Asynchronous FIFO Design

2.1 Introduction:

An Asynchronous FIFO Design refers to a FIFO Design where in the data values are written to the FIFO memory from one clock domain and the data values are read from a different clock domain, where in the two clock domains are Asynchronous to each other. Asynchronous FIFO's are widely used to safely pass the data from one clock domain to another clock domain

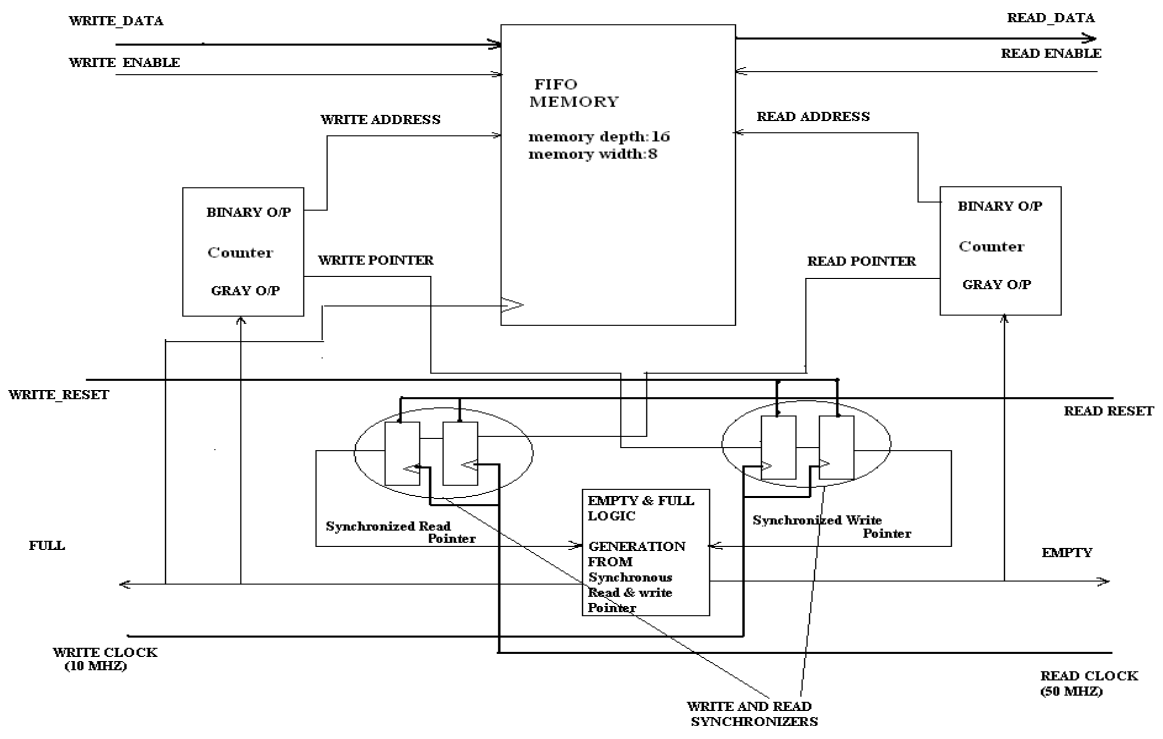


Fig 2.1 Asynchronous FIFO Design

2.2 DESCRIPTION OF FIFO DESIGNED

The above figure's refers of an Asynchronous FIFO, it will be better if each block is explained

- **FIFO MEMROY**

This is the heart of the FIFO, the depth of memory is 16 bits and width is 8 bits,
It has an the following **inputs**
Write Data (8 bit), Write Enable, Read Enable, Write Clock, Write address (4 bit), Read Address (4 bit) And an **output** i.e. Read Data (8 bit)

Data which is to be written and the address where it has to be written is supplied at the input port write data and write address. At the positive edge of the clock when Write enable is enabled so now the data is been written into the FIFO memory, now it has to be Read out, for that to happen Read enable should be Enabled and the address from which the data has to be read should be specified at the input port Read address.

This is the Memory operation in brief .now we have to control the memory in such a way that it meets the requirements of the FIFO.

➢ BINARY & GRAY COUNTER

We need to design a counter which can give Binary and Gray output's, the need for Binary counter is to address the FIFO MEMORY i.e. Write and Read address. And the need of Gray counter is for addressing Read and Write pointers.

Once the counter with binary and Gray code output is designed it is then Port mapped with Memory's Read address, write address, Read pointer, Write Pointer.

The Use Full and Empty logic for addressing the memory
Empty: the counter takes Empty signal and increments the Read address depending on this.
Full: when ever the Full signal is high the counter should not increment write address

```
If (~EMPTY)                          If (~FULL)
        Increment Read Address               Increment Write Address
        Else                                 Else
        No Increment                         no increment
```

➢ SYNCHRONIZER'S

Synchronizers are very simple in operation; they are made of 2 D Flip Flop's.
As the FIFO is operating at 2 different clock domains so there is a need to synchronize the Write and Read pointers for generating empty and full logic which in turn is used for addressing the FIFO memory.
The Figure below shows how synchronization takes place; the logic behind this is very simple.
What we are trying to do over here is , passing the Write Pointer to a D Flip Flop which is driven by the Read clock and in the same manner the Read pointer is fed to a D Flip Flop which is driven by Write Clock, so as a result of this we get Read Pointer (which is operating under Read clock) and Synchronized Write Pointer which is also operating under Read clock, and the same with Write pointer and Synchronized Read Pointer, so now we can compare them and derive a logic for Generating Empty and Full conditions, which is the most important design part of this FIFO

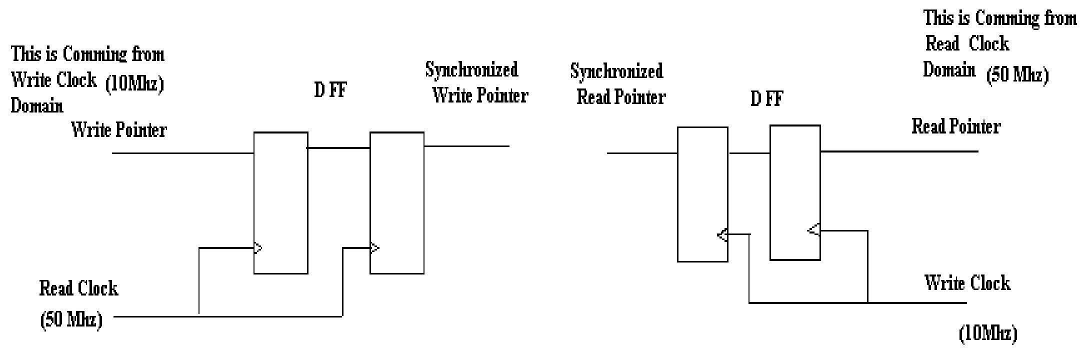

This is Comming from Write Clock (10Mhz) Domain
Write Pointer
D FF
Synchronized Write Pointer
Read Clock (50 Mhz)
Synchronized Read Pointer
D FF
This is Comming from Read Clock Domain (50 Mhz)
Read Pointer
Write Clock
(10Mhz)

Fig 2.2: Synchronizer Logic

- **EMPTY AND FULL LOGIC BLOCK**

In the above section we discussed mainly on the synchronizing part, I think the figures below are self explanatory

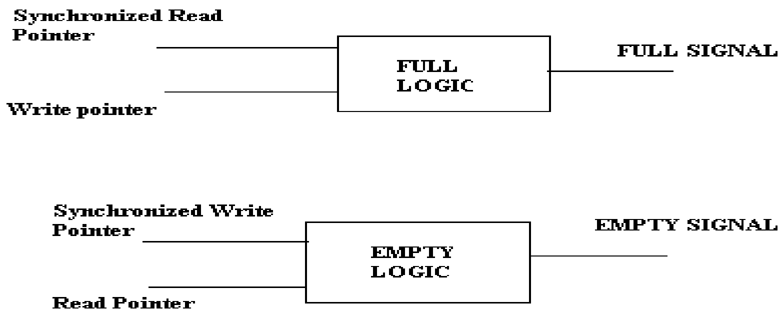


Fig: 2.3 Empty and Full Logic

If ((synchronized Write pointer = = Read pointer) &&
(Synchronized Write pointer [3:0] = = Read pointer [3:0] then
Empty=1;
If (Write pointer= = {~ synchronized Read pointer [4:3], synchronized Read pointer [2:0]) then
Full=1;

Fig: Empty & Full logic generation

2.3 RESULT ANALYSIS (OUTPUT ANALYSIS)

2.3.1 Output Waveform 1

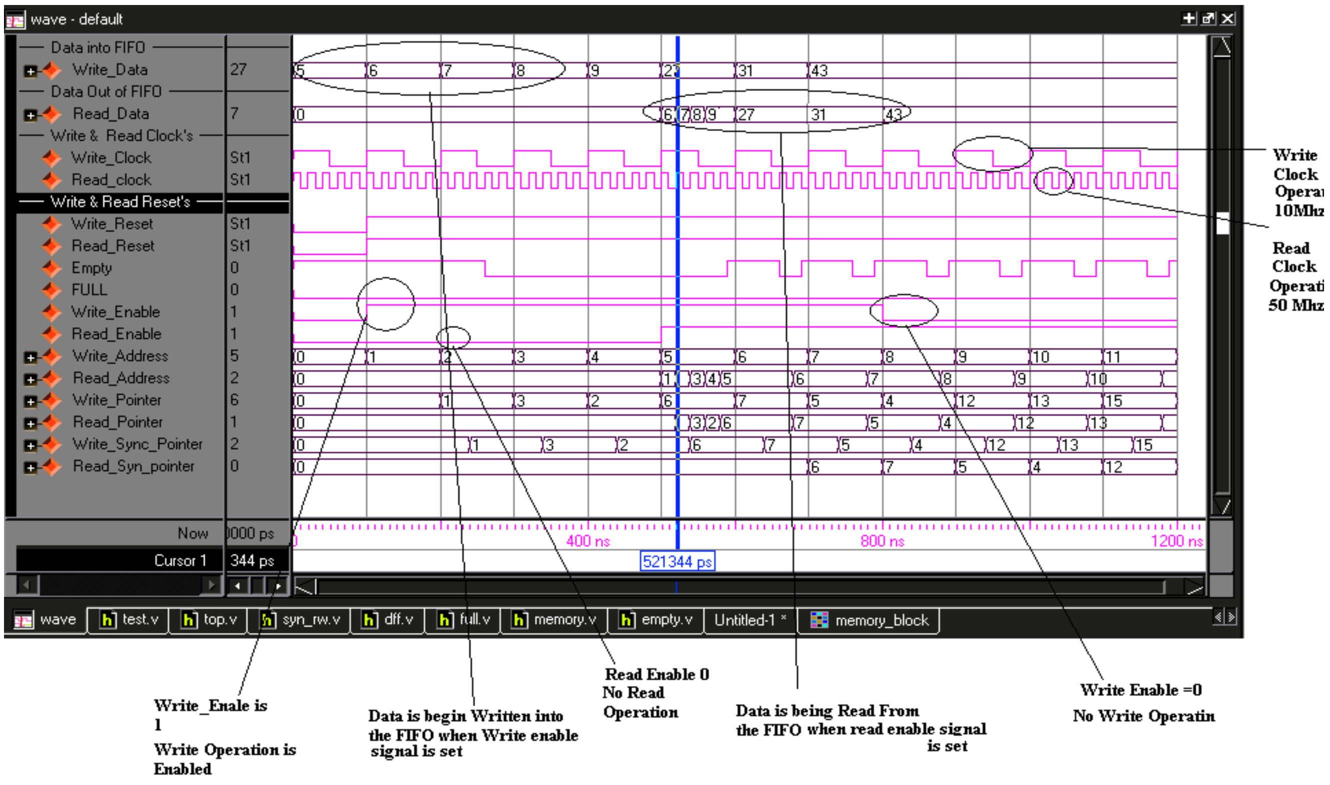
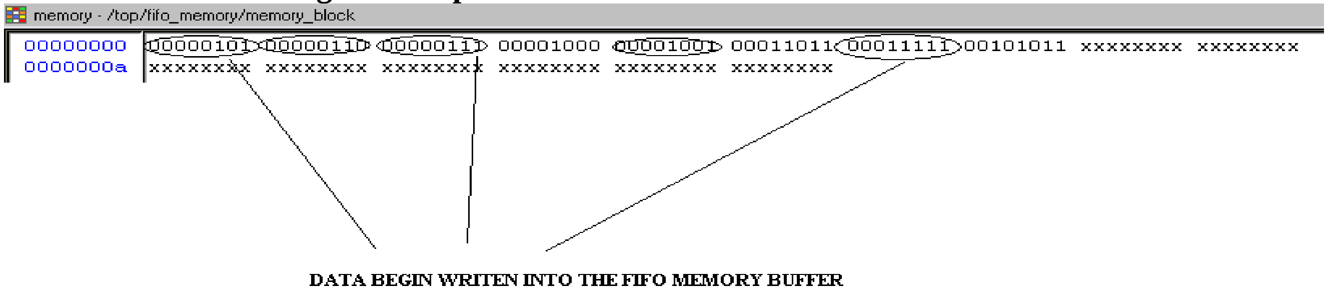


Fig 2.4: Output Waveform 1



```
memory - /top/fifo_memory/memory_block
00000000  00000101 00000110 00000111 00001000 00001001 00011011 00011111 00101011 xxxxxxxx xxxxxxxx
0000000a  xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
```

Fig 2.5 Output Waveform 2

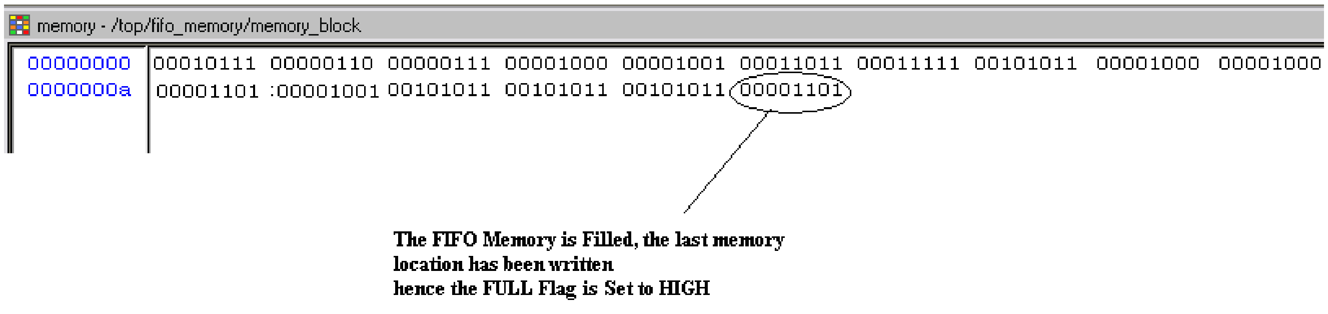
The above waveform is analyzed below,

- The Read clock is operating at 50 MHz and Write Clock is operating at 10 MHz
- System is first Reset and then Set, the operation begins
- To start with the Write_enable signal is high(i.e. Write operation is active)
- Read_enable is set to '0' (Read operation disabled)
- Input data is fed into the FIFO memory and its being written in the memory as shown in Fig
- To start with values 5,6,7,8,9,23,31,41 are written into FIFO memory as you can see in Fig 2..
- The write address keeps on incrementing because(Write_enable= = 1)
- At 500 ns Read Enable is set to '1'(enabled) and the Read address starts incrementing and data which was written into the FIFO memory comes out in the order it was written(

First In First out i.e. The Data which was written first into the FIFO memory is being Read out first i.e. 5 is readout first then 6 ,7,8,23,31,41

- We can also notice that the Read and Write Pointer's are synchronized
- The Empty and full conditions will be explained in next waveforms

2.3.2 Output Waveform 2



memory - /top/fifo_memory/memory_block

```
00000000  00010111  00000110  00000111  00001000  00001001  00011011  00011111  00101011  00001000  00001000
0000000a  00001101 :00001001  00101011  00101011  00101011  00001101
```

The FIFO Memory is Filled, the last memory
location has been written
hence the FULL Flag is Set to HIGH

Fig 2.6: **Output Waveform 3**

The above waveform is analyzed below,

- Once the FIFO memory is filled the Full flag goes high.

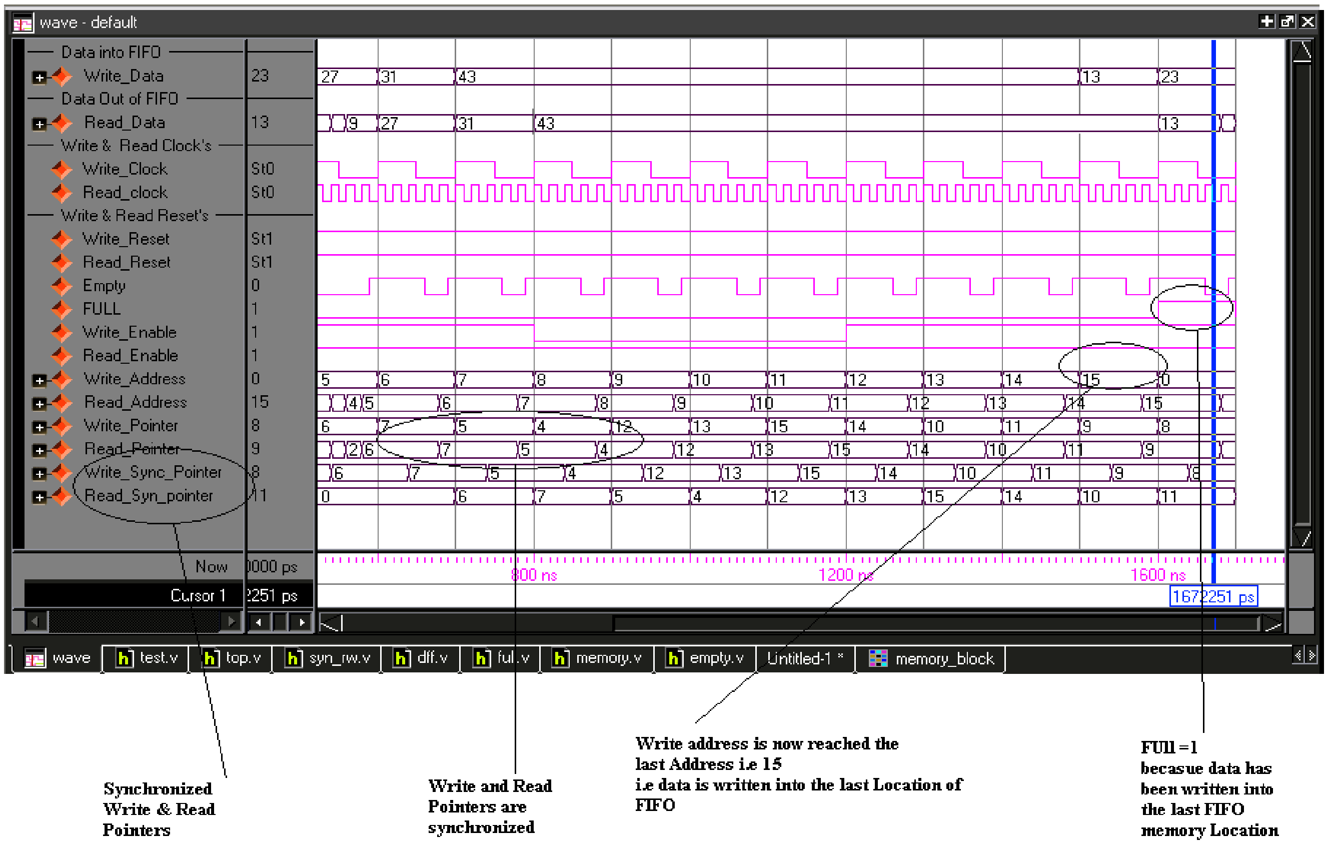


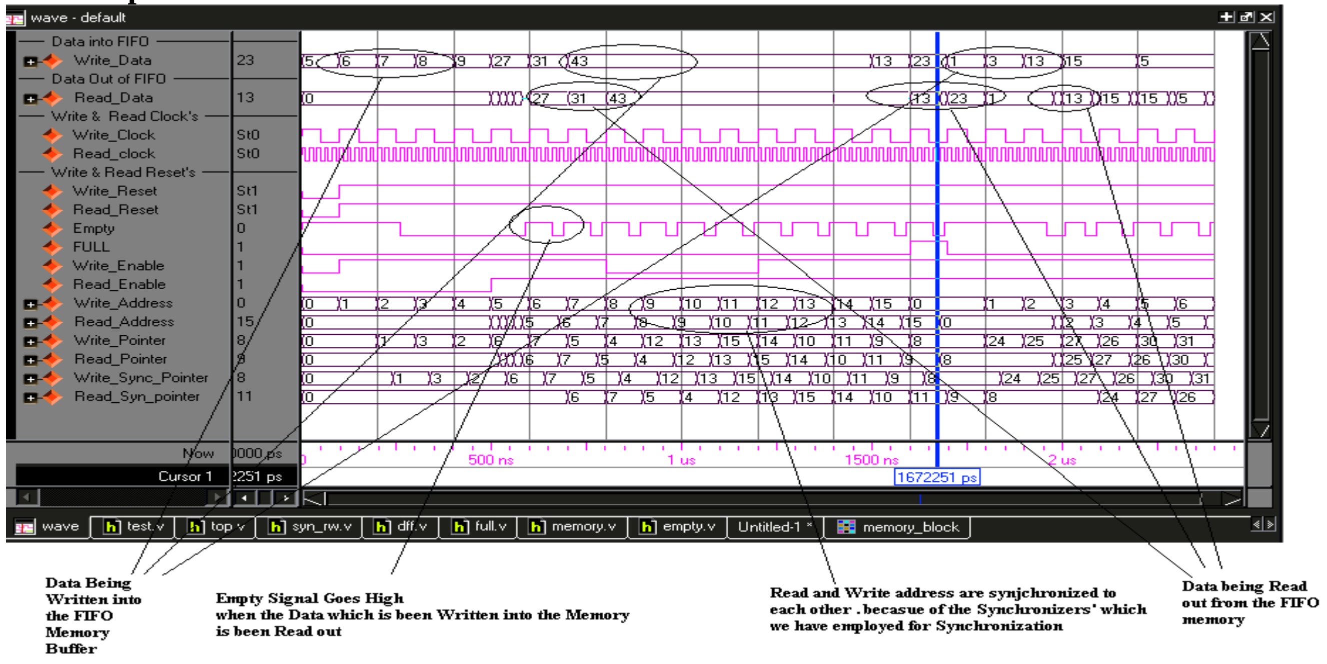
Fig 2.7: Output Waveform 4

- ➢ As we can see from the waveform when the write_Address reached 15 i.e.

The FIFO memory is filled hence the Full signal goes high

- ➢ Empty signal goes high when ever the data which is being written into the memory is Read out

2.3.3 Output Waveform 3



Data Being Written into the FIFO Memory Buffer

Empty Signal Goes High when the Data which is been Written into the Memory is been Read out

Read and Write address are synjchronized to each other . becasue of the Synchronizers' which we have employed for Synchronization

Data being Read out from the FIFO memory

Fig 2.8: Output Waveform 4

- ➢ All the operations shown again for all cases

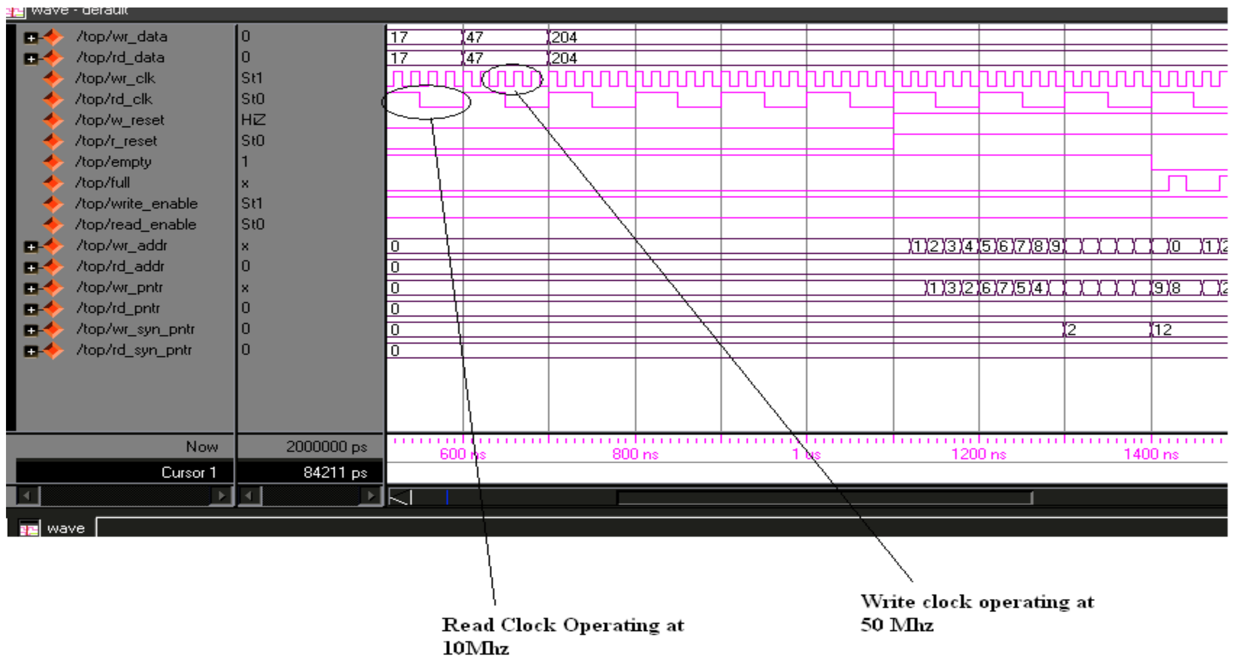


Fig 2.8(a): FIFO operation for different clock frequencies

> Here we can notice that even if we change the write and read clock frequencies the FIFO design works

2.4: TEST BENCH FOR FIFO

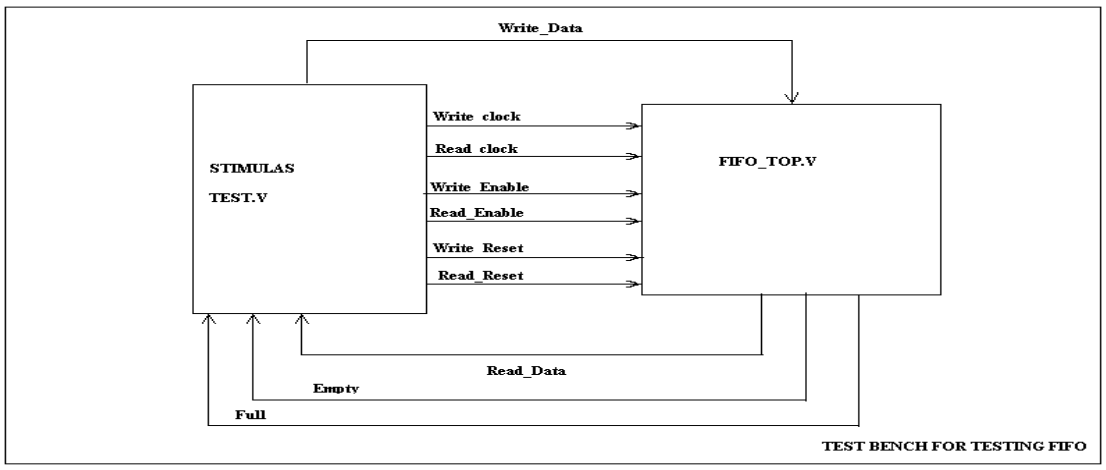


Fig 2.9: Test Bench

2.3.1Stimulus Block

We now write the stimulus block to check if the Asynchronous FIFO Design is functioning correctly. In this case we must control the following

- Write & Read Clock's
- Write & Read Resets
- Write & Read Enable
- Write Data

So that the regular function of the Asynchronous FIFO and the Reset and Enable Mechanism are both tested, we use the Waveform shown in Fig 2.9(a) to test the Design, Waveform's for Write clock, Read Clock, Write & Read Resets , Write & Read Enable are shown.

Data out, Empty and full signal's are then monitored. as we can see in the fig 2.9(b)

Stimulus Block

```
module tb_top();


reg wr_clk,rd_clk;
reg[7:0] data_in;
wire[7:0] data_out;
wire rd_empty,wr_full;
reg reset_w;
reg reset_r;
reg write_enable,read_enable;

top top_1(.wr_data(data_in),
        .rd_data(data_out),
        .wr_clk(wr_clk),
        .rd_clk(rd_clk),
        .w_reset(reset_w),
        .r_reset(reset_r),
        .write_enable(write_enable) ,
        .read_enable(read_enable),

        .empty(rd_empty),
        .full(wr_full));


initial
begin
  #0data_in=8'h0;
   #50_000 data_in=8'b00000001;    //  DATA WHICH IS SUPPLIED
   #80_000 data_in=8'h2;
   #70_000 data_in=8'h3;
   #79_000 data_in=8'h4;
   #80_000 data_in=8'h5;
   #40_000 data_in=8'h6;
   #60_000 data_in=8'h7;
   #50_000 data_in=8'h8;
   #50_000 data_in=8'h9;
   #20_000 data_in=8'h10;
   #70_000 data_in=8'h11;
   #80_000 data_in=8'h12;
   #19_000 data_in=8'h13;
   #10_000 data_in=8'h14;
   #80_000 data_in=8'h15;
end

initial
begin
     wr_clk=1'b0;
      write_enable=1'b0;
```

```
       read_enable=1'b0;
   end
     initial
    always
  #50000 wr_clk=~wr_clk; //end                    // READ AND WRITE CLOCK GENERATION
     rd_clk=1'b0;
     initial
   begin
   always
   #10000 rd_clk=~rd_clk;
  end
initial
     reset_r=1'b0;
   begin
        initial
   #5000 reset_r=1'b1;
   //end
initial

     reset_w =1'b0;
     initial
       #5000 reset_w=1'b1;
initial

   #5000 write_enable=1'b1;
   initial
   # 50000 read_enable=1'b1;
   initial
   begin
   #1000000000 $finish; end

    initial
   $monitor( "$time data_out,empty ,full= %d %d %d",data_out,rd_empty,wr_full);



endmodule
```

Once the stimulus Block is completed, we are ready to run the stimulation and verify the functional correctness of the design block. The output obtained when the stimulus and design blocks are stimulated is shown in Fig 2.9(a) and 2.9(b)

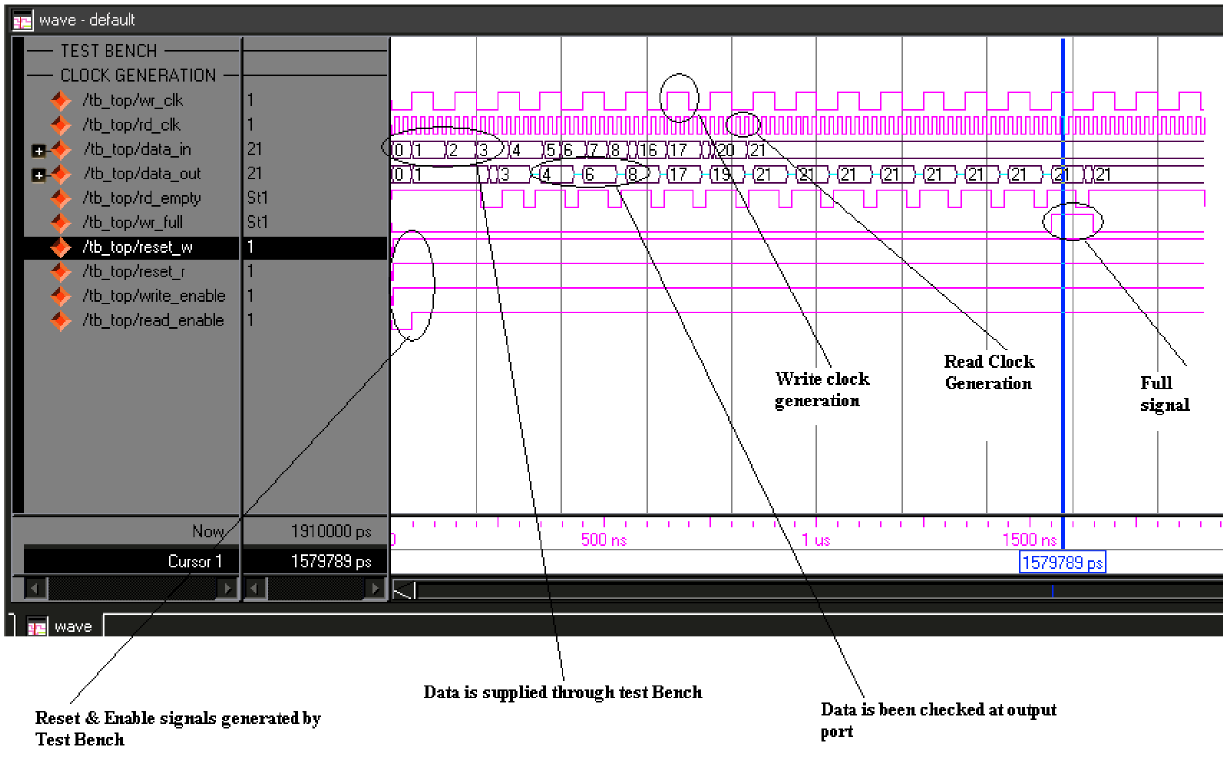
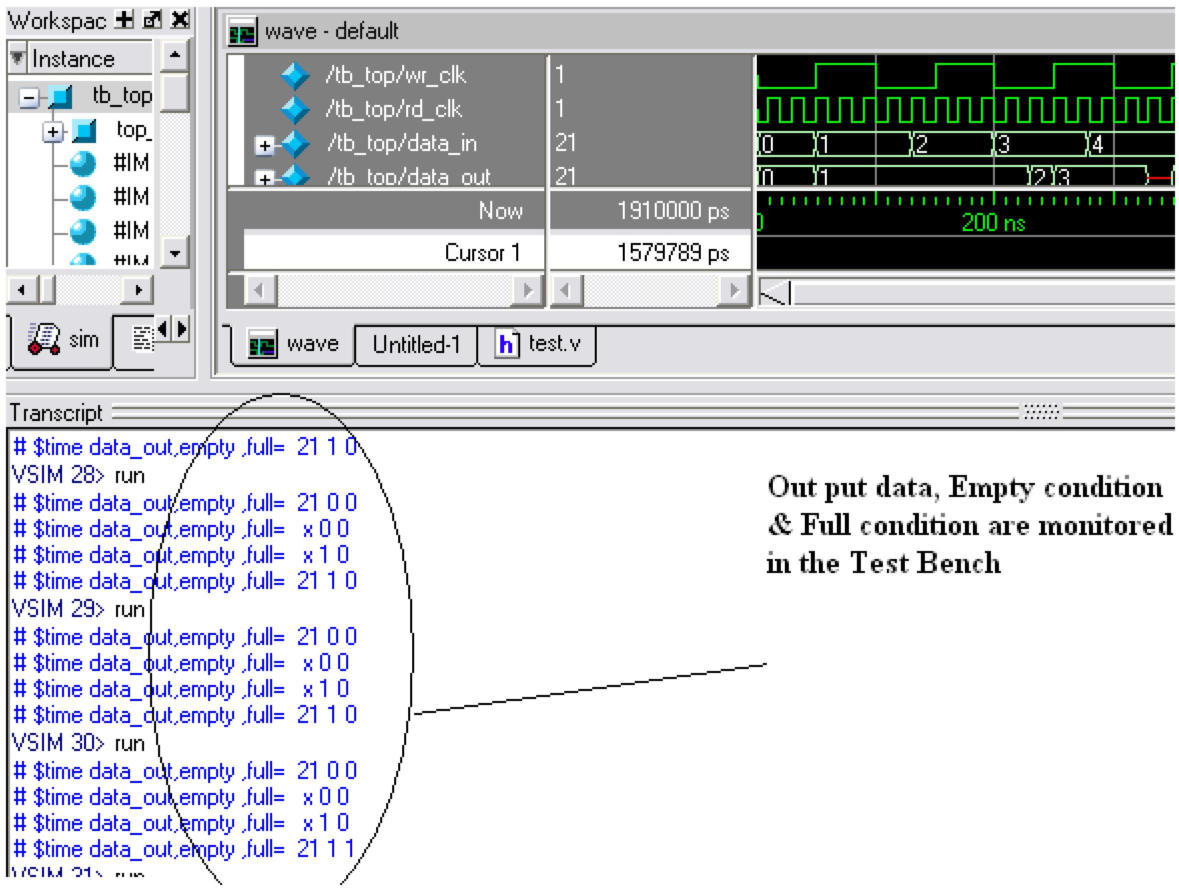


Fig 2.9(a) : Stimulus Waveform



```
# $time data_out,empty ,full=  21 1 0
VSIM 28> run
# $time data_out,empty ,full=  21 0 0
# $time data_out,empty ,full=   x 0 0
# $time data_out,empty ,full=   x 1 0
# $time data_out,empty ,full=  21 1 0
VSIM 29> run
# $time data_out,empty ,full=  21 0 0
# $time data_out,empty ,full=   x 0 0
# $time data_out,empty ,full=   x 1 0
# $time data_out,empty ,full=  21 1 0
VSIM 30> run
# $time data_out,empty ,full=  21 0 0
# $time data_out,empty ,full=   x 0 0
# $time data_out,empty ,full=   x 1 0
# $time data_out,empty ,full=  21 1 1
VSIM 31> run
```

Fig 2.9(b): Monitoring output's (out put of stimulus)

2.5 Logic Synthesis

Logic Synthesis is the process of converting a high level description of the design into an optimized, gate-level representation, using the cells in the technology library.

Logic Synthesis tool accepts high level descriptions at the register transfer Level (RTL). And a technology library produces an optimized gate level net list, Translation, Logic optimization, and technology mapping are the internal process in a logic synthesis tool and are normally invisible to the user. Not all verilog constructs are acceptable to a logic synthesis tool.

Synthesis Summary For Asynchronous FIFO

```
=========================================================================
*                       Advanced HDL Synthesis                          *
=========================================================================

Advanced RAM inference ...
Advanced multiplier inference ...
Advanced Registered AddSub inference ...
```

```
Dynamic shift register inference ...

=========================================================================
HDL Synthesis Report

Macro Statistics
# LUT RAMs                         : 1
 16x8-bit dual-port distributed RAM: 1
# Adders/Subtractors               : 2
 5-bit adder                       : 2
# Registers                        : 10
 1-bit register                    : 2
 5-bit register                    : 8
# Comparators                      : 3
 4-bit comparator equal            : 1
 5-bit comparator equal            : 2
# Xors                             : 8
 1-bit xor2                        : 8

=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : top.ngr
Top Level Output File Name         : top
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 24

Macro Statistics :
# RAM                              : 1
#      16x8-bit dual-port distributed RAM: 1
# Registers                        : 10
#      1-bit register              : 2
#      5-bit register              : 8
# Comparators                      : 3
#      4-bit comparator equal      : 1
#      5-bit comparator equal      : 2

Cell Usage :
# BELS                             : 39
#      LUT1                        : 4
#      LUT1_D                      : 1
#      LUT2                        : 16
#      LUT2_D                      : 1
#      LUT2_L                      : 3
#      LUT3                        : 2
#      LUT3_L                      : 4
#      LUT4                        : 4
#      LUT4_L                      : 4
# FlipFlops/Latches                : 41
#      FDC                         : 1
#      FDCE                       : 21
#      FDP                        : 1
#      FDR                        : 18
# RAMS                            : 8
#      RAM16X1D                   : 8
# Clock Buffers                   : 2
#      BUFGP                      : 2
# IO Buffers                      : 22
#      IBUF                       : 12
#      OBUF                       : 10
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s200ft256-4
```

Number of Slices: 37 out of   1920   1%
Number of Slice Flip Flops: 41 out of   3840   1%
Number of 4 input LUTs: 47 out of   3840   1%
Number of bonded IOBs: 22 out of   173   12%
Number of GCLKs: 2 out of   8   25%

Total memory usage is 64632 kilobytes

A Few Snap Shots of Synthesis process

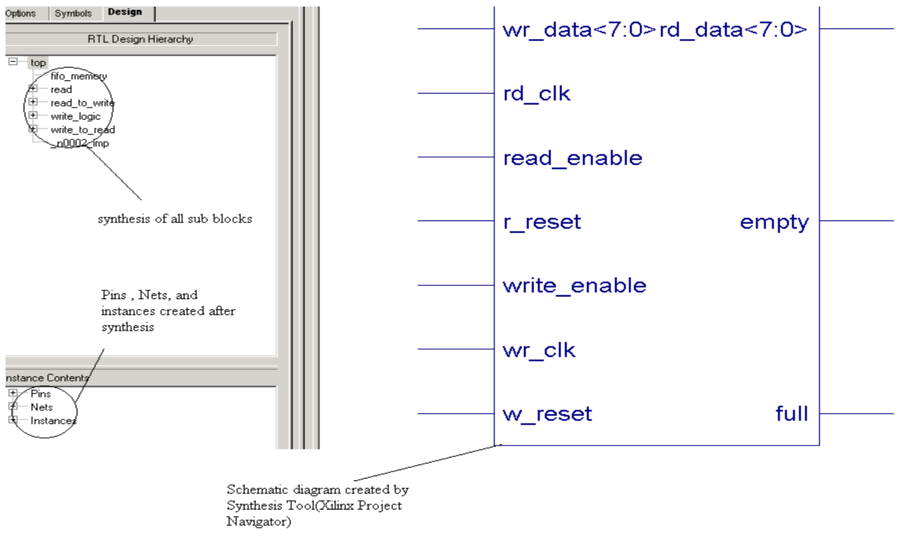


Fig 2.9(c): Schematic of the FIFO converted from Verilog Code

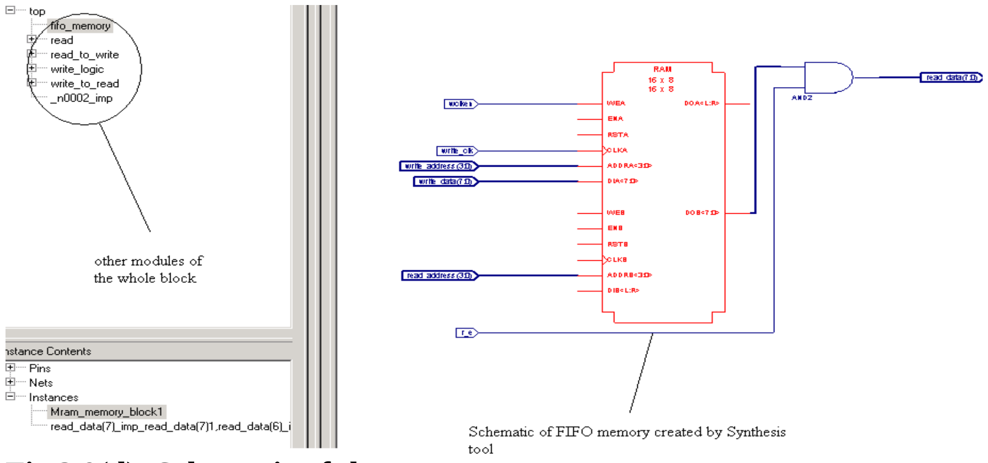


Fig 2.9(d): Schematic of the memory

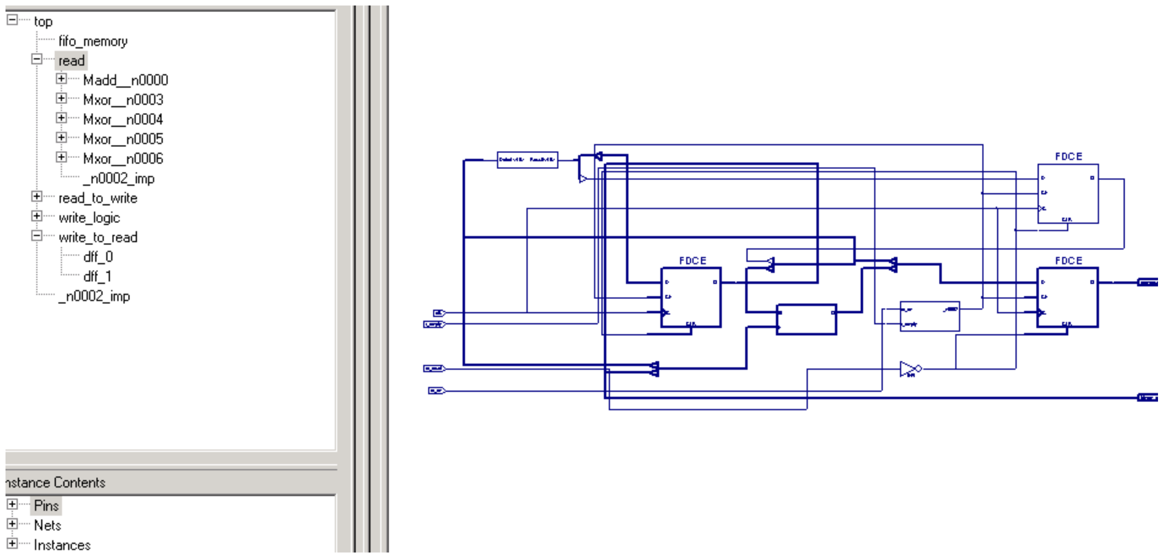


Fig 2.9(d): Schematic of the Counter

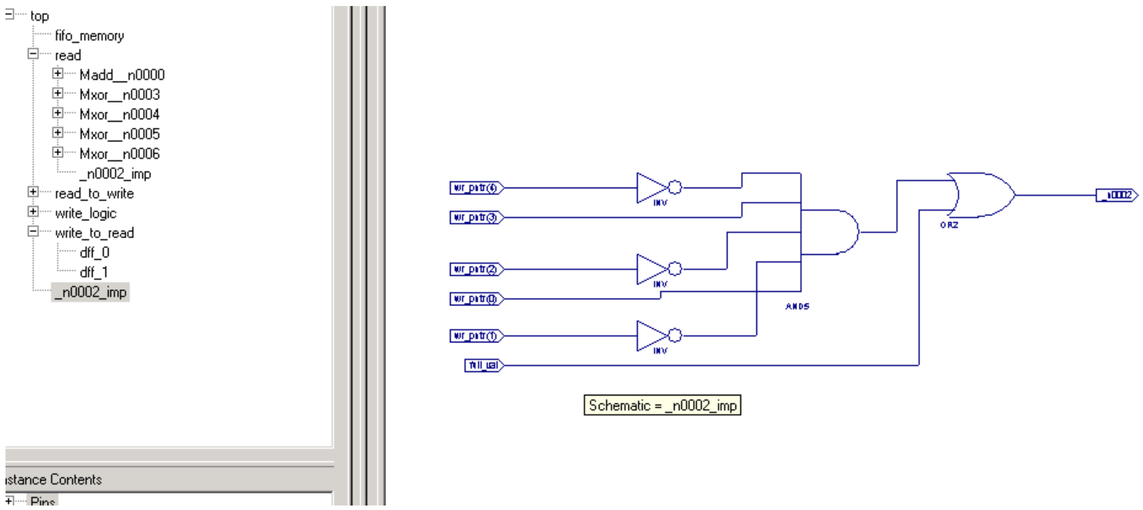
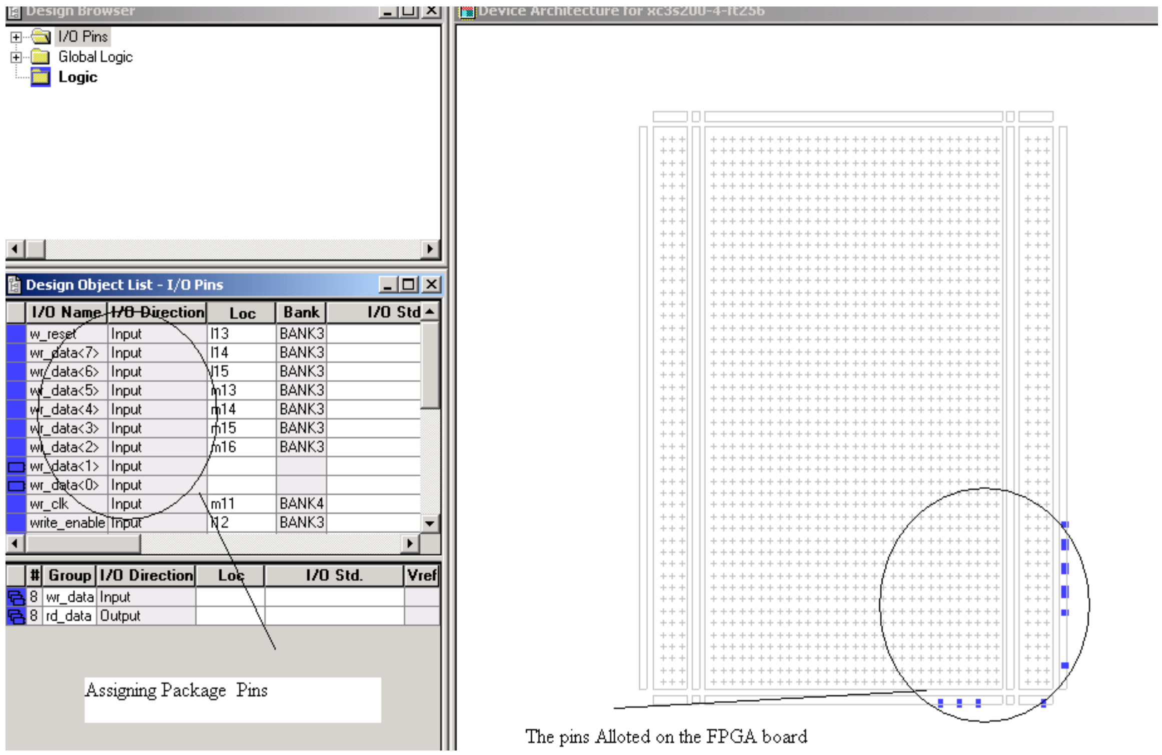


Fig 2.9(e): Schematic of Full and empty logic


Design Browser
I/O Pins
Global Logic
Logic
Design Object List - I/O Pins
Device Architecture for xc3s200-4-ft256
Assigning Package Pins
The pins Alloted on the FPGA board

I/O Name	I/O Direction	Loc	Bank	I/O Std
w_reset	Input	l13	BANK3	
wr_data<7>	Input	l14	BANK3	
wr_data<6>	Input	l15	BANK3	
wr_data<5>	Input	m13	BANK3	
wr_data<4>	Input	m14	BANK3	
wr_data<3>	Input	m15	BANK3	
wr_data<2>	Input	m16	BANK3	
wr_data<1>	Input			
wr_data<0>	Input			
wr_clk	Input	m11	BANK4	
write_enable	Input	l12	BANK3	

#	Group	I/O Direction	Loc	I/O Std.	Vref
8	wr_data	Input			
8	rd_data	Output			

Fig 2.9(f): Pin assignments