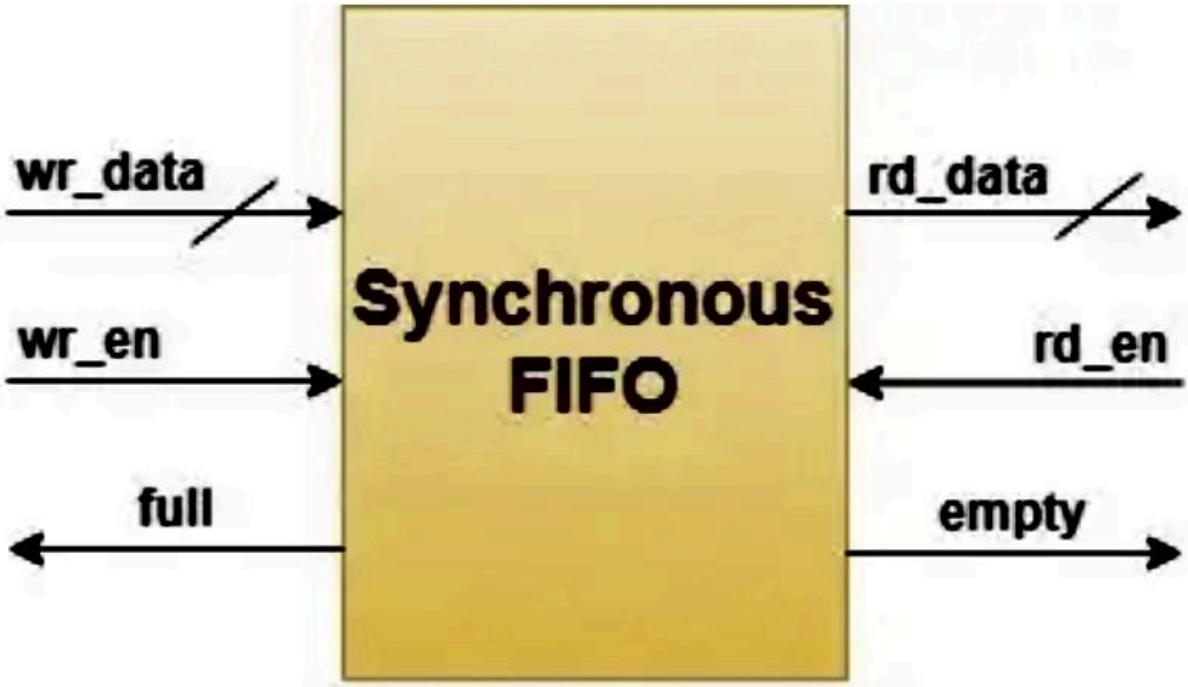


Basic FIFO block diagram

Adityagt_verilog

FIFO: FIRST IN FIRST OUT

- FIFO is a design that returns the data in FIRST IN FIRST OUT order.

:- THERE ARE TWO TYPES OF FIFO'S:
1.) SYNCHRONOUS FIFO (Same clock freq)
2.) ASYNCHRONOUS FIFO (different clock)

⊙ SYNCHRONOUS FIFO :-

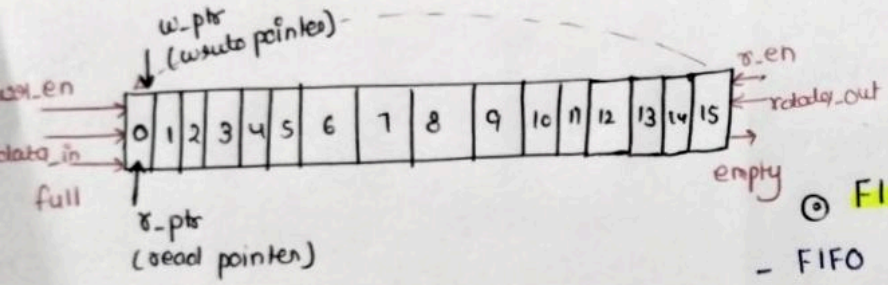
- IN SYNCHRONOUS FIFO, data read and write operations use the same clock Foequency. usually, they are used with high clk Foequency to support high speed System

NOTE :- FLAGS IN FIFO :-

- FIFO should have 2 ouput signals, full and empty to indicate its status. its work like flag.

Full = 1
- indicates that FIFO is full.
- don't do furthes write. Since I am full.

Empty = 1
- indicates that FIFO is empty.
- don't do furthes write Since I am full.



- PROCESSES : 2 Processes
1.) write to the FIFO.
2.) Read From the FIFO.

⊙ FIFO write operation :-

- FIFO Can store / write the wr_data at every posedge of the clk based on wr_en signal till it is full.
- The write_ptr incremented on every data write in FIFO MEMORY.

SYNCHRONOUS FIFO OPERATION:

SIGNALS (means PORT'S) :-

wr_en :- write enable
wr_data_i :- write data input
full :- FIFO is full
empty :- FIFO is empty
Toggle the wr_toggle_f or rd_toggle_f.
rd_en : read enable
rdata_i :- read data input
w_ptr :- write pointer
r_ptr :- read pointer

⊙ READ OPERATION:

- The data Can be taken out or read from the FIFO at every posedge of the clk based on the rd_en signal till is empty.
- The read_ptr gets incremented on every data read from FIFO memory.

⊙ Why we need FIFO?

→ A FIFO is an array of memory commonly used in hardware to transfer, transfer data between two circuits with different clocks.

- chip operation is all about data transfer.

ex: talking over phone is a data transfer. sending message is data transfer.

- FIFO uses a dual PORT memory and there will be two pointers to point read and write address.

⊙ FIFO Full and FIFO empty flags are of great Concern as no data should be written in full Condition. and

- no data should be read in empty Condition as it can lead to loss of data or generation of non relevant data.

• The full and empty Condition of FIFO controlled using "Binary or gray pointer"

• while The gray pointer are used for generating full and empty Condition for Asynchronous FIFO.

⊙ POINTERS TO CONTROL OPERATIONS:

• write pointer and Read pointer:

- There are two pointer, write-ptr and read-ptr
- which helps in steering the data into and out of the memory array.

• They store the write and read address value associated with the memory array. after each sucessful data write and/or read, the corresponding pointer is incremented by one to point to the next address.

** • read pointer and write pointe should be one bit higher than read and write address.

⊙ FIFO CIRCUIT DIAGRAM :-

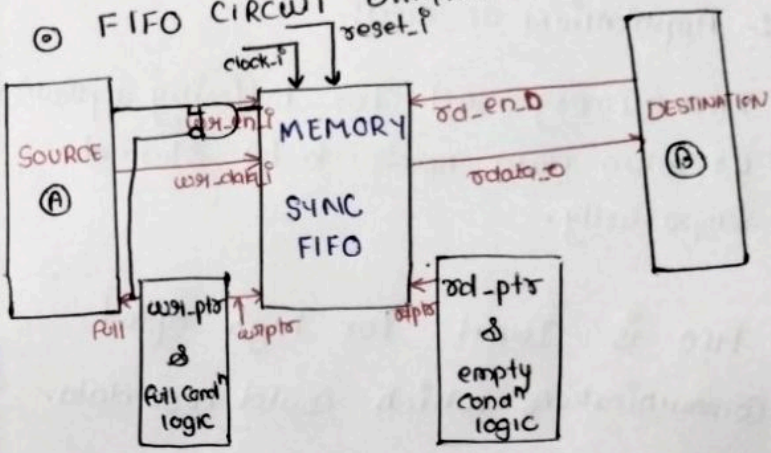


Fig: SynchRONOUS FIFO

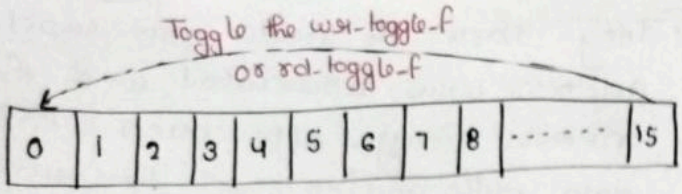
:- does FIFO required address?

Not required its has pointer (read-pointer and write-pointer.

:- for every write/read happening, track the next write/read Position using a write pointer and read Pointer.

⊙ whenever write happens, increment wr_ptr by 1.

⊙ whenever read happens, increment rd_ptr by 1.



- wr_ptr and rd_ptr are matching. toggle flags are not matching = Full
- wr_ptr and rd_ptr are matching, toggle flags are also matching = EMPTY

:- APPLICATION OF FIFO:-

- FIFO MEMORY used For buffering applications or when data needs to be stored temporarily.
- FIFO is Ideal For high Speed Communications which could lose data.
- Some example of Synchronous Communication are in-person meeting, massages.

⊙ difference Between Synchronous and Asynchronous FIFO?

Synchronous FIFO:-

-A FIFO (FIRST IN FIRST OUT) is a Buffer with Separate read and write ports for sending signals / data.

- Synchronous FIFO ~~and Asynchronous FIFO~~ both are Controlled by the clock from same domain means same clock frequency.

ASYNCHRONIOUS FIFO:-

- in case of an asynchronous FIFO Both read and write ports are from different clock frequency.
- usually Asynchronous FIFO is a Solution for clock domain crossing, If the data are moving a faster clock frequency to a slow clock frequay, Asynchronous FIFO is used.

fifo.v (~\Desktop\sure_trust assignment\Synchronous_FIFO) - GVIM

File Edit Tools Syntax Buffers Window Help

```
 1 module fifo(clk_i,rst_i,wdata_i,wr_en_i,rd_en_i,full_o,empty_o,error_o,rdata_o);
 2 parameter DEPTH=16;
 3 parameter WIDTH=4;
 4 parameter ADDR_WIDTH=$clog2(DEPTH);
 5 input clk_i,rst_i,wr_en_i,rd_en_i;
 6 input [WIDTH-1:0]wdata_i;
 7 reg [ADDR_WIDTH-1:0]wr_ptr;//these signals are internal signals ,need not to declare inside portlist
 8 reg [ADDR_WIDTH-1:0]rd_ptr;
 9 output reg empty_o,full_o,error_o;
10 output reg [WIDTH-1:0]rdata_o;
11 integer i;
12 reg wr_toggle_f;
13 reg rd_toggle_f;
14 reg [WIDTH-1:0]fifo[DEPTH-1:0];//fifo(is memory)declaration
15 always@(posedge clk_i)begin
16     if(rst_i==1)begin
17         full_o=0;
18         empty_o=1;
19         rdata_o=0;
20         error_o=0;
21         wr_ptr=0;
22         rd_ptr=0;
23         wr_toggle_f=0;
24         rd_toggle_f=0;
25         for(i=0;i<DEPTH;i=i+1)begin
26             fifo[i]=0;
27         end
28     end
29     else begin
30         if(wr_en_i==1)begin
31             if(full_o == 0)begin
32                 error_o=0;
```

fifo.v (~\Desktop\sure_trust assignment\Synchronous_FIFO) - GVIM

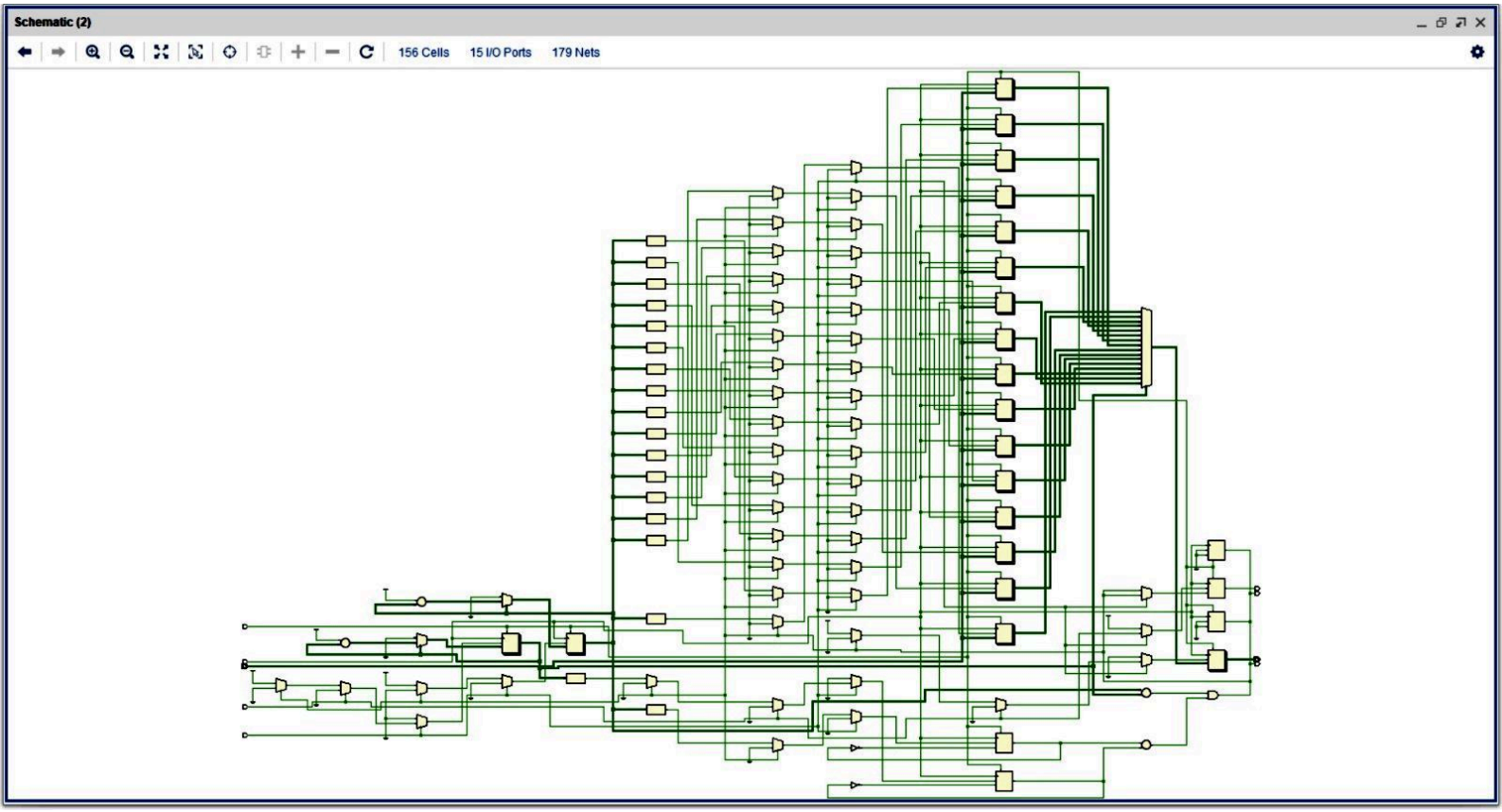
File Edit Tools Syntax Buffers Window Help

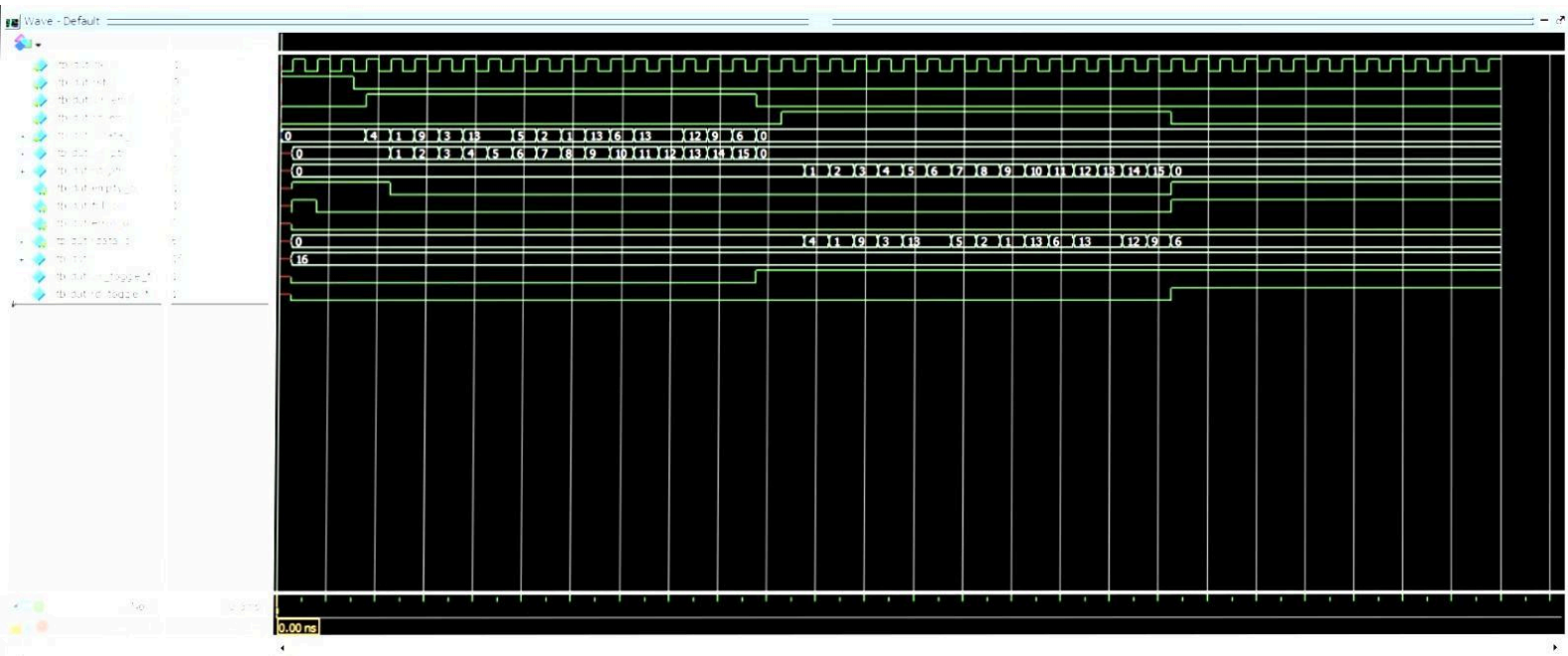
```
32                  error_o =0;
33                  fifo[wr_ptr]=wdata_i;// write data(wdata) will be assigned to the write pointer
34                  if(wr_ptr == DEPTH-1)begin
35                      wr_ptr =0;
36                      wr_toggle_f = ~wr_toggle_f;
37                  end
38              else begin
39                  wr_ptr = wr_ptr+1;
40              end
41          end
42          else begin
43              error_o=1;
44          end
45      end
46      else begin
47              if(rd_en_i==1)begin
48                  if(empty_o == 0)begin
49                      error_o =0;
50                      rdata_o=fifo[rd_ptr];
51                      if(rd_ptr == DEPTH-1)begin
52                          rd_ptr =0;
53                          rd_toggle_f = ~rd_toggle_f;
54                      end
55                  else begin
56                      rd_ptr = rd_ptr+1;
57                  end
58              end
59              else begin
60                  error_o=1;
61              end
62          end
63      end
```

fifo.v (~\Desktop\sure_trust assignment\Synchronous_FIFO) - GVIM

File Edit Tools Syntax Buffers Window Help

```
45      end
46      else begin
47              if(rd_en_i==1)begin
48                  if(empty_o == 0)begin
49                      error_o =0;
50                      rdata_o=fifo[rd_ptr];
51                      if(rd_ptr == DEPTH-1)begin
52                          rd_ptr =0;
53                          rd_toggle_f = ~rd_toggle_f;
54                      end
55                  else begin
56                      rd_ptr = rd_ptr+1;
57                  end
58              end
59              else begin
60                  error_o=1;
61              end
62           end
63      end
64 end
65 end
66 always@(*) begin
67      full_o =0;
68      empty_o =0;
69      if(wr_ptr==rd_ptr && wr_toggle_f==rd_toggle_f)begin
70          empty_o =1;
71      end
72      if(wr_ptr==rd_ptr && wr_toggle_f==rd_toggle_f)begin
73          full_o =1;
74      end
75 end
76 endmodule
```


Schematic (2)
156 Cells
15 I/O Ports
179 Nets


Wave - Default
0.00 ns

tb_fifo.v (~\Desktop\sure_trust assignment\Synchronous_FIFO) - GVIM

File Edit Tools Syntax Buffers Window Help

```
 1 `include "fifo.v"
 2 module tb;
 3 parameter DEPTH=16;
 4 parameter WIDTH=4;
 5 parameter ADDR_WIDTH=$clog2(DEPTH);
 6 reg clk_i,rst_i,wr_en_i,rd_en_i;
 7 reg [WIDTH-1:0]wdata_i;
 8 //reg [ADDR_WIDTH-1:0]wr_ptr;//these signals are internal signals ,need not to declare inside portlist
 9 //reg [ADDR_WIDTH-1:0]rd_ptr;
10 wire empty_o,full_o,error_o;
11 wire [WIDTH-1:0]rdata_o;
12 integer i;
13 //wire wr_toggle_f;
14 //wire rd_toggle_f;
15
16 fifo #(.DEPTH(DEPTH), .WIDTH(WIDTH), .ADDR_WIDTH(ADDR_WIDTH)) dut(clk_i,rst_i,wdata_i,wr_en_i,rd_en_i,full_o,empty_o,error_o,rdata_o);
17
18 initial begin
19 clk_i = 0;
20 forever #5 clk_i = ~clk_i;
21 end
22
23 initial begin
24     rst_i=1;
25     wr_en_i=0;
26     rd_en_i=0;
27     wdata_i=0;
28     #30;
29     rst_i =0;
30     for(i=0;i<=DEPTH; i =i+1)begin
31         @(posedge clk_i)
32         wdata_i =$random;
```

```
30      for(i=0;i<=DEPTH; i =i+1)begin
31          @(posedge clk_i)
32          wdata_i =$random;
33          wr_en_i=1;
34      end
35          wdata_i = 0;
36          wr_en_i =0;
37      for(i=0;i<=DEPTH; i =i+1)begin
38          @(posedge clk_i)
39          rd_en_i=1;
40      end
41          rd_en_i=0;
42  end
43  initial begin
44  #500;
45  $finish;
46  end
47  endmodule
```