MANISH PUNDIR
2022PVL0063
2022pvl0063@iitjammu.ac.in

23th october 2022

Minor Project 1

Asynchronous FIFO

1 Problem Statement

To Write a HDL code for Asynchronous FIFO.

1.1 Background Information

The Memory of FIFO have to be same as birthdate of Student.
The Student having Odd Roll Number have to encode read and write pointers in binary code while the student having even Roll Number have to encode read and write pointers in Gray code

2 Solution

2.1 What is Asynchronous FIFO ?

Asynchronous FIFOs are used as buffers between two asynchronous clock domains to exchange data safely. Data is written into the FIFO from one clock domain and it is read from another clock domain. This requires a memory architecture wherein two ports of memory are available- one is for input (or write or push) operation and another is for output (or read or pop) operation. Generally FIFOs are used where write operation is faster than read operation. However, even with the different speed and access types the average rate of data transfer remains constant. FIFO pointers keep track of number of FIFO memory locations read and written and corresponding control logic circuit prevents FIFO from either under flowing or overflowing. FIFO architectures inherently have a challenge of synchronizing itself with the pointer logic of other clock domain and control the read and write operation of FIFO memory locations safely. A detailed and careful analysis of synchronizer circuit along with pointer logic is required to understand the synchronization of two FIFO pointer logic circuits which is responsible for accessing the FIFO read and write ports independently controlled by different clocks.

2.2 Working of an Asynchronous FIFO

2.2.1 Writing of Data

Writing part came into play at the positive edge of ”write clock” and at ”write enable =1 ”. The given data queue is written to FIFO memory at each positive edge of write clock.

2.2.2 Temporary storage of data in Memory Buffer

The FIFO memory is somewhat similar to cache memory. It can store the queued input data(of any defined width) in memory addresses one by one. A FIFO queue serves as a buffer that can store values that have been generated by the producer but not yet used by the consumer, thus allowing the producer to temporarily run faster than the consumer and the consumer to then ”catch up” when it’s ready, without losing any values. As long as there is space left in the queue's buffer, the producer can keep working. If the buffer fills up, the producer must stop work and wait for the consumer to process one or more values before continuing.The queue is ”full” if all the addresses of the buffer elements are occupied, and ”empty” if none of them are occupied.

2.2.3 Reading of Data

Reading part came into play at the positive edge of ”read clock” and at ”read enable =1 ”. The stored data queue in FIFO memory is readed one by one t each positive edge of read clock.

2.2.4 Synchronisation

• Synchronization of write pointer : The synchronizer uses Grey code to prevent meta stability between the two clock domains. The pointer on the write side is encoded from binary to Grey code and that goes through a full synchronizer, and then translated again into binary on the read side. The synchronized write pointer is than compared to the read pointer to generate the empty signal used for flow controlling the FIFO extract side. As the comparison between the changed write pointer and the read pointer is inevitably later than the actual load of data to the memory array, the FIFO depth should be able to absorb further loads happening during that delay time.
• Synchronization of read pointer : The read pointer synchronization uses a Grey code encoding to overcome the uncertainties of clock domain crossing and prevent unexpected read pointer values at the write clock domain. The read pointer is encoded into Grey code, goes through a full synchronizer and then decoded back into binary, to be compared with the write pointer to generate the full indication. the full indication is used for flow control of the loaded data. The comparison is late, compared to the actual read operation, so more entries may be extracted during that time. The FIFO depth needs to accommodate this delay in order to prevent unnecessary bubbles on the read side.

2.2.5 Reset

The reset signal is assumed to be capable of resetting both clock domains. When using synchronous reset, both clocks must be toggling when reset is de-asserted.

2.3 Tools Used

• HDL : Verilog
• Software : Xilinx Vivado

2.4 Inputs and Outputs

2.4.1 Inputs

write clock : Write clock for edging the write module.
Read clock : Read clock for edging the read module.
Reset : for resetting all the registers to Zero value.
Write Enable : for enabling the write module.
Read Enable : for enabling the reading module.
Input data Register : for writing the data in FIFO memory. The size of input register is [FIFO WIDTH - 1 : 0].

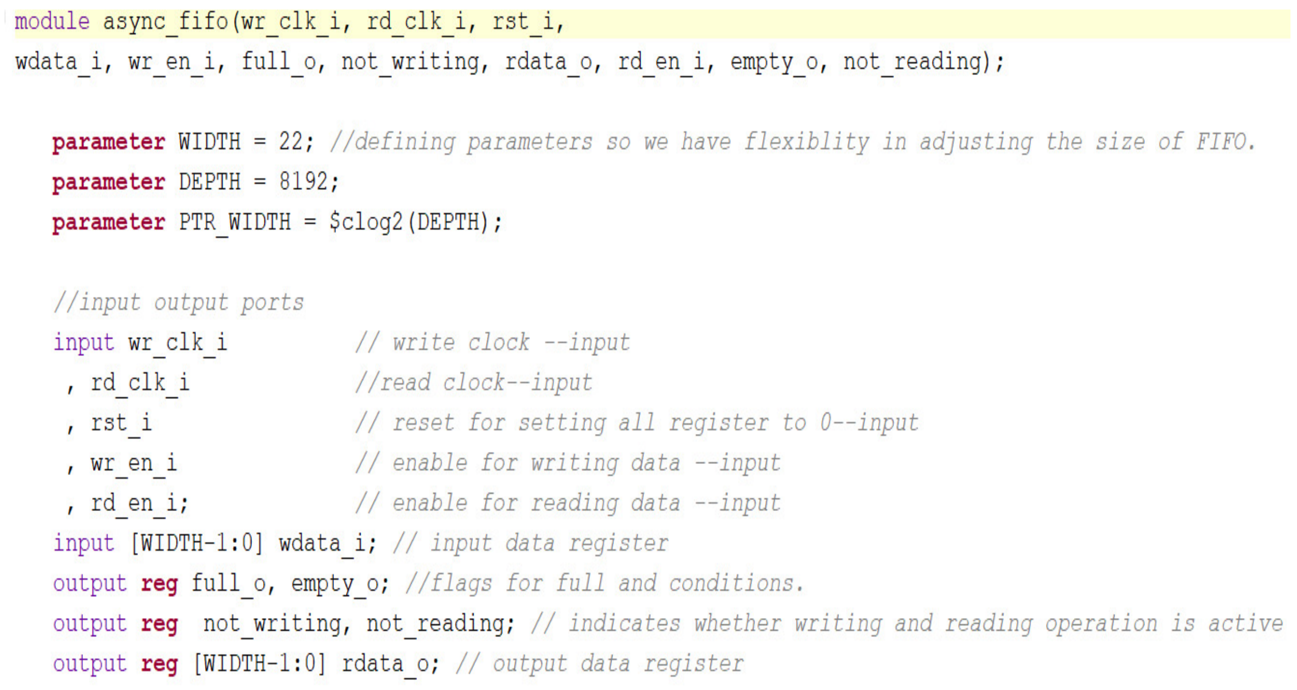
2.4.2 Outputs

FIFO Full Register(FUll Flag) : It should be high when fifo filled completely.
FIFO Empty Register(Empty Flag) : It should be high when fifo emptied completely.
Output Data Register :for Reading the data from FIFO memory. The size of output register is same as Input data Register i.e. [FIFO WIDTH - 1 : 0].

2.4.3 Internal Signals

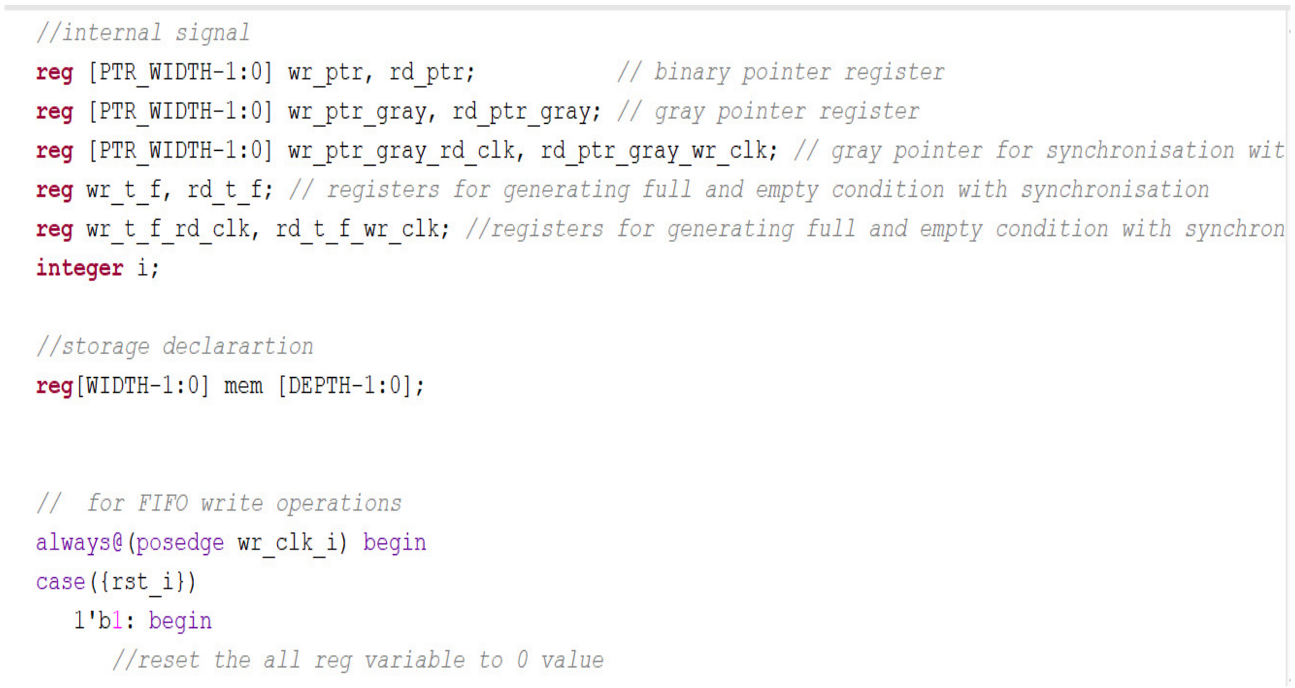
Not Writing: it should be high when writing module isn't active,
Not reading : It should be high when reading module isn't active.
Write Pointer : address of memory where data have to be written is stored in write pointer.
Read Pointer : address of memory from where data have to be read is stored in read pointer.
Read Pointer Gray : It is gray encoded write pointer.
Read pointer gray : it is gray encoded read pointer.
They Both are used in Synchronisation process. The Write Pointer is synchronised with read clock and the Read pointer is Synchronised with Write clock.

3 Verilog Code



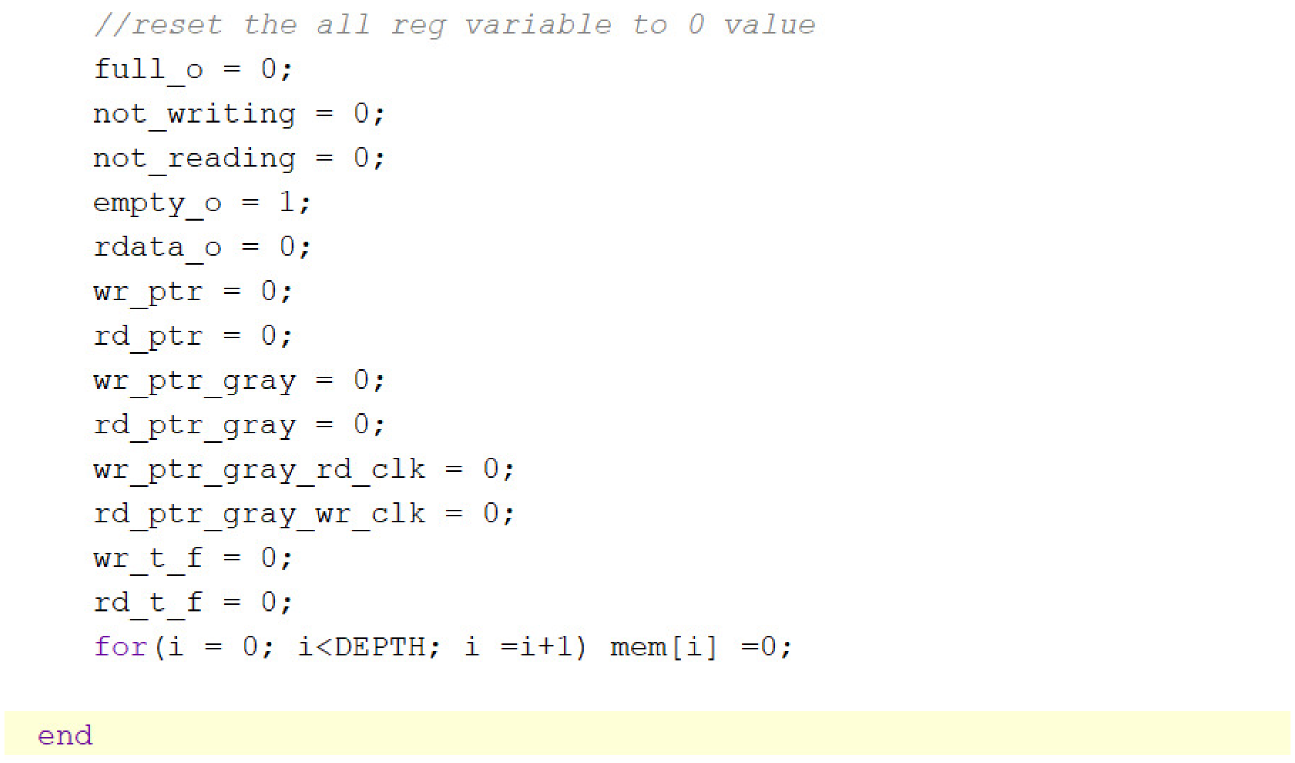
```
module async_fifo(wr_clk_i, rd_clk_i, rst_i,
wdata_i, wr_en_i, full_o, not_writing, rdata_o, rd_en_i, empty_o, not_reading);

   parameter WIDTH = 22; //defining parameters so we have flexiblity in adjusting the size of FIFO.
   parameter DEPTH = 8192;
   parameter PTR_WIDTH = $clog2(DEPTH);

   //input output ports
   input wr_clk_i          // write clock --input
    , rd_clk_i             //read clock--input
    , rst_i                // reset for setting all register to 0--input
    , wr_en_i              // enable for writing data --input
    , rd_en_i;             // enable for reading data --input
   input [WIDTH-1:0] wdata_i; // input data register
   output reg full_o, empty_o; //flags for full and conditions.
   output reg  not_writing, not_reading; // indicates whether writing and reading operation is active
   output reg [WIDTH-1:0] rdata_o; // output data register
```

Figure 1: code1



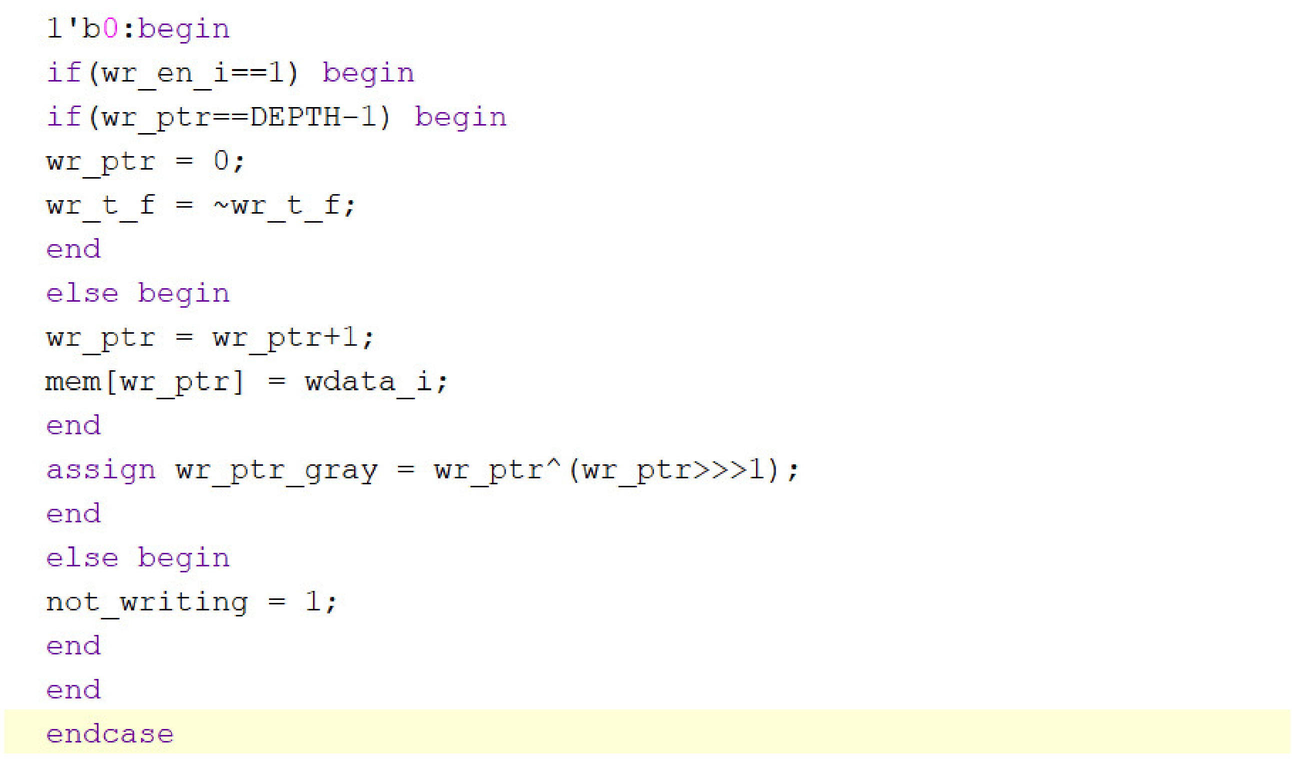
```
//internal signal
reg [PTR_WIDTH-1:0] wr_ptr, rd_ptr;           // binary pointer register
reg [PTR_WIDTH-1:0] wr_ptr_gray, rd_ptr_gray; // gray pointer register
reg [PTR_WIDTH-1:0] wr_ptr_gray_rd_clk, rd_ptr_gray_wr_clk; // gray pointer for synchronisation wit
reg wr_t_f, rd_t_f; // registers for generating full and empty condition with synchronisation
reg wr_t_f_rd_clk, rd_t_f_wr_clk; //registers for generating full and empty condition with synchron
integer i;

//storage declarartion
reg[WIDTH-1:0] mem [DEPTH-1:0];


//  for FIFO write operations
always@(posedge wr_clk_i) begin
case({rst_i})
   1'b1: begin
      //reset the all reg variable to 0 value
```

Figure 2: code2



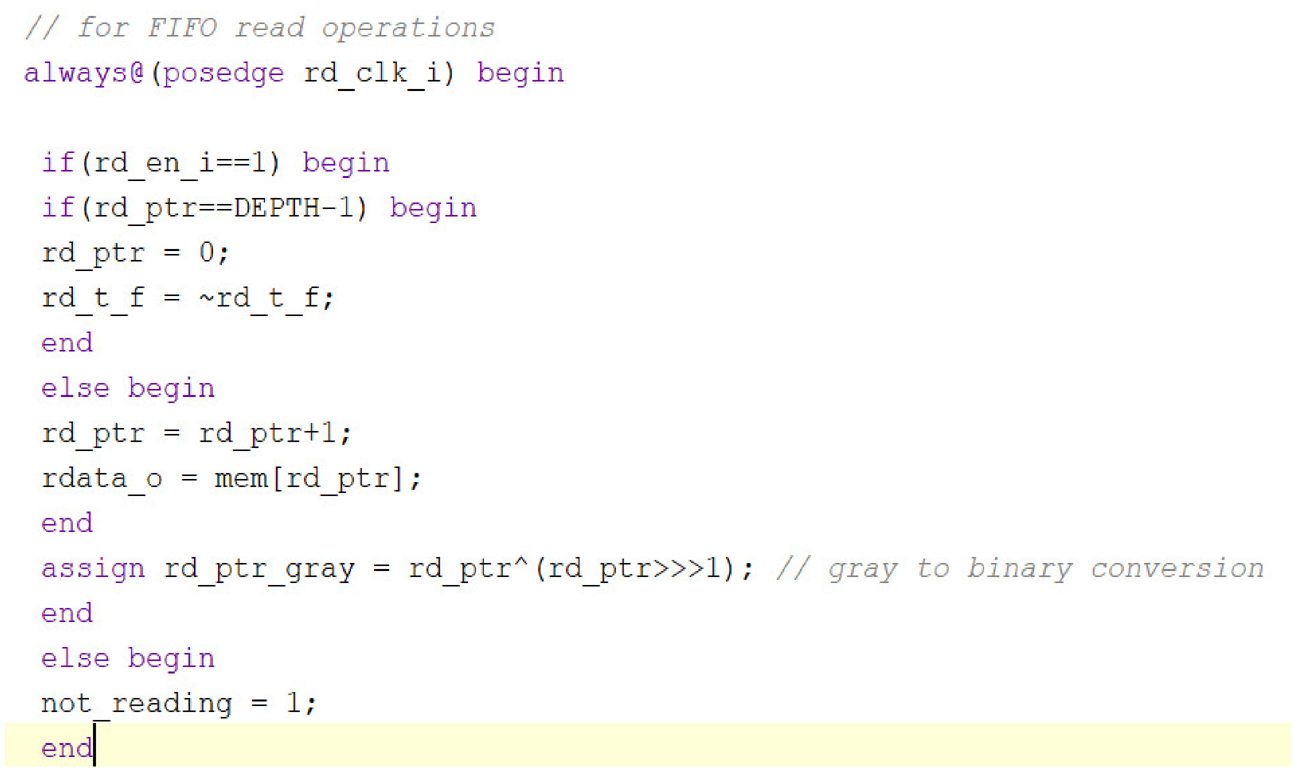
```
   //reset the all reg variable to 0 value
   full_o = 0;
   not_writing = 0;
   not_reading = 0;
   empty_o = 1;
   rdata_o = 0;
   wr_ptr = 0;
   rd_ptr = 0;
   wr_ptr_gray = 0;
   rd_ptr_gray = 0;
   wr_ptr_gray_rd_clk = 0;
   rd_ptr_gray_wr_clk = 0;
   wr_t_f = 0;
   rd_t_f = 0;
   for(i = 0; i<DEPTH; i =i+1) mem[i] =0;

end
```

Figure 3: code3



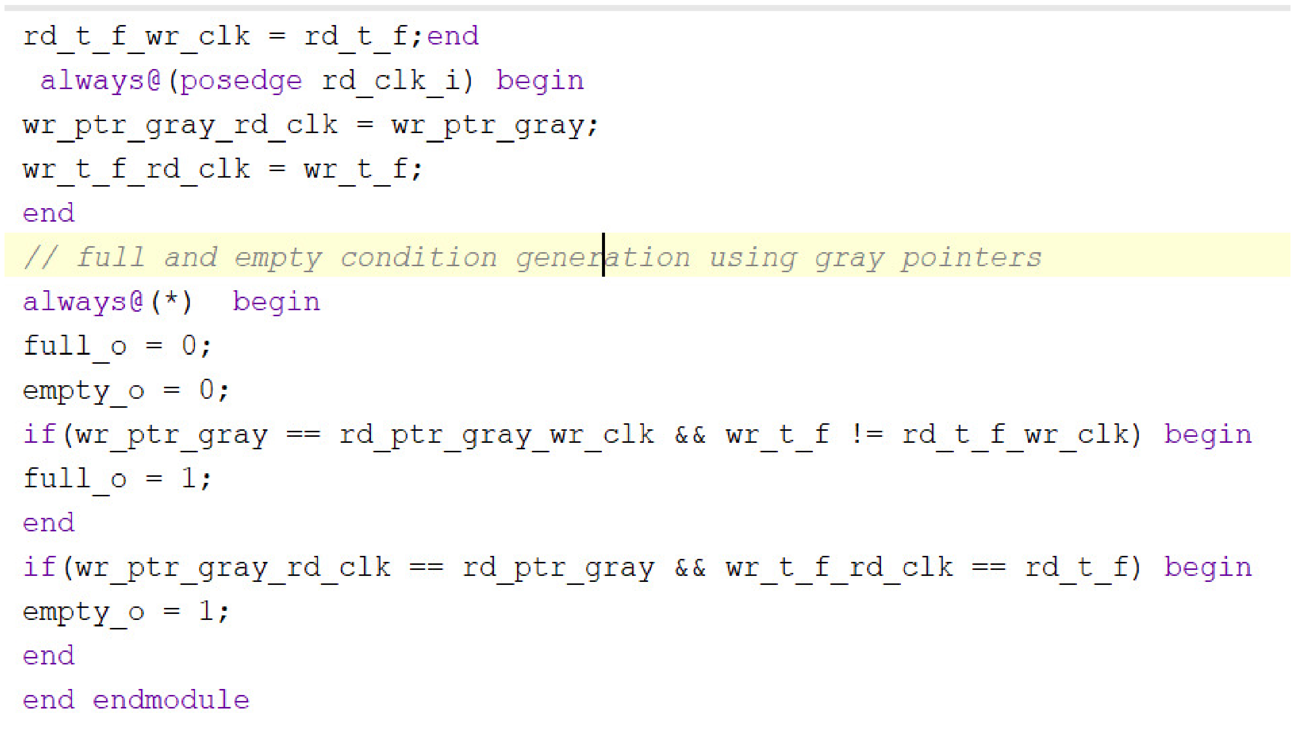
```
1'b0:begin
if(wr_en_i==1) begin
if(wr_ptr==DEPTH-1) begin
wr_ptr = 0;
wr_t_f = ~wr_t_f;
end
else begin
wr_ptr = wr_ptr+1;
mem[wr_ptr] = wdata_i;
end
assign wr_ptr_gray = wr_ptr^(wr_ptr>>>1);
end
else begin
not_writing = 1;
end
end
endcase
```

Figure 4: code4



```
// for FIFO read operations
always@(posedge rd_clk_i) begin

 if(rd_en_i==1) begin
 if(rd_ptr==DEPTH-1) begin
 rd_ptr = 0;
 rd_t_f = ~rd_t_f;
 end
 else begin
 rd_ptr = rd_ptr+1;
 rdata_o = mem[rd_ptr];
 end
 assign rd_ptr_gray = rd_ptr^(rd_ptr>>>1); // gray to binary conversion
 end
 else begin
 not_reading = 1;
 end
```

Figure 5: code5



```
rd_t_f_wr_clk = rd_t_f;end
 always@(posedge rd_clk_i) begin
wr_ptr_gray_rd_clk = wr_ptr_gray;
wr_t_f_rd_clk = wr_t_f;
end
// full and empty condition generation using gray pointers
always@(*)  begin
full_o = 0;
empty_o = 0;
if(wr_ptr_gray == rd_ptr_gray_wr_clk && wr_t_f != rd_t_f_wr_clk) begin
full_o = 1;
end
if(wr_ptr_gray_rd_clk == rd_ptr_gray && wr_t_f_rd_clk == rd_t_f) begin
empty_o = 1;
end
end endmodule
```

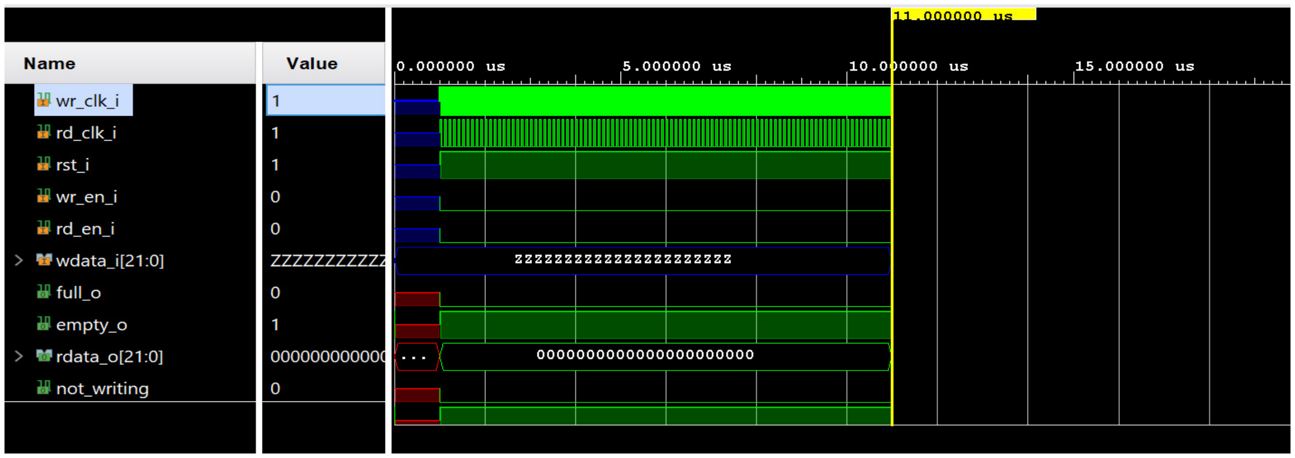
Figure 6: code6

4 Test Cases

After Lauching the simulation we firstly reset the fifo then only all the opoerations will be performed in right sequence.

4.1 Reset Operation

• write clock pedge = 1 and negedge = 0 • read clock pedge = 1 and negedge = 0
• reset =1 • write enable = 0 • Read enable = 0 • Data input = XXXXXXX



Name	Value
wr_clk_i	1
rd_clk_i	1
rst_i	1
wr_en_i	0
rd_en_i	0
wdata_i[21:0]	ZZZZZZZZZZZ
full_o	0
empty_o	1
rdata_o[21:0]	000000000000
not_writing	0

Figure 7: Reset$_1$

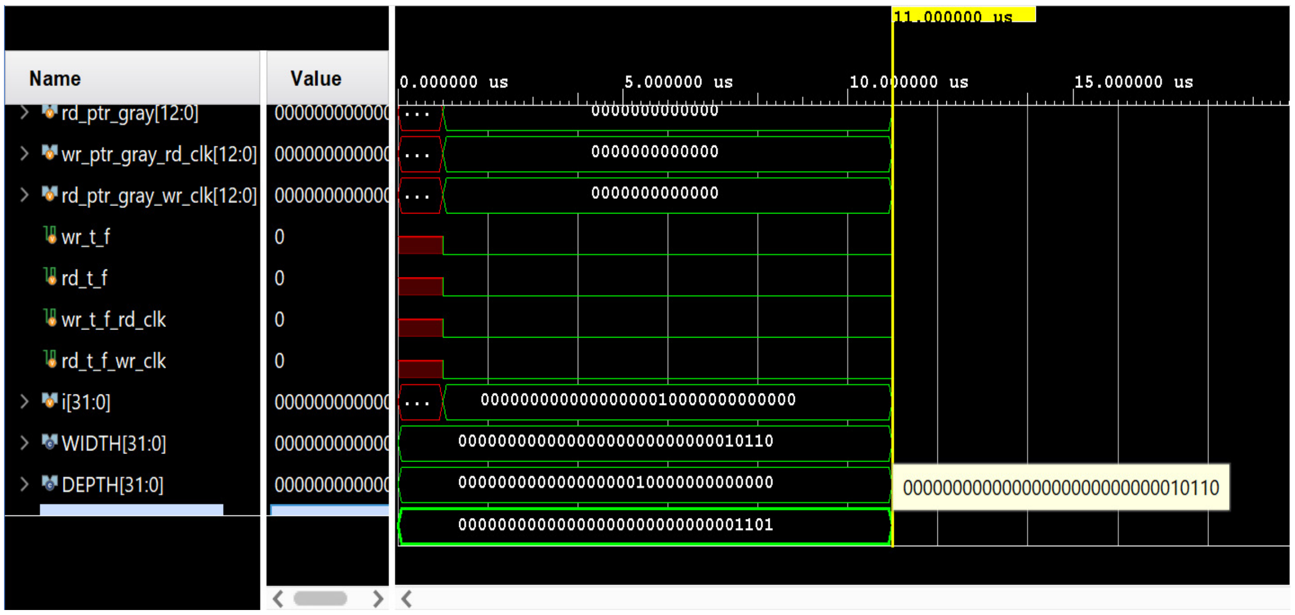


Figure 8: Reset$_2$

4.2 Write Operation

• write clock pedge = 1 and negedge = 0 • read clock pedge = 1 and negedge = 0
• reset = 0 • write enable = 1
• Read enable = 0
• first Data input = 1
• Second Data input = 2
• Third Data input = 3

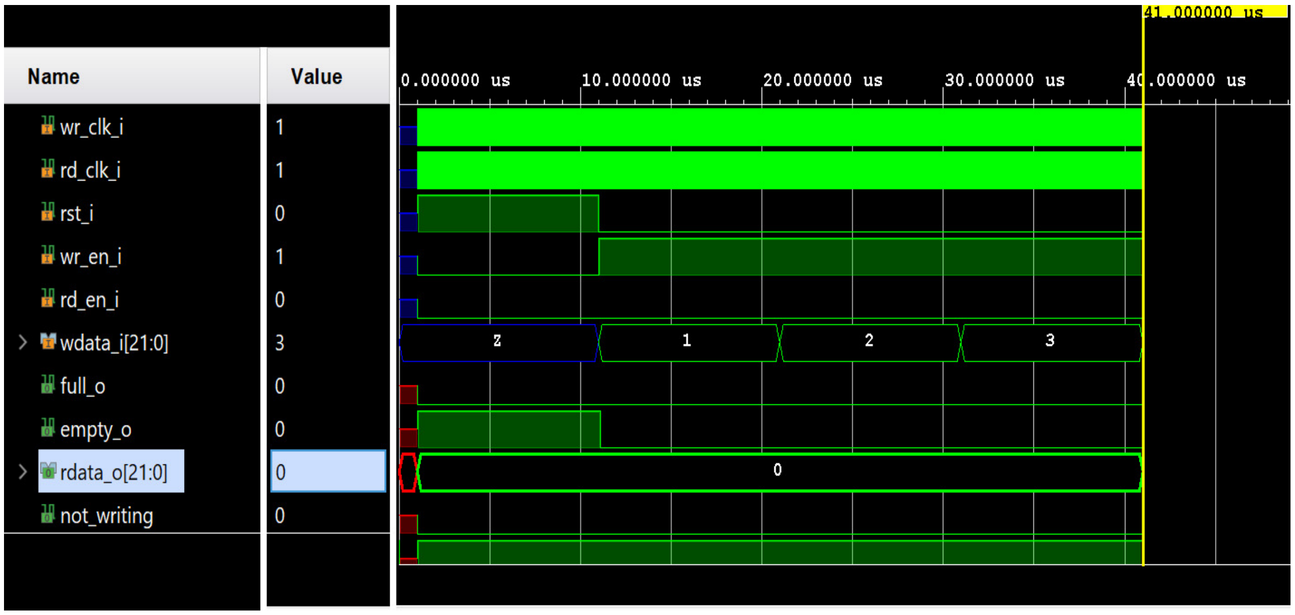
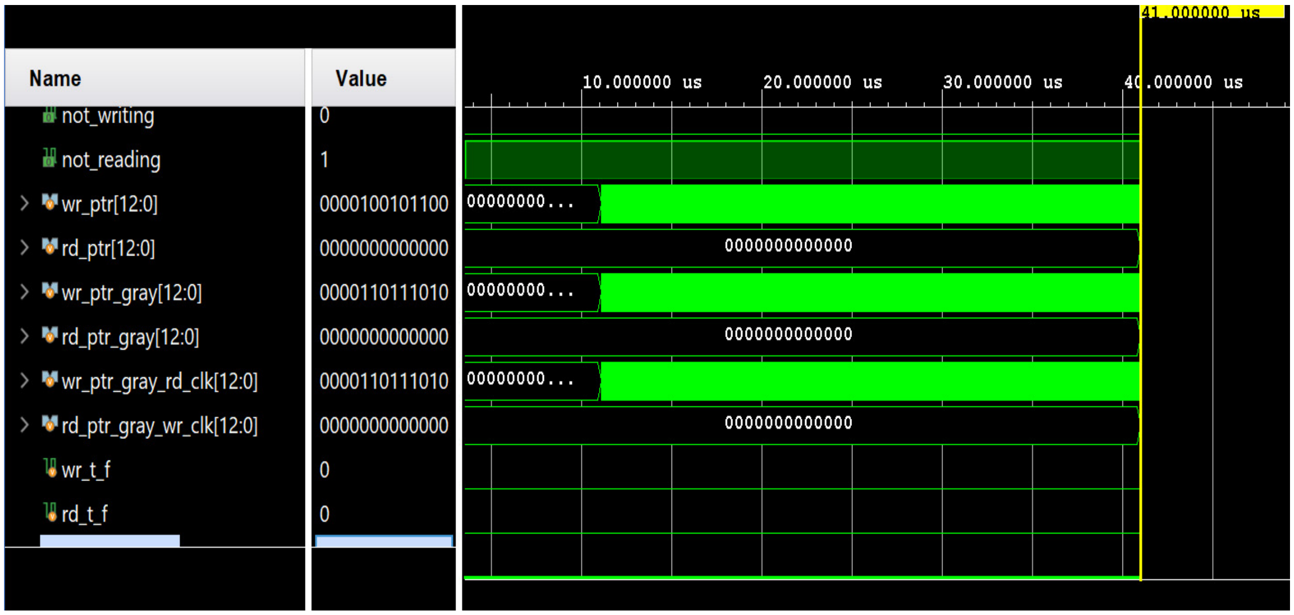


Figure 9: Write$_1$

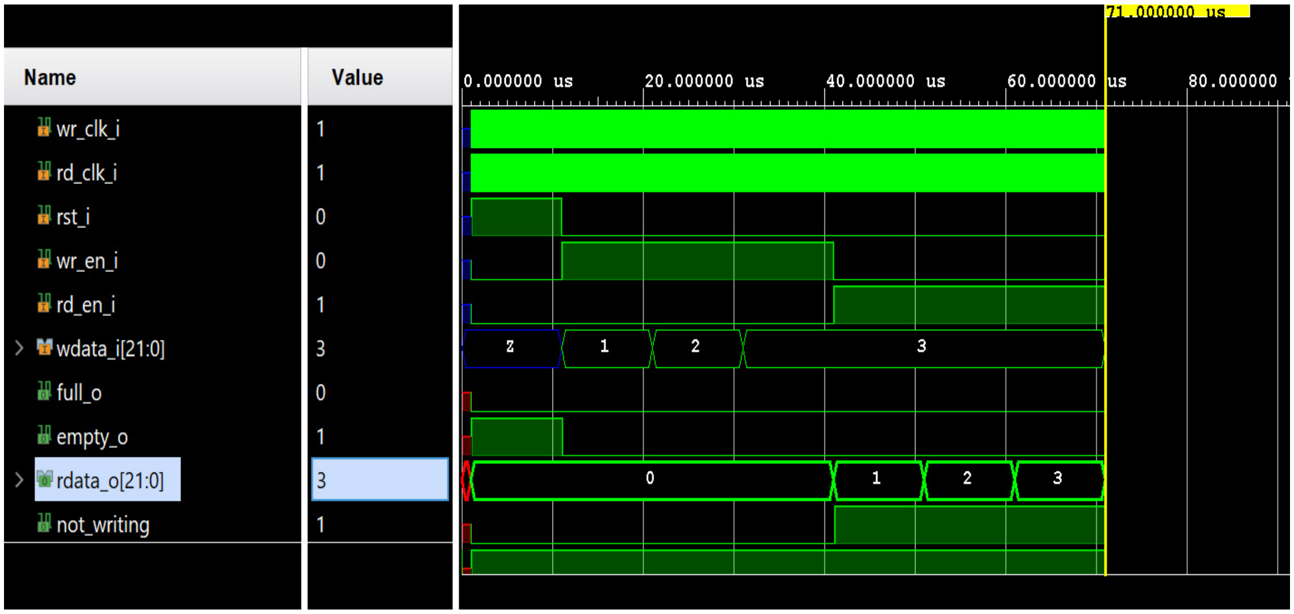


Name	Value
not_writing	0
not_reading	1
wr_ptr[12:0]	0000100101100
rd_ptr[12:0]	0000000000000
wr_ptr_gray[12:0]	0000110111010
rd_ptr_gray[12:0]	0000000000000
wr_ptr_gray_rd_clk[12:0]	0000110111010
rd_ptr_gray_wr_clk[12:0]	0000000000000
wr_t_f	0
rd_t_f	0

Figure 10: Write$_2$

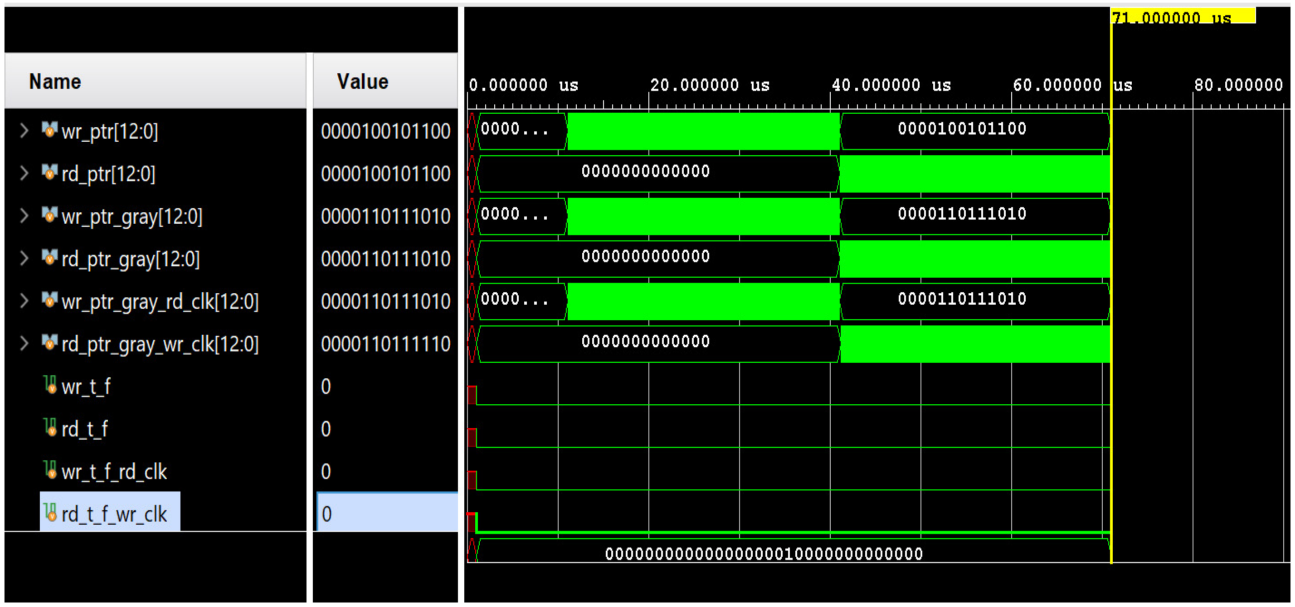
4.3 Read Operation

• write clock pedge = 1 and negedge = 0 • read clock pedge = 1 and negedge = 0
• reset = 0 • write enable = 0
• Read enable = 1
• first Data read = 1
• Second Data read = 2
• Third Data read = 3



Name	Value
wr_clk_i	1
rd_clk_i	1
rst_i	0
wr_en_i	0
rd_en_i	1
wdata_i[21:0]	3
full_o	0
empty_o	1
rdata_o[21:0]	3
not_writing	1

Figure 11: Read$_1$



Name	Value
wr_ptr[12:0]	0000100101100
rd_ptr[12:0]	0000100101100
wr_ptr_gray[12:0]	0000110111010
rd_ptr_gray[12:0]	0000110111010
wr_ptr_gray_rd_clk[12:0]	0000110111010
rd_ptr_gray_wr_clk[12:0]	0000110111110
wr_t_f	0
rd_t_f	0
wr_t_f_rd_clk	0
rd_t_f_wr_clk	0

Figure 12: Read$_2$

5 Schematic Diagram

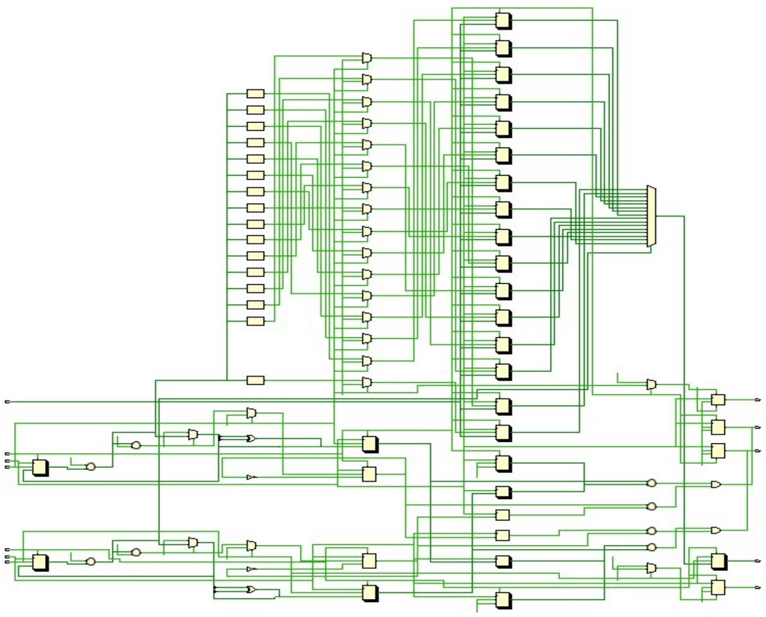


Figure 13: Schematic Diagram