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
 * auther: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
 * auther:kirollos gerges sobhy
 *************************************************************************/
module fifo_top();
    bit clk;
    /**************************************************************************************************
     * declaring the clock for the system
     *************************************************************************************************/
    initial begin
        clk=0;
        forever begin
            #1clk=~clk;
        end
    end
    /**************************************************************************
     * concatinate 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:

```
/*******************************************************************************
 * module:fifo_test
 * auther:kirollos gerges sobhy
 *******************************************************************************/

class fifo_class;
    rand logic rst;
    rand logic wr_en;
    rand logic rd_en;
    rand logic [15:0] data;
    rand logic [15:0] other;
    constraint c {
        rst dist {0:=1,1:=99};
        wr_en dist {0:=50,1:=50};
        rd_en dist {0:=50,1:=50};
        other != (16'h0000&&16'hffff);
        data dist {other:=50,0:=50,16'hffff:=50};
    }


endclass

module fifo_tb(FIFO_if.TEST f);
    logic [15:0] fifo_queue[$];
    logic [15:0] rand_test[10000];
    int index;
    int q_ind[$];
    covergroup cov @(posedge f.clk);
        rst_cover_point: coverpoint f.rst_n{
        bins z_rst={0};
        bins one_rst={1};
        }
        wr_cover_point: coverpoint f.wr_en{
        bins z_w={0};
        bins one_w={1};
        }
        rd_cover_point: coverpoint f.rd_en{
        bins z_rd={0};
        bins one_rd={1};
        }
        data_cover_point: coverpoint f.data_in{
        bins max={16'hffff};
        bins min ={16'h0000};
        bins other =default;
        bins min_max=(16'h0000=>16'hffff);
        bins max_min=(16'hffff=>16'h0000);
        }
        c1: cross rst_cover_point,wr_cover_point{
            bins wr_rst= binsof(rst_cover_point.one_rst) && binsof(wr_cover_point.one_w);
        }
```

```
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          //dirct 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
90          end
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          //tetsting the read from full fifo
98          while(fifo_queue.size!=0)begin
99              assert (cl.randomize());
100             //f.rd_en=cl.rd_en;
```

```
            f.rd_en=1;
            if(f.rd_en==1)begin
                @(negedge f.clk);
                check_read(fifo_queue.pop_front());
            end



        end
        //random test

        for(int i=0;i<10000;i++)begin
            assert (cl.randomize());
            f.wr_en=cl.wr_en;
            f.rd_en=!cl.wr_en;
            f.rst_n=cl.rst;
            if(f.rd_en==1&&f.rst_n==1&&f.wr_en==0&&f.empty==0)begin
                @(negedge f.clk);
                check_read(fifo_queue.pop_front());
            end
            if(f.wr_en==1&&f.rst_n==1&&f.rd_en==0&&f.full==0)begin
                f.data_in=cl.data;
                fifo_queue.push_back(f.data_in);
                @(negedge f.clk);
            end

        end




        $display("donne");
        $stop;



    end
    task reset();
        f.rst_n=0;
        @(negedge f.clk);
        f.rst_n=1;

    endtask
    task check_read(input logic [15:0] expected);
        if(expected!==f.data_out) $display("there is an error the data out is=%0h and it was expected %0h",f.data_out,expected);
    endtask
    // assert that if the fifo is full then there is no write
    property a1;
        @(posedge f.clk) f.full |=> !f.wr_ack;
    endproperty
```

```
    endproperty
        //this check is almostfull and then one write is done then full signal will get high
    property a2;
        @(posedge f.clk) (f.almostfull ##1 f.wr_ack) |=> f.full;
    endproperty
        //if almostfull is rise and then rd_en is one then empty flag must be set
    property a3;
        @(posedge f.clk) f.rd_en&f.almostempty |=> f.empty;
    endproperty
        // if empty is set and rd_en is set then under flow is done
    property a4;
        @(posedge f.clk) (f.empty &f.rd_en) |-> f.underflow;
    endproperty
        //if full is set and there is a write operation is done then over fllow is set
    property a5;
        @(posedge f.clk) (f.full&f.wr_en) |=> f.overflow;
    endproperty
    assert1: assert property(a1);
    assert2: assert property(a2);
    assert3: assert property(a3);
    assert4: assert property(a4);
    assert5: assert property(a5);

    c_assert1: cover property(a1);
    c_assert2: cover property(a2);
    c_assert3: cover property(a3);
    c_assert4: cover property(a4);
    c_assert5: cover property(a5);


endmodule
```

The snapshots:

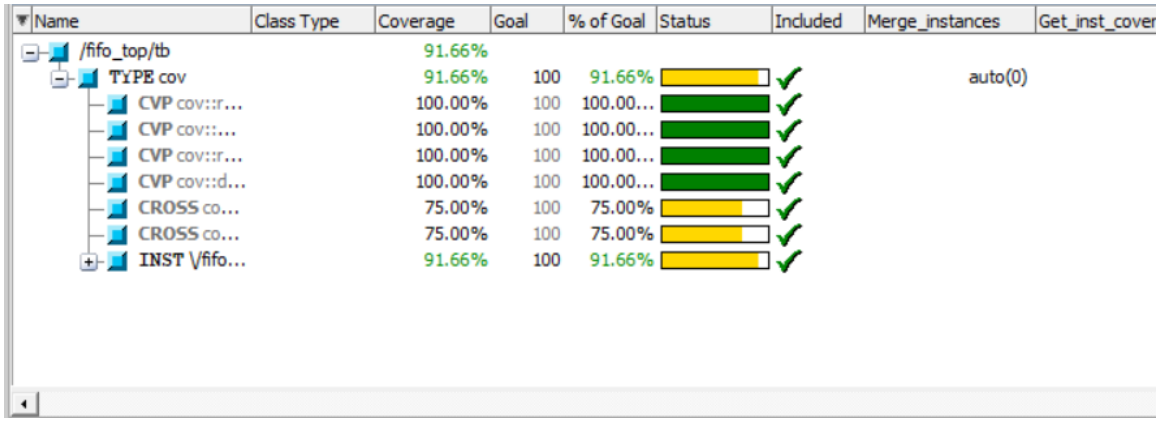
Functional coverage

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 Design   Lang File(Line)      Hits Status
                                         Unit   UnitType
--------------------------------------------------------------------------------------------
/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
```



The assertions coverage: