Formal Verification 2025

FIFO Verification
Project1 Group32

Student Name: 吳家維、胡家豪

Student ID: N26134926、N26122246

目錄

1. FIFO Spec

本次實作的 FIFO Spec 以及 Port 如下

I/O type	bit(s)	port name	description
input	1	clk	Positive edge trigger clock
input	1	rstN	Negative edge trigger reset
input	1	write_en	High if and only if writing data
input	1	read_en	High if and only if reading data
input	4	write_data	Input data
output	4	read_data	Output data
output	1	full	High when the FIFO is full
output	1	empty	High when the FIFO is empty

深度以及資料位寬皆為 4，支援同時讀寫，進行讀操作時的當前 cycle 就會將讀資料輸出。

FIFO 可以分成三個部分: 讀操作、寫操作、空滿判斷

讀操作:

```
// Read Operation
always_ff@(posedge clk or negedge rstN) begin
  if(!rstN) begin
    read_ptr <= 3'd0;
  end else if(read_en && (!empty || write_en)) begin
    read_ptr <= read_ptr + 3'd1;
  end
end
// For Read Write at same time , since data is not written into memory
assign rd_wr_same_full = (read_en && write_en && full ); // For full condition check
assign rd_wr_same_empty = (read_en && write_en && empty);
assign read_data = (!rd_wr_same_empty)? mem[true_read_ptr] : write_data;
```

寫操作:

```
// Write Operation
always_ff@(posedge clk or negedge rstN) begin
  if(!rstN) begin
    write_ptr <= 3'd0;
    for(int i = 0; i < DEPTH; i++) begin
      mem[i] <= 4'd0;
    end
  end else if(write_en && (!full || read_en)) begin
    write_ptr <= write_ptr + 3'd1;
    mem[true_write_ptr] <= write_data;
  end
end
```

空滿判斷:

當寫指標追上讀指標時代表滿了，將讀寫指標都多一個 bit 代表是否有多繞一圈。因此在判斷 full 時要先比較是否多繞一圈，在判斷後面的 pointer 是否相等。

```
// Full and Empty Flags
assign full = (msb_write_ptr != msb_read_ptr) && (true_write_ptr == true_read_ptr);
assign empty = (write_ptr == read_ptr);
```

2. Simulation Results

採用打隨機 pattern 的方式進行模擬驗證，主要可以分成 read、write 和 monitor 三個 task，以及一個 systemverilog 內建的 queue 用來當作 golden model。在模擬開始時會隨機打 read write 的兩個 task，write 會將資料寫進 queue 內，read 會將資料從 queue 裡面 pop 出來必且透過

monitor 這個 task 來做比對，以及確認是否有 underflow 或 overflow。

Read Operation:

```
task automatic read_operation();
  forever begin
    repeat($urandom_range(1,5)) @(posedge clk);
    read_en <= 1;
    @(posedge clk);
    read_en <= 0;
  end
endtask
```

Write Operation:

```
task automatic write_operation();
  forever begin
    repeat($urandom_range(1,5)) @(posedge clk);
      write_en <= 1;
      write_data <= $urandom_range(0, 15); // unsigned 4bit data, range from  0  to  15
      @(posedge clk);
      write_en <= 0;

      //Push data to queue
      if(write_en && (!full || read_en)) begin
        queue_model.push_back(write_data);
      end

  end
endtask
```

Monitor:

```
task automatic monitor();
  forever begin
    @(posedge clk);
    if(read_en && (!empty || write_en)) begin
      if(queue_model.size() == 0) begin
        $display("Underflow detected");
        $fatal;
      end else begin
        logic [3:0] expected_data = queue_model.pop_front();
        if(expected_data != read_data) begin
          $display("Data Mismatch: Got %0d, Expected %0d", read_data, expected_data);
          $fatal;
        end else begin
          $display("Data Matched: %0d", read_data);
        end
      end
    end else if(queue_model.size() == 5) begin
        $display("Overflow detected");
        $fatal;
    end
  end
endtask
```

3. Verification Techniques and Properties

i. Jasper Scoreboard 3

透過 bind 語法將我們設計的 FIFO 接上 jasper sb3

因為支援同時讀寫要特別在 vld in out 做邏輯操作

在滿了但同時讀寫時可以寫進 sb

在空了但同時讀寫時可以讀出 sb

```
bind FIFO my_scoreboard u_sb (
    .clk(clk),
    .rst_b(rstN),
    .vld_in(write_en && (!full || read_en)),
    .data_in(write_data),
    .vld_out(read_en && (!empty || write_en)),
    .data_out(read_data)
);
```

ii. SVA

主要驗 no overflow 和 data integrity。

a. No overflow

我們自己寫的 scoreboard 內部有一個 counter 會去計算目前 sb mem 有幾筆資料，在 counter 等於我們設定的最大深度(4)時，如果只有寫但沒有讀的話就是 overflow 了

```
// Overflow Check (Assertion: Overflow when FIFO is full and write occurs without read)
no_overflow : assert property (
    @(posedge clk)
    disable iff (!rst_b)
    !(counter == 4'd4 && vld_in && !vld_out)); // FIFO overflow check
```

b. Data integrity

當讀出資料時會先將讀資料 delay 1T，同樣 read_ptr 也會 delay 1T 後再去找 mem 內的值，再去和 sb mem 內的資料比對。會這樣做的原因是如果 empty 同時讀寫的話資料會下 1T 才寫進去 mem 內，當前 cycle 會沒法讀到，因此需要延後 1T 做比較。

```
logic [3:0] rd_ptr_1t;
// Capturing output data and valid signal
always_ff @(posedge clk or negedge rst_b) begin
  if (!rst_b) begin
    data_out_1t <= 4'd0;
    rd_ptr_1t <= 3'd0;
  end else begin
    data_out_1t <= data_out;
    rd_ptr_1t <= rd_ptr;
  end
end

// check
assign data_out_golden = sb_mem[rd_ptr_1t];
```

```
// Data Integrity Check (Assertion: Written data should match read data)
data_integrity : assert property (
    @(posedge clk)
    disable iff (!rst_b)
    (vld_out) |=> (data_out_golden == data_out_1t)); // Ensuring data integrity
```

4. Bugs Injection and Re-Verification\

插入了兩個 Bug，

第一個是空滿時不能同時讀寫

第二個是 full 邏輯判斷錯誤。

i. 空滿時不能同時讀寫

```
// Read Operation
always_ff@(posedge clk or negedge rstN) begin
  if(!rstN) begin
    read_ptr <= 3'd0;
  end else if(read_en && !empty) begin
    read_ptr <= read_ptr + 3'd1;
  end
end

//do not support read and write at the same time when fifo is empty or full
assign read_data = mem[true_read_ptr];

// Write Operation
always_ff@(posedge clk or negedge rstN) begin
  if(!rstN) begin
    write_ptr <= 3'd0;
    for(int i = 0; i < DEPTH; i++) begin
      mem[i] <= 4'd0;
    end
  end else if(write_en && !full) begin
    write_ptr <= write_ptr + 3'd1;
    mem[true_write_ptr] <= write_data;
  end
end
```

ii. full 邏輯判斷錯誤

```
// Full and Empty Flags
assign full = (msb_write_ptr != msb_read_ptr)
            && ((true_write_ptr + 'd1) == true_read_ptr);
assign empty = (write_ptr == read_ptr);
```