Yousef Ahmed Mohamed

Verification Project

FIFO

OVERVIEW

The synchronous FIFO (First-In, First-Out) buffer is designed to store and release data sequentially, ensuring that the first data written is the first read out. It acts as a rate buffer between modules operating in the same clock domain.
while adhering strictly to the FIFO principle.

Key Design Features

The FIFO design includes essential control and status mechanisms to ensure reliable operation:

- Write/Read Pointers: Manage data flow and track read/write positions with automatic wraparound for continuous operation.
- Status Flags: Full, Empty, Almost Full, and Almost Empty indicators provide real-time buffer state awareness.
- Error Handling: Overflow and Underflow flags detect invalid write or read operations, preventing data corruption.
- These features collectively enable smooth, ordered, and efficient data transfer within synchronous systems.

VERIFIVATION OVERVIEW

The FIFO verification environment is organized modularly to separate stimulus generation, coverage collection, and checking. It follows a transaction-based testbench structure, ensuring scalability and clarity.

- **Top Module:**
  - Generates clock signal, instantiates the interface, connects it to the DUT, testbench, and monitor, and manages simulation flow and coverage saving.
- **Interface:**
  - Serves as the communication bridge between the DUT and testbench, containing all input/output signals.
- **Testbench:**
  - Applies randomized stimulus to the DUT through the interface using the FIFO_transaction class with constrained random distributions. It coordinates the test sequence (reset → randomize → drive → complete) and signals test completion through the test_finished flag defined in shared_pkg.
- **Monitor:**
  - Samples the interface each cycle, captures transactions, collects coverage, and checks DUT outputs via the scoreboard. It runs coverage and checking in parallel and prints error logs then ends simulation when test_finished is asserted .

VERIFIVATION OVERVIEW

- **Transaction Class:**
  - Stores DUT signals, defines reset and enable constraints, and generates randomized input patterns for functional verification.
- **Coverage Class:**
  - Uses a covergroup to record cross coverage between: wr_en, rd_en and FIFO status signals
  - Ensures all possible write/read and state combinations are tested.
  - Samples coverage data using the sample_data() method.
- **Scoreboard class:**
  - Implements a reference model that predicts expected FIFO outputs.
  - Compares DUT outputs against the reference model:
  - 1) Increments correct_count on a match and Increments error_count and prints details on mismatch.
- **Assertions (inside DUT)**
  - • Enabled only in simulation using ifdef SIM.
  - • Verify main design rules:
  - 1) Reset clears all pointers and counters.
  - 2) wr_ack asserted only on valid writes.
  - 3) overflow occurs only when writing while full.
  - 4) underflow occurs only when reading while empty.
  - 5) Status flags (full, empty, almostfull, almostempty) reflect correct count.
  - 6) Pointer wraparound functions correctly.
  - • Provide additional assertion coverage during simulation.

DESIGN BUGS REPORT

1. Incomplete Reset Implementation in write and read logic.
   - Before:

```
always @(posedge intrf.clk or negedge intrf.rst_n) begin
    if (!intrf.rst_n) begin
        wr_ptr <= 0;
    end
```

```
always @(posedge intrf.clk or negedge intrf.rst_n) begin
    if (!intrf.rst_n) begin
        rd_ptr <= 0;
    end
```

   - After:

```
always @(posedge intrf.clk or negedge intrf.rst_n) begin
    if (!intrf.rst_n) begin
        wr_ptr <= 0;
        intrf.overflow <= 0; //<<<<<<
        intrf.wr_ack <= 0;   //<<<<<<
    end
```

```
always @(posedge intrf.clk or negedge intrf.rst_n) begin
    if (!intrf.rst_n) begin
        rd_ptr <= 0;
        intrf.underflow <= 0; //<<<<<<
    end
```

data_out *signal doesnt have to be reset.*

DESIGN BUGS REPORT

2. Read and write pointers are not safely wrapped arround.

- Before:

```
else if (intrf.wr_en && count < intrf.FIFO_DEPTH) begin
    mem[wr_ptr] <= intrf.data_in;
    intrf.wr_ack <= 1;
    intrf.overflow <= 0;
end
```

```
else if (intrf.rd_en && count != 0) begin
    intrf.data_out <= mem[rd_ptr];
    intrf.underflow <= 0;
end
```

- After:

```
else if (intrf.wr_en && count < intrf.FIFO_DEPTH) begin
    mem[wr_ptr] <= intrf.data_in;
    intrf.wr_ack <= 1;
    wr_ptr <= (wr_ptr == intrf.FIFO_DEPTH-1)? 0: wr_ptr + 1; //<<<<<<
    intrf.overflow <= 0;
end
```

```
else if (intrf.rd_en && count != 0) begin
    intrf.data_out <= mem[rd_ptr];
    rd_ptr = (rd_ptr == intrf.FIFO_DEPTH-1)? 0: rd_ptr + 1; //<<<<<<
    intrf.underflow <= 0;
end
```

DESIGN BUGS REPORT

3. Underflow logic should be sequenctial in read logic, not combinational.

- Before:

```
assign underflow = (empty && rd_en)? 1 : 0;
```

- After:

```
else
    if(intrf.rd_en && count == 0)
        intrf.underflow <= 1; //<<<<<<
    else
        intrf.underflow <= 0; //<<<<<<
```

DESIGN BUGS REPORT

4. The code doesn't handle read and write operations in the same cycle.

- Before:

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({wr_en, rd_en} == 2'b10) && !full)
            count <= count + 1;
        else if ( ({wr_en, rd_en} == 2'b01) && !empty)
            count <= count - 1;
    end
end
```

- After:

```
else begin
    if  (({intrf.wr_en, intrf.rd_en} == 2'b10) && !intrf.full)
        count <= count + 1;
    else if ( ({intrf.wr_en, intrf.rd_en} == 2'b01) && !intrf.empty)
        count <= count - 1;

    else if ( ({intrf.wr_en, intrf.rd_en} == 2'b11) && intrf.empty)  //<<<<<<
        count <= count + 1;
    else if ( ({intrf.wr_en, intrf.rd_en} == 2'b11) && intrf.full)  //<<<<<<
        count <= count - 1;
end
```

DESIGN BUGS REPORT

5. Incorrect almostfull logic:

- Before:

```
assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
```

- After:

```
assign intrf.almostfull = (count == intrf.FIFO_DEPTH-1)? 1 : 0;
```

FULL DESIGN BEFORE DEBUGGING

```
module FIFO(data_in, wr_en, rd_en, clk, rst_n, full, empty, almostfull,
            almostempty, wr_ack, overflow, underflow, data_out);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
input [FIFO_WIDTH-1:0] data_in;
input clk, rst_n, wr_en, rd_en;
output reg [FIFO_WIDTH-1:0] data_out;
output reg wr_ack, overflow;
output full, empty, almostfull, almostempty, underflow;

localparam max_fifo_addr = $clog2(FIFO_DEPTH);

reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];

reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
reg [max_fifo_addr:0] count;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        wr_ptr <= 0;
    end
    else if (wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= data_in;
        wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
    end
    else begin
        wr_ack <= 0;
        if (full & wr_en)
            overflow <= 1;
        else
            overflow <= 0;
    end
end
```

FULL DESIGN BEFORE DEBUGGING

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        rd_ptr <= 0;
    end
    else if (rd_en && count != 0) begin
        data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({wr_en, rd_en} == 2'b10) && !full)
            count <= count + 1;
        else if ( ({wr_en, rd_en} == 2'b01) && !empty)
            count <= count - 1;
    end
end

assign full = (count == FIFO_DEPTH)? 1 : 0;
assign empty = (count == 0)? 1 : 0;
assign underflow = (empty && rd_en)? 1 : 0;
assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
assign almostempty = (count == 1)? 1 : 0;

endmodule
```

FULL DESIGN AFTER DEBUGGING

```
module FIFO (FIFO_if intrf);

localparam max_fifo_addr = $clog2(intrf.FIFO_DEPTH);

reg [intrf.FIFO_WIDTH-1:0] mem [intrf.FIFO_DEPTH-1:0];

reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
reg [max_fifo_addr:0] count;

always @(posedge intrf.clk or negedge intrf.rst_n) begin
    if (!intrf.rst_n) begin
        wr_ptr <= 0;
        intrf.overflow <= 0; //<<<<<<
        intrf.wr_ack <= 0; //<<<<<<
    end
    else if (intrf.wr_en && count < intrf.FIFO_DEPTH) begin
        mem[wr_ptr] <= intrf.data_in;
        intrf.wr_ack <= 1;
        wr_ptr <= (wr_ptr == intrf.FIFO_DEPTH-1)? 0: wr_ptr + 1; //<<<<<<
        intrf.overflow <= 0;
    end
    else begin
        intrf.wr_ack <= 0;
        if (intrf.full && intrf.wr_en)
            intrf.overflow <= 1;
        else
            intrf.overflow <= 0;
    end
end
```

FULL DESIGN AFTER DEBUGGING

```
always @(posedge intrf.clk or negedge intrf.rst_n) begin
    if (!intrf.rst_n) begin
        rd_ptr <= 0;
        intrf.underflow <= 0; //<<<<<<
    end
    else if (intrf.rd_en && count != 0) begin
        intrf.data_out <= mem[rd_ptr];
        rd_ptr = (rd_ptr == intrf.FIFO_DEPTH-1)? 0: rd_ptr + 1; //<<<<<<
        intrf.underflow <= 0;
    end
    else
        if(intrf.rd_en && count == 0)
            intrf.underflow <= 1; //<<<<<<
        else
            intrf.underflow <= 0; //<<<<<<
end

always @(posedge intrf.clk or negedge intrf.rst_n) begin
    if (!intrf.rst_n) begin
        count <= 0;
    end
    else begin
        if  (({intrf.wr_en, intrf.rd_en} == 2'b10) && !intrf.full)
            count <= count + 1;
        else if ( ({intrf.wr_en, intrf.rd_en} == 2'b01) && !intrf.empty)
            count <= count - 1;

        else if ( ({intrf.wr_en, intrf.rd_en} == 2'b11) && intrf.empty)  //<<<<<<
            count <= count + 1;
        else if ( ({intrf.wr_en, intrf.rd_en} == 2'b11) && intrf.full)  //<<<<<<
            count <= count - 1;
    end
end

assign intrf.full = (count == intrf.FIFO_DEPTH)? 1 : 0;
assign intrf.empty = (count == 0)? 1 : 0;
assign intrf.almostfull = (count == intrf.FIFO_DEPTH-1)? 1 : 0;
assign intrf.almostempty = (count == 1)? 1 : 0;
```

ASSERTIONS (IN SAME DESIGN FILE)

```
always_comb begin
    if (!intrf.rst_n)
        a_rst: assert final (rd_ptr == 0 && wr_ptr == 0 && count == 0);

end

property full_p1;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    (count == intrf.FIFO_DEPTH) |-> (intrf.full);
endproperty

property full_p2;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    (count != intrf.FIFO_DEPTH) |-> (!intrf.full);
endproperty

property wr_ack_p1;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    (intrf.wr_en && !intrf.full) |-> ##1 (intrf.wr_ack);
endproperty

property wr_ack_p2;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    !(intrf.wr_en && !intrf.full) |-> ##1 (!intrf.wr_ack);
endproperty

property overflow_p1;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    (intrf.wr_en && intrf.full) |-> ##1 (intrf.overflow);
endproperty

property overflow_p2;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    !(intrf.wr_en && intrf.full) |-> ##1 (!intrf.overflow);
endproperty

property empty_p1;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    (count == 0) |-> (intrf.empty);
endproperty

property empty_p2;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    (count != 0) |-> (!intrf.empty);
endproperty
```

ASSERTIONS (IN SAME DESIGN FILE)

```
property almostempty_p1;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    (count == 1) |-> (intrf.almostempty);
endproperty

property almostempty_p2;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    (count != 1) |-> (!intrf.almostempty);
endproperty

property underflow_p1;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    (intrf.rd_en && intrf.empty) |-> ##1 (intrf.underflow);
endproperty

property underflow_p2;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    !(intrf.rd_en && intrf.empty) |-> ##1 (!intrf.underflow);
endproperty

property almostfull_p1;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    (count == intrf.FIFO_DEPTH-1) |-> (intrf.almostfull);
endproperty

property almostfull_p2;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    (count != intrf.FIFO_DEPTH-1) |-> (!intrf.almostfull);
endproperty

property wr_p_wrap;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    (intrf.wr_en && !intrf.full && wr_ptr == intrf.FIFO_DEPTH-1) |-> ##1 (wr_ptr ==0);
endproperty

property rd_p_wrap;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    (intrf.rd_en && !intrf.empty && rd_ptr == intrf.FIFO_DEPTH-1) |-> ##1 (rd_ptr ==0);
endproperty

property count_wrap;
    @(posedge intrf.clk)
    ($past(count) == intrf.FIFO_DEPTH && intrf.rst_n == 0) |-> (count ==0);
endproperty

property wr_threshold;
    @(posedge intrf.clk) disable iff(!intrf.rst_n)
    (wr_ptr < intrf.FIFO_DEPTH);
endproperty
```

ASSERTIONS (IN SAME DESIGN FILE)

```
    property rd_threshold;
        @(posedge intrf.clk) disable iff(!intrf.rst_n)
        (rd_ptr < intrf.FIFO_DEPTH);
    endproperty

    property count_threshold;
        @(posedge intrf.clk) disable iff(!intrf.rst_n)
        (count < intrf.FIFO_DEPTH+1);
    endproperty

    `ifdef SIM
        full_a1:       assert property(full_p1);
        full_a2:       assert property(full_p2);
        wr_ack_a1:     assert property(wr_ack_p1);
        wr_ack_a2:     assert property(wr_ack_p2);
        overflow_a1:   assert property(overflow_p1);
        overflow_a2:   assert property(overflow_p2);
        empty_a1:      assert property(empty_p1);
        empty_a2:      assert property(empty_p2);
        underflow_a1: assert property(underflow_p1);
        underflow_a2: assert property(underflow_p2);
        almostempty_a1: assert property(almostempty_p1);
        almostempty_a2: assert property(almostempty_p2);
        almostfull_a1: assert property(almostfull_p1);
        almostfull_a2: assert property(almostfull_p2);
        wr_p_wrap_a: assert property(wr_p_wrap);
        rd_p_wrap_a: assert property(rd_p_wrap);
        count_wrap_a: assert property(count_wrap);
        wr_p_threshold_a: assert property(wr_threshold);
        rd_p_threshold_a: assert property(rd_threshold);
        count_threshold_a: assert property(count_threshold);
    `endif

        full_c1:       cover property(full_p1);
        full_c2:       cover property(full_p2);
        wr_ack_c1:     cover property(wr_ack_p1);
        wr_ack_c2:     cover property(wr_ack_p2);
        overflow_c1:   cover property(overflow_p1);
        overflow_c2:   cover property(overflow_p2);
        empty_c1:      cover property(empty_p1);
        empty_c2:      cover property(empty_p2);
        underflow_c1: cover property(underflow_p1);
        underflow_c2: cover property(underflow_p2);
        almostempty_c1: cover property(almostempty_p1);
        almostempty_c2: cover property(almostempty_p2);
        almostfull_c1: cover property(almostfull_p1);
        almostfull_c2: cover property(almostfull_p2);
        wr_p_wrap_c: cover property(wr_p_wrap);
        rd_p_wrap_c: cover property(rd_p_wrap);
        count_wrap_c: cover property(count_wrap);
        wr_p_threshold_c: cover property(wr_threshold);
        rd_p_threshold_c: cover property(rd_threshold);
        count_threshold_c: cover property(count_threshold);
endmodule
```

VERIFICATION PLAN

Functionality Check	Functional Coverage	Stimulus Generation	Requirement	Label	
Assertions and **scoreboard** confirm all outputs reset .correctly	-	Reset is randomly asserted and deasserted during simulation while other signals are .randomized	When reset is asserted, all FIFO pointers, data, and flags must return to default values regardless of previous .random state	**Reset Behavior**	1
Scoreboard ensures the written data is correctly captured in the reference .model	Cross coverage of **wr_en** and **rd_en** with **.wr_ack** states	wr_en, data_in, and rd_en are randomized with constraints to favor legal operations while occasionally testing boundary .conditions	A valid write should occur only when the .FIFO is not full	**Write Operation**	2
Scoreboard compares DUT data_out against the expected output from the .reference model	-	rd_en and wr_en are randomized each cycle with legal and .illegal combinations	A read should occur only when the FIFO has valid data (i.e., not .(empty	**Read Operation**	3
Assertions verify the flags toggle precisely at the defined depth .boundaries	Cross cover bins for **wr_en** and **wr_en** with **full/ empty/almost full/almostem .pty**	Random writes/reads continuously change FIFO depth .from 0 to max	Flags must accurately reflect FIFO occupancy across all random depth .changes	**Full / Empty Flags**	4
Assertions check the overflow/underflo w flags match the corresponding .illegal conditions	Cover bins for **overflow/ underflow with wr_en and rd_en** .occurrences	Random stimulus occasionally forces writes on a full FIFO (Overflow) or reads on an empty FIFO .((Underflow	Illegal operations must assert the proper error .flags	**Overflow / Underflow**	5

SHARED PACKAGE

```
package shared_pkg;
    bit test_finished;

    event trigg;

    int count_ref;
    integer data_out_error_count, data_out_correct_count;
    integer full_error_count, full_correct_count;
    integer almostfull_error_count, almostfull_correct_count;
    integer empty_error_count, empty_correct_count;
    integer almostempty_error_count, almostempty_correct_count;
    integer wr_ack_error_count, wr_ack_correct_count;
    integer overflow_error_count, overflow_correct_count;
    integer underflow_error_count, underflow_correct_count;

endpackage
```

INTERFACE

```
interface FIFO_if(input clk);

    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;

    logic [FIFO_WIDTH-1:0]  data_in;
    logic                   rst_n;
    logic                   wr_en;
    logic                   rd_en;
    logic [FIFO_WIDTH-1:0]  data_out;
    logic                   wr_ack;
    logic                   overflow;
    logic                   full;
    logic                   empty;
    logic                   almostfull;
    logic                   almostempty;
    logic                   underflow;

endinterface
```

TRANSACTION PACKAGE

```
package transaction_pkg;
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;

    class FIFO_transaction;
        rand logic [FIFO_WIDTH-1:0] data_in;
        rand logic wr_en;
        rand logic rd_en;
        rand logic rst_n;
        logic [FIFO_WIDTH-1:0] data_out;
        logic full;
        logic almostfull;
        logic empty;
        logic almostempty;
        logic overflow;
        logic underflow;
        logic wr_ack;
        int RD_EN_ON_DIST;
        int WR_EN_ON_DIST;

        function new(int RD_EN_ON_DIST = 30, int WR_EN_ON_DIST = 70);
            this.RD_EN_ON_DIST = RD_EN_ON_DIST;
            this.WR_EN_ON_DIST = WR_EN_ON_DIST;
        endfunction

        constraint reset_c {rst_n dist {0 := 1, 1 := 9};}
        constraint wr_en_c {wr_en dist {1 := WR_EN_ON_DIST, 0 := (100 - WR_EN_ON_DIST)};}
        constraint rd_en_c {rd_en dist {1 := RD_EN_ON_DIST, 0 := (100 - RD_EN_ON_DIST)};}
    endclass
endpackage
```

SCOREBOARD PACKAGE

```
package scoreboard_pkg;
    import shared_pkg::*;
    import transaction_pkg::*;

    class FIFO_scoreboard;
        parameter FIFO_WIDTH = 16;
        parameter FIFO_DEPTH = 8;

        logic  [FIFO_WIDTH-1:0] data_out_ref;
        logic full_ref;
        logic almostfull_ref;
        logic empty_ref;
        logic almostempty_ref;
        logic overflow_ref;
        logic underflow_ref;
        logic wr_ack_ref;

        logic [FIFO_WIDTH - 1:0] mem_ref [$];

        function void check_data(FIFO_transaction F_txn);
            reference_model(F_txn);
            if(F_txn.data_out != data_out_ref) begin
                $display("%t ERROR: data_out: %h, Expected: %h", $time(), F_txn.data_out, data_out_ref);
                data_out_error_count++;
            end
            else data_out_correct_count++;


            if(F_txn.full != full_ref) begin
                $display("%t ERROR: full = %h Expected: %h", $time(), F_txn.full, full_ref);
                full_error_count++;
            end
            else full_correct_count++;


            if(F_txn.almostfull != almostfull_ref) begin
                $display("%t ERROR: almostfull = %h, Expected = %h", $time(), F_txn.almostfull, almostfull_ref);
                almostfull_error_count++;
            end
            else almostfull_correct_count++;


            if(F_txn.empty != empty_ref) begin
                $display("%t ERROR: empty = %h, Expected = %h", $time(), F_txn.empty, empty_ref);
                empty_error_count++;
            end
            else empty_correct_count++;


            if(F_txn.almostempty != almostempty_ref) begin
                $display("%t ERROR: almostempty = %h, Expected = %h", $time(), F_txn.almostempty, almostempty_ref);
                almostempty_error_count++;
            end
            else almostempty_correct_count++;
```

SCOREBOARD PACKAGE

```
    if(F_txn.overflow != overflow_ref) begin
        $display("%t ERROR: overflow = %h, Expected = %h", $time(), F_txn.overflow, overflow_ref);
        overflow_error_count++;
    end
    else overflow_correct_count++;


    if(F_txn.underflow != underflow_ref) begin
        $display("%t ERROR: underflow = %h, Expected = %h", $time(), F_txn.underflow, underflow_ref);
        underflow_error_count++;
    end
    else underflow_correct_count++;


    if(F_txn.wr_ack != wr_ack_ref) begin
        $display("%t ERROR: wr_ack = %h, Expected = %h", $time(), F_txn.wr_ack, wr_ack_ref);
        wr_ack_error_count++;
    end
    else wr_ack_correct_count++;
endfunction


function void reference_model(FIFO_transaction F_txn);
    //rst_n
    if(!F_txn.rst_n) begin
        full_ref = 0;
        almostfull_ref = 0;
        empty_ref = 1;
        almostempty_ref = 0;
        overflow_ref = 0;
        underflow_ref = 0;
        wr_ack_ref = 0;
        count_ref = 0;
        mem_ref.delete();
    end
    else begin
        //write
        if(F_txn.wr_en && !full_ref) begin
            mem_ref.push_back(F_txn.data_in);
            wr_ack_ref = 1;
        end
        else wr_ack_ref = 0;

        //read
        if(F_txn.rd_en && !empty_ref)
            data_out_ref = mem_ref.pop_front();
```

SCOREBOARD PACKAGE

```
            //overflow and underflow
            if(full_ref && F_txn.wr_en) overflow_ref = 1;
            else overflow_ref = 0;

            if(empty_ref && F_txn.rd_en) underflow_ref = 1;
            else underflow_ref = 0;


            //counter
            if(F_txn.wr_en && !full_ref)
                count_ref++;

            if(F_txn.rd_en && !empty_ref)
                count_ref--;


            //full/empty
            if(count_ref == FIFO_DEPTH) full_ref = 1;
            else full_ref = 0;

            if(count_ref == 0) empty_ref = 1;
            else empty_ref = 0;

            if(count_ref == FIFO_DEPTH-1) almostfull_ref = 1;
            else almostfull_ref = 0;

            if(count_ref == 1) almostempty_ref = 1;
            else almostempty_ref = 0;
        end
    endfunction
endclass
endpackage
```

COVERAGE PACKAGE

```
package coverage_pkg;
import transaction_pkg::*;

    class FIFO_coverage;
        FIFO_transaction F_cvg_txn;

        covergroup FIFO_cvg;
            cp_wr_en: coverpoint F_cvg_txn.wr_en;
            cp_rd_en: coverpoint F_cvg_txn.rd_en;
            cp_full: coverpoint F_cvg_txn.full;
            cp_empty: coverpoint F_cvg_txn.empty;
            cp_almostempty: coverpoint F_cvg_txn.almostempty;
            cp_almostfull: coverpoint F_cvg_txn.almostfull;
            cp_overflow: coverpoint F_cvg_txn.overflow;
            cp_underflow: coverpoint F_cvg_txn.underflow;
            cp_wr_ack: coverpoint F_cvg_txn.wr_ack;

            c1: cross cp_wr_en, cp_rd_en, cp_full{
                ignore_bins b1_full = binsof(cp_wr_en) intersect {1}
                && binsof(cp_rd_en) intersect {1}
                && binsof(cp_full) intersect {1};

                ignore_bins b2_full = binsof(cp_wr_en) intersect {0}
                && binsof(cp_rd_en) intersect {1}
                && binsof(cp_full) intersect {1};
            }
            c2: cross cp_wr_en, cp_rd_en, cp_empty;
            c3: cross cp_wr_en, cp_rd_en, cp_almostfull;
            c4: cross cp_wr_en, cp_rd_en, cp_almostempty;
            c5: cross cp_wr_en, cp_rd_en, cp_overflow{
                ignore_bins b1_overflow = binsof(cp_wr_en) intersect {0}
                && binsof(cp_rd_en) intersect {1}
                && binsof(cp_overflow) intersect {1};

                ignore_bins b2_overflow = binsof(cp_wr_en) intersect {0}
                && binsof(cp_rd_en) intersect {0}
                && binsof(cp_overflow) intersect {1};
            }
            c6: cross cp_wr_en, cp_rd_en, cp_underflow{
                ignore_bins b1_underflow = binsof(cp_wr_en) intersect {1}
                && binsof(cp_rd_en) intersect {0}
                && binsof(cp_underflow) intersect {1};

                ignore_bins b2_underflow = binsof(cp_wr_en) intersect {0}
                && binsof(cp_rd_en) intersect {0}
                && binsof(cp_underflow) intersect {1};
            }
```

COVERAGE PACKAGE

```
            c7: cross cp_wr_en, cp_rd_en, cp_wr_ack{
                ignore_bins b1_full = binsof(cp_wr_en) intersect {0}
                && binsof(cp_wr_ack) intersect {1}
                && binsof(cp_wr_ack) intersect {1};

                ignore_bins b2_full = binsof(cp_wr_en) intersect {0}
                && binsof(cp_wr_ack) intersect {0}
                && binsof(cp_wr_ack) intersect {1};
            }
        endgroup

        function void sample_data(FIFO_transaction F_txn);
            F_cvg_txn = F_txn;
            FIFO_cvg.sample();
        endfunction

        function new();
            FIFO_cvg = new();
        endfunction
    endclass
endpackage
```

MONITOR PACKAGE

```
import transaction_pkg::*;
import coverage_pkg::*;
import scoreboard_pkg::*;
import shared_pkg::*;

module FIFO_monitor(FIFO_if intrf);
    FIFO_transaction trans;
    FIFO_coverage cov;
    FIFO_scoreboard score;

    initial begin
        trans = new();
        cov = new();
        score = new();

        forever begin
            wait(trigg.triggered);
            @(negedge intrf.clk)
            trans.data_in = intrf.data_in;
            trans.rst_n = intrf.rst_n;
            trans.wr_en = intrf.wr_en;
            trans.rd_en = intrf.rd_en;
            trans.data_out = intrf.data_out;
            trans.wr_ack = intrf.wr_ack;
            trans.overflow = intrf.overflow;
            trans.full = intrf.full;
            trans.empty = intrf.empty;
            trans.almostfull = intrf.almostfull;
            trans.almostempty = intrf.almostempty;
            trans.underflow = intrf.underflow;

            fork
                begin
                    cov.sample_data(trans);
                end

                begin
                    score.check_data(trans);
                end
            join
```

MONITOR

```
            if(test_finished) begin
                $display("================DATA OUT==================");
                $display("==== Error count: %0d", data_out_error_count);
                $display("==== Correct count: %0d", data_out_correct_count);
                $display("================FULL==================");
                $display("==== Error count: %0d ", full_error_count);
                $display("==== Correct count: %0d", full_correct_count);
                $display("================ALMOST FULL==================");
                $display("==== Error count: %0d", almostfull_error_count);
                $display("==== Correct count: %0d", almostfull_correct_count);
                $display("================EMPTY==================");
                $display("==== Error count: %0d", empty_error_count);
                $display("==== Correct count: %0d", empty_correct_count);
                $display("================ALMOST EMPTY==================");
                $display("==== Error count: %0d", almostempty_error_count);
                $display("==== Correct count: %0d", almostempty_correct_count);
                $display("================OVERFLOW==================");
                $display("==== Error count: %0d", overflow_error_count);
                $display("==== Correct count: %0d", overflow_correct_count);
                $display("================UNDERFLOW==================");
                $display("==== Error count: %0d", underflow_error_count);
                $display("==== Correct count: %0d", underflow_correct_count);
                $display("================ACKNOWLEDGE==================");
                $display("==== Error count: %0d", wr_ack_error_count);
                $display("==== Correct count: %0d", wr_ack_correct_count);
                $stop;
            end
        end
    end
endmodule
```

TESTBENCH

```
import transaction_pkg::*;
import shared_pkg::*;

module FIFO_TB (FIFO_if intrf);

    FIFO_transaction trans;
    initial begin
        data_out_error_count = 0;
        data_out_correct_count = 0;
        full_error_count = 0;
        full_correct_count = 0;
        almostfull_error_count = 0;
        almostfull_correct_count = 0;
        empty_error_count = 0;
        empty_correct_count = 0;
        almostempty_error_count = 0;
        almostempty_correct_count = 0;
        overflow_error_count = 0;
        overflow_correct_count = 0;
        underflow_error_count = 0;
        underflow_correct_count = 0;
        wr_ack_error_count = 0;
        wr_ack_correct_count = 0;

        trans = new();

        intrf.data_in = 0;
        intrf.wr_en = 0;
        intrf.rd_en = 0;

        //reset test
        intrf.rst_n = 0;
        #0; -> trigg;
        @(negedge intrf.clk);
        intrf.rst_n = 1;

        //random test
        repeat(100000) begin
            assert(trans.randomize());
            intrf.data_in = trans.data_in;
            intrf.wr_en = trans.wr_en;
            intrf.rd_en = trans.rd_en;
            intrf.rst_n = trans.rst_n;
            #0; -> trigg;
            @(negedge intrf.clk);
        end
        test_finished = 1;
        #0; -> trigg;
    end
endmodule
```

TOP MODULE

```
module FIFO_top();

    bit clk;
    always #1 clk = ~clk;

    FIFO_if intrf(clk);
    FIFO dut(intrf);
    FIFO_TB TB(intrf);
    FIFO_monitor mon(intrf);
endmodule
```

SOURCE FILE LIST

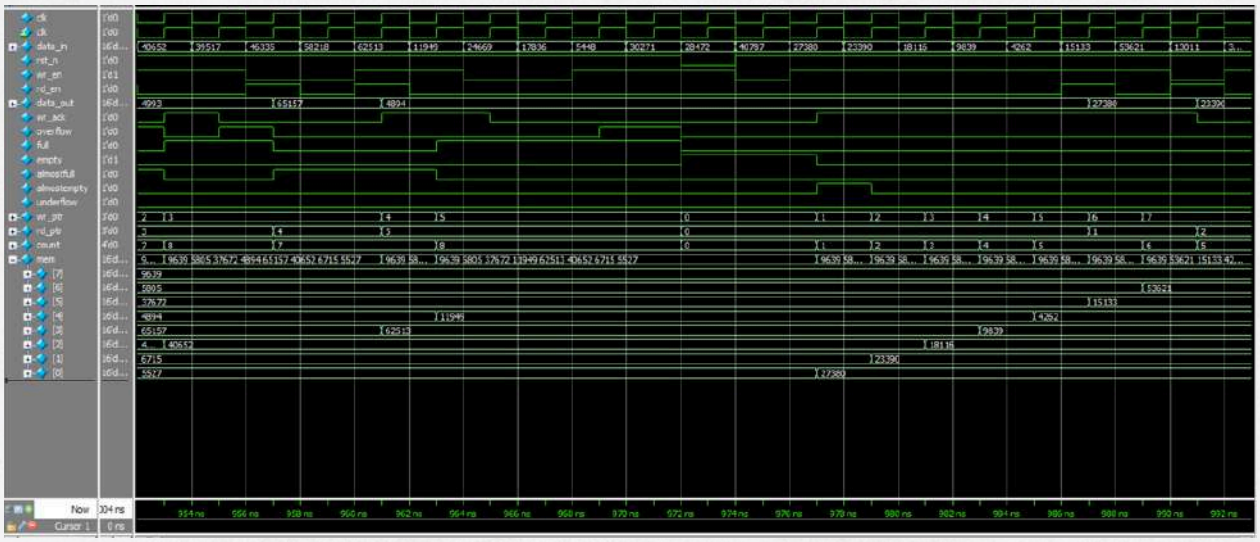
```
FIFO_pkg.sv
FIFO_transaction.sv
FIFO_coverage.sv
FIFO_scoreboard.sv
FIFO.sv
FIFO_monitor.sv
FIFO_tb.sv
top.sv
FIFO_if.sv
```

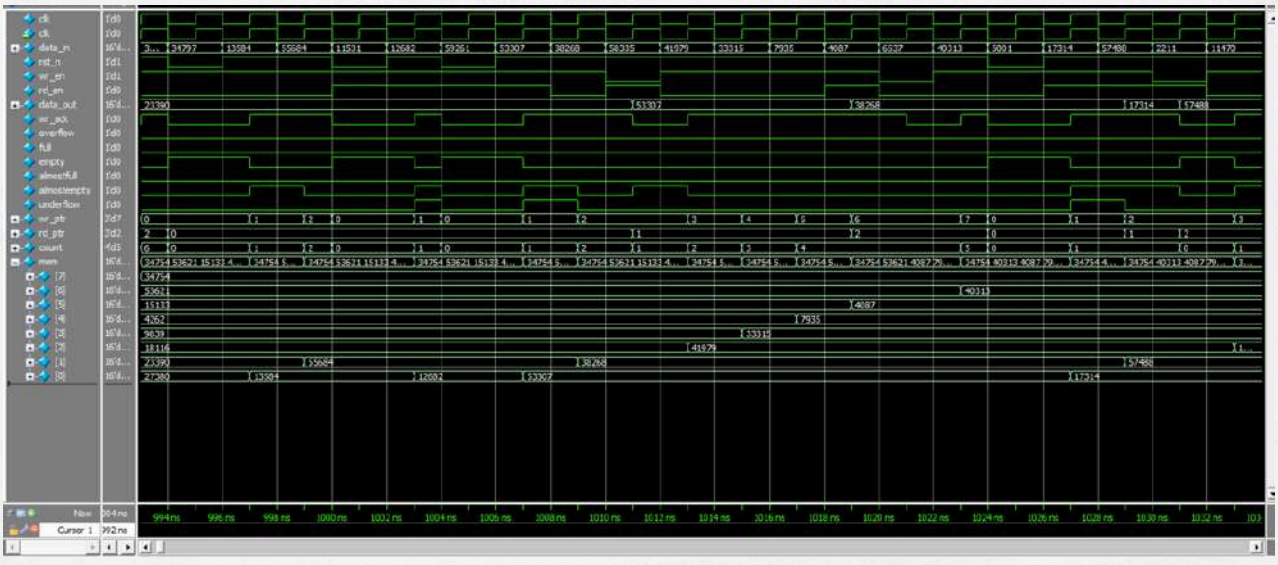
DO FILE

```
vdel -all
vlib work
vlog +define+SIM -f src_files.list  +cover -covercells
vsim -voptargs=+acc work.FIFO_top -cover
coverage save cov.ucdb -onexit
add wave *
add wave -position insertpoint sim:/FIFO_top/intrf/*
add wave -position insertpoint sim:/FIFO_top/dut/*
add wave -position insertpoint sim:/FIFO_top/dut.mem

run 0
run -all
```

QUESTASIM





QUESTASIM

```
# ================DATA OUT====================
# ==== Error count: 0
# ==== Correct count: 100002
# ================FULL====================
# ==== Error count: 0
# ==== Correct count: 100002
# ================ALMOST FULL====================
# ==== Error count: 0
# ==== Correct count: 100002
# ================EMPTY====================
# ==== Error count: 0
# ==== Correct count: 100002
# ================ALMOST EMPTY====================
# ==== Error count: 0
# ==== Correct count: 100002
# ================OVERFLOW====================
# ==== Error count: 0
# ==== Correct count: 100002
# ================UNDERFLOW====================
# ==== Error count: 0
# ==== Correct count: 100002
# ================ACKNOWLEDGE====================
# ==== Error count: 0
# ==== Correct count: 100002
# ** Note: $stop    : FIFO_monitor.sv(67)
#    Time: 200004 ns  Iteration: 1  Instance: /FIFO_top/mon
# Break in Module FIFO_monitor at FIFO_monitor.sv line 67
```

FUNCTIONAL COVERAGE

```
Covergroup Coverage:
    Covergroups                 1        na        na   100.00%
        Coverpoints/Crosses    16        na        na        na
            Covergroup Bins    66        66         0   100.00%
----------------------------------------------------------------
```

CODE COVERAGE

- Branch coverage:

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        30        30         0   100.00%

================================Branch Details================================

Branch Coverage for instance /FIFO_top/dut
```

- Statement coverage:

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      30        30         0   100.00%

================================Statement Details================================

Statement Coverage for instance /FIFO_top/dut --
```

- Toggle coverage:

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         20        20         0   100.00%

================================Toggle Details================================

Toggle Coverage for instance /FIFO_top/dut --
```

ASSERTION RESULTS

```
ASSERTION RESULTS:
--------------------------------------------------------------------
Name                File(Line)                  Failure      Pass
                                                Count        Count
--------------------------------------------------------------------
/FIFO_top/dut/a_rst  FIFO.sv(104)                     0          1
/FIFO_top/dut/full_a1
                    FIFO.sv(209)                      0          1
/FIFO_top/dut/full_a2
                    FIFO.sv(210)                      0          1
/FIFO_top/dut/wr_ack_a1
                    FIFO.sv(211)                      0          1
/FIFO_top/dut/wr_ack_a2
                    FIFO.sv(212)                      0          1
/FIFO_top/dut/overflow_a1
                    FIFO.sv(213)                      0          1
/FIFO_top/dut/overflow_a2
                    FIFO.sv(214)                      0          1
/FIFO_top/dut/empty_a1
                    FIFO.sv(215)                      0          1
/FIFO_top/dut/empty_a2
                    FIFO.sv(216)                      0          1
/FIFO_top/dut/underflow_a1
                    FIFO.sv(217)                      0          1
/FIFO_top/dut/underflow_a2
                    FIFO.sv(218)                      0          1
/FIFO_top/dut/almostempty_a1
                    FIFO.sv(219)                      0          1
/FIFO_top/dut/almostempty_a2
                    FIFO.sv(220)                      0          1
/FIFO_top/dut/almostfull_a1
                    FIFO.sv(221)                      0          1
/FIFO_top/dut/almostfull_a2
                    FIFO.sv(222)                      0          1
/FIFO_top/dut/wr_p_wrap_a
                    FIFO.sv(223)                      0          1
/FIFO_top/dut/rd_p_wrap_a
                    FIFO.sv(224)                      0          1
/FIFO_top/dut/count_wrap_a
                    FIFO.sv(225)                      0          1
/FIFO_top/dut/wr_p_threshold_a
                    FIFO.sv(226)                      0          1
/FIFO_top/dut/rd_p_threshold_a
                    FIFO.sv(227)                      0          1
/FIFO_top/dut/count_threshold_a
                    FIFO.sv(228)                      0          1
/FIFO_top/TB/#ublk#182146242#38/immed__39
                    FIFO_tb.sv(39)                    0          1
```

ASSERTION COVERAGE

```
DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                  Design Design   Lang File(Line)      Hits Status
                                      Unit   UnitType
--------------------------------------------------------------------------------------------
/FIFO_top/dut/full_c1                 FIFO   Verilog  SVA  FIFO.sv(231)    8979 Covered
/FIFO_top/dut/full_c2                 FIFO   Verilog  SVA  FIFO.sv(232)    80808 Covered
/FIFO_top/dut/wr_ack_c1               FIFO   Verilog  SVA  FIFO.sv(233)    50677 Covered
/FIFO_top/dut/wr_ack_c2               FIFO   Verilog  SVA  FIFO.sv(234)    29919 Covered
/FIFO_top/dut/overflow_c1             FIFO   Verilog  SVA  FIFO.sv(235)    5650 Covered
/FIFO_top/dut/overflow_c2             FIFO   Verilog  SVA  FIFO.sv(236)    74946 Covered
/FIFO_top/dut/empty_c1                FIFO   Verilog  SVA  FIFO.sv(237)    14450 Covered
/FIFO_top/dut/empty_c2                FIFO   Verilog  SVA  FIFO.sv(238)    75337 Covered
/FIFO_top/dut/underflow_c1            FIFO   Verilog  SVA  FIFO.sv(239)    3860 Covered
/FIFO_top/dut/underflow_c2            FIFO   Verilog  SVA  FIFO.sv(240)    76736 Covered
/FIFO_top/dut/almostempty_c1          FIFO   Verilog  SVA  FIFO.sv(241)    16181 Covered
/FIFO_top/dut/almostempty_c2          FIFO   Verilog  SVA  FIFO.sv(242)    73606 Covered
/FIFO_top/dut/almostfull_c1           FIFO   Verilog  SVA  FIFO.sv(243)    7634 Covered
/FIFO_top/dut/almostfull_c2           FIFO   Verilog  SVA  FIFO.sv(244)    82153 Covered
/FIFO_top/dut/wr_p_wrap_c             FIFO   Verilog  SVA  FIFO.sv(245)    3203 Covered
/FIFO_top/dut/rd_p_wrap_c             FIFO   Verilog  SVA  FIFO.sv(246)     649 Covered
/FIFO_top/dut/count_wrap_c            FIFO   Verilog  SVA  FIFO.sv(247)     924 Covered
/FIFO_top/dut/wr_p_threshold_c        FIFO   Verilog  SVA  FIFO.sv(248)    89787 Covered
/FIFO_top/dut/rd_p_threshold_c        FIFO   Verilog  SVA  FIFO.sv(249)    89787 Covered
/FIFO_top/dut/count_threshold_c       FIFO   Verilog  SVA  FIFO.sv(250)    89787 Covered

TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 20
```