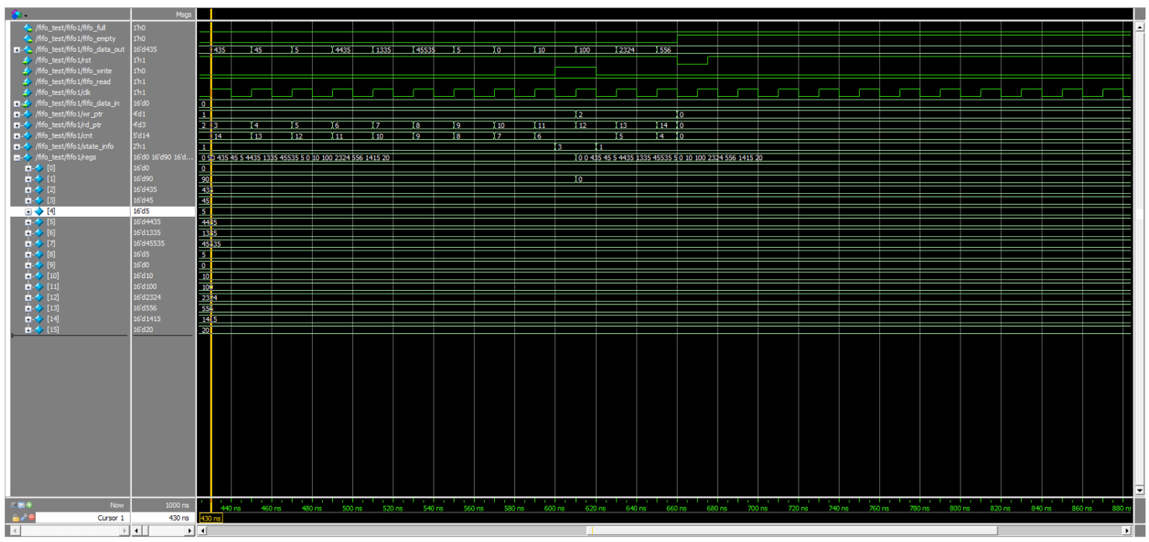
FIFO Testbench

Το testbench έχει ίδια δομή με πριν με τη διαφορά ότι όταν γίνεται bind το module με τα assertions πρέπει να περαστούν και τα εσωτερικά σήματα cnt, write_pointer και read_pointer στα assertions.

```
bind fifo fifo_property #(16,16) fifobound (.prst(rst),
↪  .pfifo_write(fifo_write), .pfifo_read(fifo_read), .pclk(clk),
↪  .pfifo_full(fifo_full), .pfifo_empty(fifo_empty),
↪  .pfifo_data_in(fifo_data_in), .pfifo_data_out(fifo_data_out),
↪  .pcnt(fifo.cnt), .pwr_ptr(fifo.wr_ptr), .prd_ptr(fifo.rd_ptr));
```

Όπως και στο προηγούμενο tb, ελέγχεται η λειτουργία του FIFO δοκιμάζοντας εγγραφές και αναγνώσεις (ταυτόγχρονα και μη) καθώς και άκυρες εγγραφές και αναγνώσεις για να βεβαιωθεί η λειτουργία των assertions.




/fifo_test/fifo1/fifo_full
/fifo_test/fifo1/fifo_empty
/fifo_test/fifo1/fifo_data_out
/fifo_test/fifo1/rst
/fifo_test/fifo1/fifo_write
/fifo_test/fifo1/fifo_read
/fifo_test/fifo1/clk
/fifo_test/fifo1/fifo_data_in
/fifo_test/fifo1/wr_ptr
/fifo_test/fifo1/rd_ptr
/fifo_test/fifo1/cnt
/fifo_test/fifo1/state_info
/fifo_test/fifo1/regs
Now
Cursor 1