

Name:kirollos gerges sobhy

Fifo verification

Test plan:

1-test the reset

2-test the write with random data

3-test the output after putting the data in a queue we made in the tetst

4-make random read and write and rest all of them is random

5- I made directed test to test the overflow and underflow

Assertions:

- `a1`: This property checks that the signal `f.full` goes low after `f.wr\_ack` is asserted, indicating that a write operation has been acknowledged and space is available in the buffer.

```
@(posedge f.clk) f.full | => !f.wr_ack;
```

- `a2`: This property checks that when the signal `f.wr\_ack` is asserted and the buffer is almost full, then the `f.full` signal is asserted, indicating that the buffer is now full.

```
@(posedge f.clk) (f.almostfull ##1 f.wr_ack) | => f.full;
```

- `a3`: This property checks that when both the `f.rd\_en` and `f.almostempty` signals are asserted, then the `f.empty` signal is asserted, indicating that the buffer is now empty.

```
@(posedge f.clk) f.rd_en & f.almostempty | => f.empty;
```

- `a4`: This property checks that when the `f.rd\_en` signal is asserted and the buffer is empty, then the `f.underflow` signal is asserted, indicating that a read operation was attempted when the buffer was already empty.

```
@(posedge f.clk) (f.empty & f.rd_en) | -> f.underflow;
```

- `a5`: This property checks that when the `f.wr\_en` signal is asserted and the buffer is full, then the `f.overflow` signal is asserted, indicating that a write operation was attempted when the buffer was already full.

```
@(posedge f.clk) (f.full & f.wr_en) | => f.overflow;
```

## Code of the top and interface of the fifo

```
/* module: fifo_interface
 * author: kirollos gerges sobhy
 */
interface FIFO_if(clk);
    * input clk
    input bit clk;
    * input design
    logic [15:0] data_in;
    logic wr_en;
    logic rd_en;
    logic rst_n;
    * output design
    logic [15:0] data_out;
    logic full;
    logic almostfull;
    logic empty;
    logic almostempty;
    logic overflow;
    logic underflow;
    logic wr_ack;
    * modports
    modport DUT(input clk, data_in, wr_en, rd_en, rst_n, output data_out, full, almostfull, empty, almostempty, overflow, underflow, wr_ack); //modports for the design
    modport TEST(input clk, data_out, full, almostfull, empty, almostempty, overflow, underflow, wr_ack, output data_in, wr_en, rd_en, rst_n); //modports for the testbench
endinterface
```

```
/* module: fifo_top
 * author: kirollos gerges sobhy
 */
module fifo_top();
    bit clk;
    * declaring the clock for the system
    initial begin
        clk=0;
        forever begin
            #1clk=~clk;
        end
    end
    * concatenate every module with the interface
    FIFO_if f (clk);
    FIFO_dut (f);
    fifo_tb tb(f);
    //bind FIFO fifo_sva fifo_sva_inst(f);
endmodule
```

The testing code and the assertions:

```
1  /*****
2  * module:fifo_test
3  * author:kirollos gerges sobhy
4  *****/
5
6  class fifo_class;
7      rand logic rst;
8      rand logic wr_en;
9      rand logic rd_en;
10     rand logic [15:0] data;
11     rand logic [15:0] other;
12     constraint c {
13         rst dist {0:=1,1:=99};
14         wr_en dist {0:=50,1:=50};
15         rd_en dist {0:=50,1:=50};
16         other != (16'h0000&&16'hffff);
17         data dist {other:=50,0:=50,16'hffff:=50};
18     }
19
20
21 endclass
22
23 module fifo_tb(FIFO_if.TEST f);
24     logic [15:0] fifo_queue[$];
25     logic [15:0] rand_test[10000];
26     int index;
27     int q_ind[$];
28     covergroup cov @(posedge f.clk);
29         rst_cover_point: coverpoint f.rst_n{
30             bins z_rst={0};
31             bins one_rst={1};
32         }
33         wr_cover_point: coverpoint f.wr_en{
34             bins z_w={0};
35             bins one_w={1};
36         }
37         rd_cover_point: coverpoint f.rd_en{
38             bins z_rd={0};
39             bins one_rd={1};
40         }
41         data_cover_point: coverpoint f.data_in{
42             bins max={16'hffff};
43             bins min =(16'h0000);
44             bins other =default;
45             bins min_max=(16'h0000=>16'hffff);
46             bins max_min=(16'hffff=>16'h0000);
47         }
48         c1: cross rst_cover_point,wr_cover_point{
49             bins wr_rst= binsof(rst_cover_point.one_rst) && binsof(wr_cover_point.one_w);
50         }
51     }
```

```

51     c2: cross rst_cover_point,rd_cover_point{
52         bins rd_rst= binsof(rst_cover_point.one_rst) && binsof(rd_cover_point.one_rd);
53     }
54
55 }
56
57
58 endgroup
59 cov cover_inst =new();
60 fifo_class cl=new();
61 initial begin
62     f.rd_en=0;//just to not see x
63     f.wr_en=0;// just for initialization
64     // first we reset the design before anything
65     reset();
66     //make the tests input
67     for(int i=0;i<10000;i++)begin
68         assert(cl.randomize());
69         rand_test[i]=cl.data;
70     end
71     //direct test to test the under flow flag
72     f.rd_en=1;
73     f.data_in=rand_test[0];
74     @(negedge f.clk);
75     f.rd_en=0;
76
77     // testing a the write of full fifo
78     index=0;
79     while (fifo_queue.size!=512)begin
80         assert (cl.randomize());
81         //f.wr_en=cl.wr_en;
82         f.wr_en=1;
83         if(f.wr_en==1)begin
84             f.data_in=rand_test[index];
85             fifo_queue.push_back(f.data_in);
86             index++;
87         end
88         @(negedge f.clk);
89     end
90
91     f.wr_en=0;
92     //direct test to test the over flow flag and assertions
93     f.wr_en=1;
94     f.data_in=$random;
95     @(negedge f.clk);
96     f.wr_en=0;
97     //testing the read from full fifo
98     while(fifo_queue.size!=0)begin
99         assert (cl.randomize());
100         //f.rd_en=cl.rd_en;

```

```

101         f.rd_en=1;
102         if(f.rd_en==1)begin
103             @(negedge f.clk);
104             check_read(fifo_queue.pop_front());
105         end
106
107
108
109     end
110     //random test
111
112     for(int i=0;i<10000;i++)begin
113         assert (cl.randomize());
114         f.wr_en=cl.wr_en;
115         f.rd_en=cl.wr_en;
116         f.rst_n=cl.rst;
117         if(f.rd_en==1&&f.rst_n==1&&f.wr_en==0&&f.empty==0)begin
118             @(negedge f.clk);
119             check_read(fifo_queue.pop_front());
120         end
121         if(f.wr_en==1&&f.rst_n==1&&f.rd_en==0&&f.full==0)begin
122             f.data_in=cl.data;
123             fifo_queue.push_back(f.data_in);
124             @(negedge f.clk);
125         end
126     end
127
128
129
130
131     $display("donne");
132     $stop;
133
134
135
136
137     end
138     task reset();
139         f.rst_n=0;
140         @(negedge f.clk);
141         f.rst_n=1;
142     endtask
143     task check_read(input logic [15:0] expected);
144         if(expected!=f.data_out) $display("there is an error the data out is=%0h and it was expected %0h",f.data_out,expected);
145     endtask
146     // assert that if the fifo is full then there is no write
147     property a1;
148         @(posedge f.clk) f.full |>= !f.wr_ack;
149     endproperty
150

```

```

150     endproperty
151     //this check is almostfull and then one write is done then full signal will get high
152     property a2;
153         @(posedge f.clk) (f.almostfull ##1 f.wr_ack) |>= f.full;
154     endproperty
155     //if almostfull is rise and then rd_en is one then empty flag must be set
156     property a3;
157         @(posedge f.clk) f.rd_en&f.almostempty |>= f.empty;
158     endproperty
159     // if empty is set and rd_en is set then under flow is done
160     property a4;
161         @(posedge f.clk) (f.empty &f.rd_en) |>= f.underflow;
162     endproperty
163     //if full is set and there is a write operation is done then over flow is set
164     property a5;
165         @(posedge f.clk) (f.full&f.wr_en) |>= f.overflow;
166     endproperty
167     assert1: assert property(a1);
168     assert2: assert property(a2);
169     assert3: assert property(a3);
170     assert4: assert property(a4);
171     assert5: assert property(a5);
172
173     c_assert1: cover property(a1);
174     c_assert2: cover property(a2);
175     c_assert3: cover property(a3);
176     c_assert4: cover property(a4);
177     c_assert5: cover property(a5);
178
179
180     endmodule

```

The snapshots:

## Functional coverage

Covergroup instance \fifo_top/tb/cover_inst	91.66%	100	-	Uncovered
covered/total bins:	16	18	-	
missing/total bins:	2	18	-	
% Hit:	88.88%	100	-	
Coverpoint rst_cover_point	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin z_rst	1	1	-	Covered
bin one_rst	10827	1	-	Covered
Coverpoint wr_cover_point	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin z_w	5397	1	-	Covered
bin one_w	5431	1	-	Covered
Coverpoint rd_cover_point	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin z_rd	5432	1	-	Covered
bin one_rd	5396	1	-	Covered
Coverpoint data_cover_point	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
bin max	3373	1	-	Covered
bin min	3300	1	-	Covered
bin min_max	557	1	-	Covered
bin max_min	534	1	-	Covered
default bin other	4155		-	Occurred
Cross c1	75.00%	100	-	Uncovered
covered/total bins:	3	4	-	
missing/total bins:	1	4	-	
% Hit:	75.00%	100	-	
Auto, Default and User Defined Bins:				
bin wr_rst	5431	1	-	Covered
bin <one_rst,z_w>	5396	1	-	Covered
bin <z_rst,z_w>	1	1	-	Covered
bin <^,one_w>	0	1	1	ZERO
Cross c2	75.00%	100	-	Uncovered
covered/total bins:	3	4	-	
missing/total bins:	1	4	-	
% Hit:	75.00%	100	-	
Auto, Default and User Defined Bins:				
bin rd_rst	5396	1	-	Covered
bin <one_rst,z_rd>	5431	1	-	Covered
bin <z_rst,z_rd>	1	1	-	Covered

Cross c2	75.00%	100	-	Uncovered
covered/total bins:	3	4	-	
missing/total bins:	1	4	-	
% Hit:	75.00%	100	-	
Auto, Default and User Defined Bins:				
bin rd_rst	5396	1	-	Covered
bin <one_rst,z_rd>	5431	1	-	Covered
bin <z_rst,z_rd>	1	1	-	Covered
bin <*,one_rd>	0	1	1	ZERO
Directive Coverage:				
Directives	5	5	0	100.00%
DIRECTIVE COVERAGE:				
-----				
Name	Design Unit	Design UnitType	Lang File(Line)	Hits Status
-----				
/fifo_top/tb/c_assert1	fifo_tb	Verilog	SVA fifo_tb.sv(173)	2 Covered
/fifo_top/tb/c_assert2	fifo_tb	Verilog	SVA fifo_tb.sv(174)	1 Covered
/fifo_top/tb/c_assert3	fifo_tb	Verilog	SVA fifo_tb.sv(175)	94 Covered
/fifo_top/tb/c_assert4	fifo_tb	Verilog	SVA fifo_tb.sv(176)	1 Covered
/fifo_top/tb/c_assert5	fifo_tb	Verilog	SVA fifo_tb.sv(177)	1 Covered

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_cover
/fifo_top/tb		91.66%						
TYPE cov		91.66%	100	91.66%	<div><div></div></div>	✓		auto(0)
CVP cov::r...		100.00%	100	100.00...	<div><div></div></div>	✓		
CVP cov::i...		100.00%	100	100.00...	<div><div></div></div>	✓		
CVP cov::r...		100.00%	100	100.00...	<div><div></div></div>	✓		
CVP cov::d...		100.00%	100	100.00...	<div><div></div></div>	✓		
CROSS co...		75.00%	100	75.00%	<div><div></div></div>	✓		
CROSS co...		75.00%	100	75.00%	<div><div></div></div>	✓		
INST V/fifo...		91.66%	100	91.66%	<div><div></div></div>	✓		



The assertions coverage:

The screenshot shows the Xilinx Vivado IDE interface. The top panel displays the Assertions coverage window, which lists several assertions for the module /fifo\_top/tb/c\_asse... SVA. The bottom panel shows the Transcript window, which contains the compilation and simulation log.

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cnplt %	Cnplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/fifo_top/tb/c_asse... SVA	SVA	✓	Off	2	1	Unl...	1	100%	✓	✓	0	0	0 ns	0
/fifo_top/tb/c_asse... SVA	SVA	✓	Off	1	1	Unl...	1	100%	✓	✓	0	0	0 ns	0
/fifo_top/tb/c_asse... SVA	SVA	✓	Off	94	1	Unl...	1	100%	✓	✓	0	0	0 ns	0
/fifo_top/tb/c_asse... SVA	SVA	✓	Off	1	1	Unl...	1	100%	✓	✓	0	0	0 ns	0
/fifo_top/tb/c_asse... SVA	SVA	✓	Off	1	1	Unl...	1	100%	✓	✓	0	0	0 ns	0

```
-- Compiling module fifo_tb
-- Compiling module fifo_top
#
# Top level modules:
#   fifo_top
# End time: 19:25:53 on May 01, 2023, Elapsed time: 0:00:00
# Errors: 0, Warnings: 1
# vsim -voptargs="+acc" work.fifo_top --coverage
# Start time: 19:25:53 on May 01, 2023
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
# Loading sv_std.std
# Loading work.fifo_top(fast)
# Loading work.FIFO_tf(fast__1)
# Loading work.FIFO(fast)
# Loading work.fifo_tb_sv_unit(fast)
# Loading work.fifo_tb(fast)
# done
# ** Note: Setop : fifo_tb.sv(133)
# Time: 21656 ns Iteration: 1 Instance: /fifo_top/tb
# Break in Module fifo_tb at fifo_tb.sv line 133
```