PCI Target device

Tables of contents

1. Block Diagram

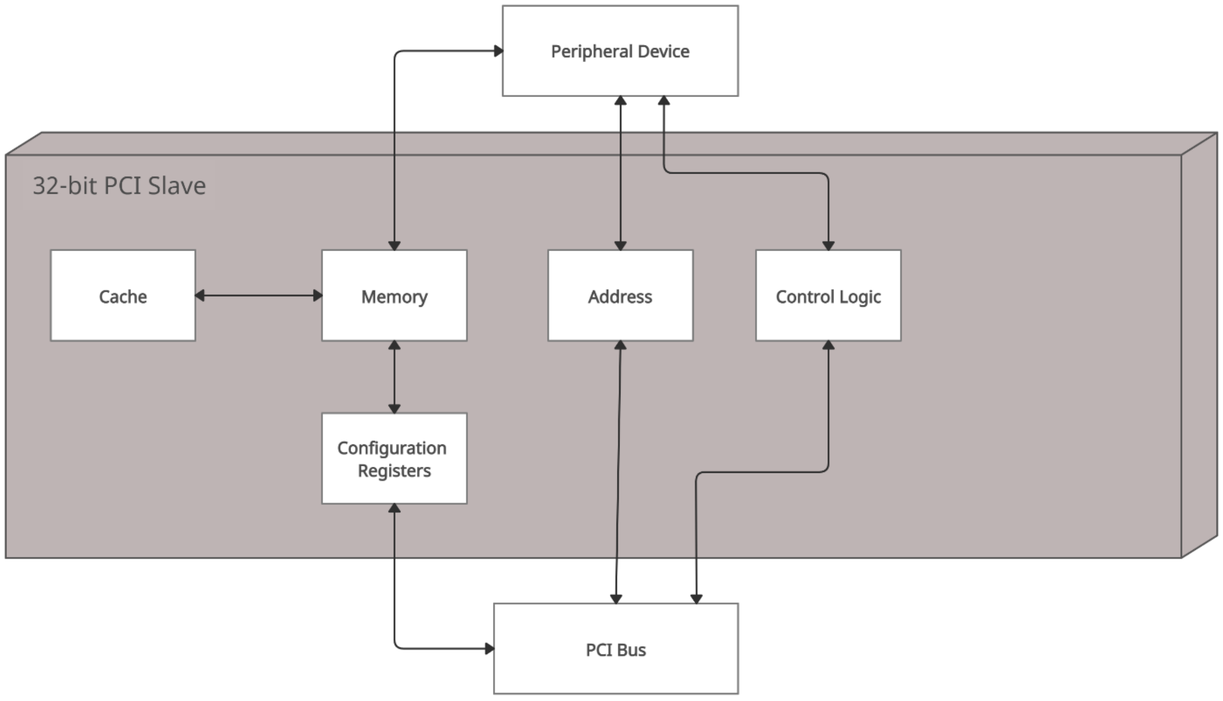


Figure 1 represents block diagram

<u>Explanation:</u>

- The Block diagram shows PCI slave which can be a network card or Ethernet card for example.
- The PCI slave is connected to peripheral device and PCI bus.
- The control logic will determine the operation that the target will do (read or write) and send it to the buffers to send or receive data from PCI bus.
- The configuration registers will help in read or write operation (internally) but will not participate in any operation with outside peripheral device.
- The Memory holds the data that the slave reads during read operation or the slave writes in it during write operation.
- The Cache is a buffer that is used when there is overflow and the memory is full, where the data in the memory is passed to the cache so, the next transactions is done in the memory.

2. Signals Description

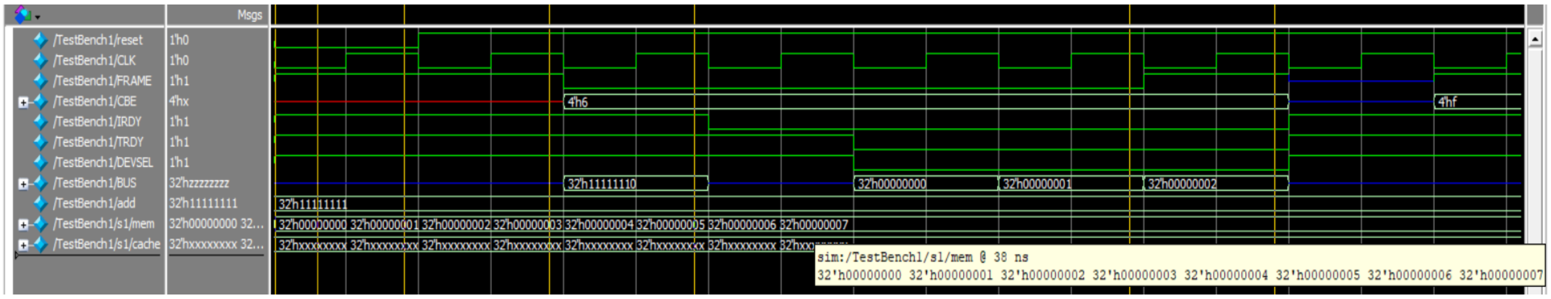
Signal	Description
CLK	[Input] signal to the module represents the synchronous clock
BUS	[Inout] PCI Bus that carries the data to memory during write operation and from memory during read operation.
CBE	[Input] Bus Command and Byte Enables are multiplexed on the same PCI pins. During the address phase of a transaction, define the bus command. And byte enable during data phase.
FRAME	[Input] signal which determines the beginning and duration of transaction.
IRDY	[Input] signal which represents that initiator is ready to send or receive data.
TRDY	[Output]Target Ready indicates the target agent's (selected device's) ability to complete the current data phase of the transaction.
Reset	[Input] Reset is used to bring PCI-specific registers, sequencers, and signals to a consistent state.
DEVSEL	[Output] signal which represents that target has recognized its address.
Add	[Parameter] 32-bit Parameter that represents the address of the slave .
Ro	[Parameter] Indicates the read operation
Wo	[Parameter] Indicates the write operation
Mem	[Reg] Represents the target memory we will write and read from.
Cache	[Reg] Additional memory used to prevent overflow as it saves all the data stored in the mem [reg] to allow the mem to receive the new data that would

	of caused overflow.
r_w	[Reg] Indicates the operation that the slave (target) will operate on.
FR_DA	[Reg] Indicates if the frame has been de-asserted or not.
Overflow	[Reg] Indicates if overflow occurs.
I	[Reg] Used as index.
bus_reg	[Reg] Used to read the data in the slave to transfer it to the BUS to be sent to the master. (Inout wires can't be used in @always blocks so we assign BUS to bus_reg during any read operation)
Enable	[Reg] Indicates that the BUS has the address of the slave correctly.
clk_count	[Reg] Used to count the number of cycles.
over_count	[Reg] Used to count the number of cycles after the occurrence of any overflow.

3. Different Scenarios

• Test bench 1

Normal read operation from add[0] (address) for 3 cycles where we get 3 zeros.



```
32'h00000000 32'h00000001 32'h00000002 32'h00000003 32'h00000004 32'h00000005 32'h00000006 32'h00000007
32'hxxxxxxxx 32'hxxxxxxxx 32'hxxxxxxxx 32'hxxxxxxxx 32'hxxxxxxxx 32'hxxxxxxxx 32'hxxxxxxxx 32'hxxx
sim:/TestBench1/s1/mem @ 38 ns
32'h00000000 32'h00000001 32'h00000002 32'h00000003 32'h00000004 32'h00000005 32'h00000006 32'h00000007
```

Fig. 1

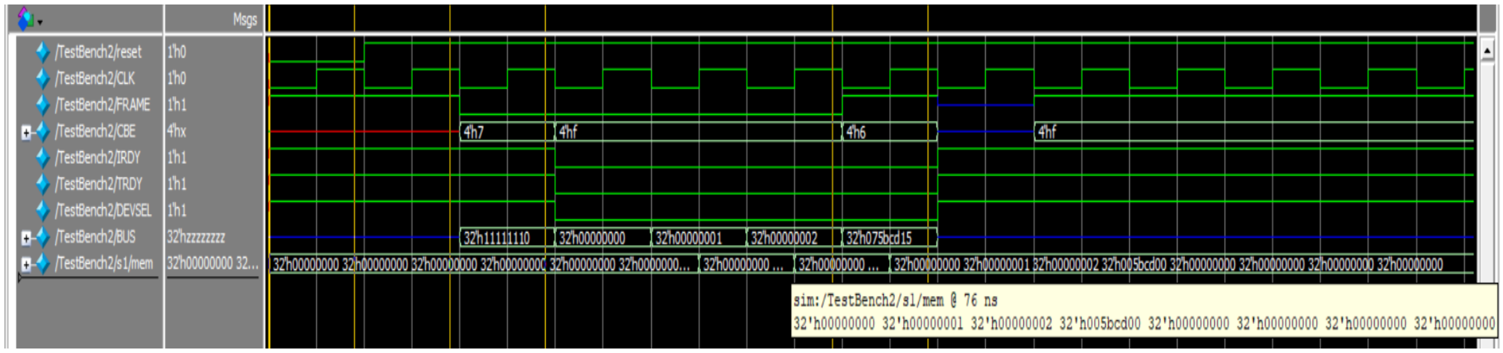
```
module TestBench1();
////////// DEFAULT  \\\\\\\\

reg reset;
reg CLK,FRAME,IRDY;
reg [3:0] CBE;
wire TRDY,DEVSEL;
wire [31:0] BUS;
parameter add=32'h11111111;
reg cond; // Writing Condition
reg[31:0] buffer;

assign BUS = (cond ) ? buffer : (32'hzzzzzzzz);

initial
begin
$monitor($time, , ,  " %d   %b  %b  %b ", BUS, FRAME, TRDY, DEVSEL);

reset=1;
CLK=0;
IRDY=1;
FRAME=1;
cond=0;
reset=0;
#10
reset=1;
#10
```

```
////////// DEFAULT  \\\\\\\\
//code
cond=1;
CBE=4'b0110;
buffer[31:0]= add;
buffer[2:0]= 3'b000;
FRAME=0;
#10
cond=0;
IRDY=0;
#30
FRAME=1;
#10
IRDY=1;




//code

///////// DEFAULT \\\\\\\\


FRAME=1'bz;
buffer=32'hzzzzzzzz;
CBE=4'bzzzz;
#10
FRAME=1;
CBE=4'b1111;

end

always
begin
#5
CLK= ~ CLK;
end


slave s1 ( reset,CLK, TRDY,IRDY, BUS,CBE, FRAME, DEVSEL);
endmodule
```

- **Test bench 2**

Normal write operation for 4 cycles. (Input 4 numbers)

Byte enable is activated on the last written word (0110).



```
sim:/TestBench2/s1/mem @ 76 ns
32'h00000000 32'h00000001 32'h00000002 32'h005bcd00 32'h00000000 32'h00000000 32'h00000000 32'h00000000
```

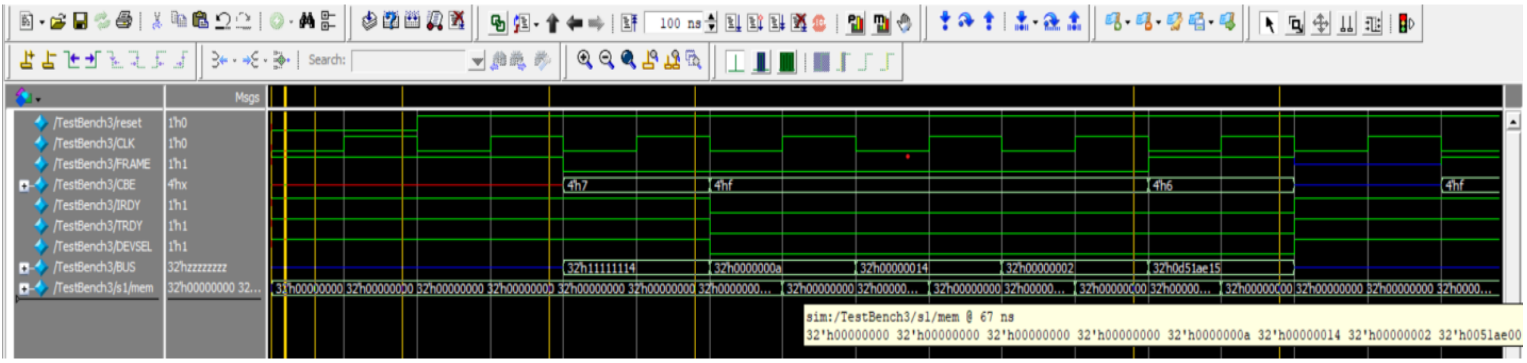
Fig. 2

```
module TestBench2();
////////// DEFAULT  \\\\\\\\

reg reset;
reg CLK,FRAME,IRDY;
reg [3:0] CBE;
wire TRDY,DEVSEL;
wire [31:0] BUS;
parameter add=32'h11111111;
reg cond; // Writing Condition
reg[31:0] buffer;

assign BUS = (cond ) ? buffer : (32'hzzzzzzzz);

initial
begin
$monitor($time, , ,  " %d   %b  %b  %b ", BUS, FRAME, TRDY, DEVSEL);

reset=1;
CLK=0;
IRDY=1;
FRAME=1;
cond=0;
reset=0;
#10
reset=1;
#10
////////// DEFAULT  \\\\\\\\
//code
cond=1;
FRAME=0;
buffer[31:0]= add;
```

```
buffer[2:0]= 3'b000;
CBE=4'b0111;
#10
buffer=0;
CBE=4'b1111;
IRDY=0;


#10
buffer=1;
CBE=4'b1111;

#10
buffer=2;
CBE=4'b1111;
#10
FRAME=1;
buffer=123456789;
CBE=4'b0110;
#10
//code
///////// DEFAULT \\\\\\\\
IRDY=1;
FRAME=1'bz;
buffer=32'hzzzzzzzz;
CBE=4'bzzzz;
#10
FRAME=1;
CBE=4'b1111;
end

always
begin
#5
CLK= ~ CLK;
end


slave s1 ( reset,CLK, TRDY,IRDY, BUS,CBE, FRAME, DEVSEL);
endmodule
```

- ## Test bench 3

Write operation but in this test bench we write from index 4 mem[4] , 8 write for 4 clock cycles. (Byte enable is activated 0110 on the last written word)



```
sim:/TestBench3/s1/mem @ 67 ns
32'h00000000 32'h00000000 32'h00000000 32'h00000000 32'h0000000a 32'h00000014 32'h00000002 32'h0051ae00
```

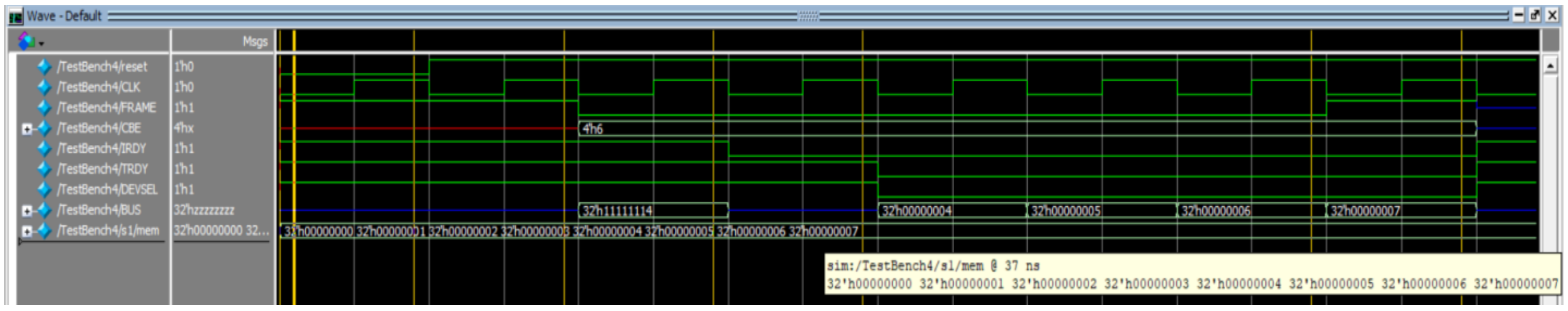
Fig. 3

```
module TestBench3();
////////// DEFAULT  \\\\\\\\

reg reset;
reg CLK,FRAME,IRDY;
reg [3:0] CBE;
wire TRDY,DEVSEL;
wire [31:0] BUS;
parameter add=32'h11111111;
reg cond; // Writing Condition
reg[31:0] buffer;

assign BUS = (cond ) ? buffer : (32'hzzzzzzzz);

initial
begin
$monitor($time, , ,  " %d   %b  %b  %b ", BUS, FRAME, TRDY, DEVSEL);

reset=1;
CLK=0;
IRDY=1;
FRAME=1;
cond=0;
reset=0;
#10
reset=1;
#10
```

```
////////// DEFAULT  \\\\\\\\
//code
cond=1;
FRAME=0;
buffer[31:0]= add;
buffer[2:0]= 3'b100;
CBE=4'b0111;
#10
buffer=10;
CBE=4'b1111;
IRDY=0;


#10
buffer=20;
CBE=4'b1111;

#10
buffer=2;
CBE=4'b1111;
#10
FRAME=1;
buffer=223456789;
CBE=4'b0110;
#10



//code
///////// DEFAULT \\\\\\\\

IRDY=1;
FRAME=1'bz;
buffer=32'hzzzzzzzz;
CBE=4'bzzzz;
#10
FRAME=1;
CBE=4'b1111;

end

always
begin
#5
CLK= ~ CLK;
end


slave s1 ( reset,CLK, TRDY,IRDY, BUS,CBE, FRAME, DEVSEL);
endmodule
```

- **Test bench 4**

  Read operation from mem[2] to mem[5]



```
sim:/TestBench4/s1/mem @ 37 ns
32'h00000000 32'h00000001 32'h00000002 32'h00000003 32'h00000004 32'h00000005 32'h00000006 32'h00000007
```

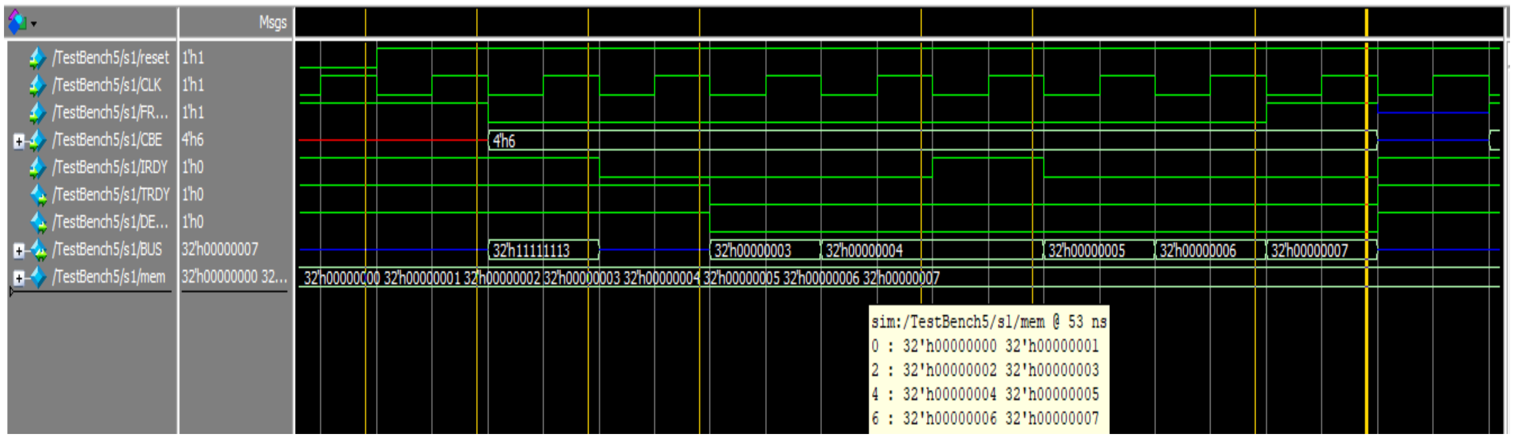
Fig. 4

```
module TestBench4();
////////// DEFAULT  \\\\\\\\

reg reset;
reg CLK,FRAME,IRDY;
reg [3:0] CBE;
wire TRDY,DEVSEL;
wire [31:0] BUS;
parameter add=32'h11111111;
reg cond; // Writing Condition
reg[31:0] buffer;

assign BUS = (cond ) ? buffer : (32'hzzzzzzzz);

initial
begin
$monitor($time, , ,  " %d   %b  %b  %b ", BUS, FRAME, TRDY, DEVSEL);

reset=1;
CLK=0;
IRDY=1;
FRAME=1;
cond=0;
reset=0;
#10
reset=1;
#10
////////// DEFAULT  \\\\\\\\
//code
cond=1;
CBE=4'b0110;
buffer[31:0]= add;
buffer[2:0]= 3'b100;
FRAME=0;
```

```
#10
cond=0;
IRDY=0;
#40
FRAME=1;
#10
IRDY=1;

//code

///////// DEFAULT \\\\\\\\


FRAME=1'bz;
buffer=32'hzzzzzzzz;
CBE=4'bzzzz;
#10
FRAME=1;
CBE=4'b1111;

end

always
begin
#5
CLK= ~ CLK;
end


slave s1 ( reset,CLK, TRDY,IRDY, BUS,CBE, FRAME, DEVSEL);

endmodule
```

- **Test bench 5**

The slave reads from mem[3] and after 2 transactions IRDY is asserted for 1 clock cycle. (By default mem[i] = i)


Msgs
/TestBench5/s1/reset 1'h1
/TestBench5/s1/CLK 1'h1
/TestBench5/s1/FR... 1'h1
/TestBench5/s1/CBE 4'h6
/TestBench5/s1/IRDY 1'h0
/TestBench5/s1/TRDY 1'h0
/TestBench5/s1/DE... 1'h0
/TestBench5/s1/BUS 32'h00000007
/TestBench5/s1/mem 32'h00000000 32...
4'h6
32'h11111113
32'h00000003
32'h00000004
32'h00000005
32'h00000006
32'h00000007
32'h00000000 32'h00000001 32'h00000002 32'h00000003 32'h00000004 32'h00000005 32'h00000006 32'h00000007

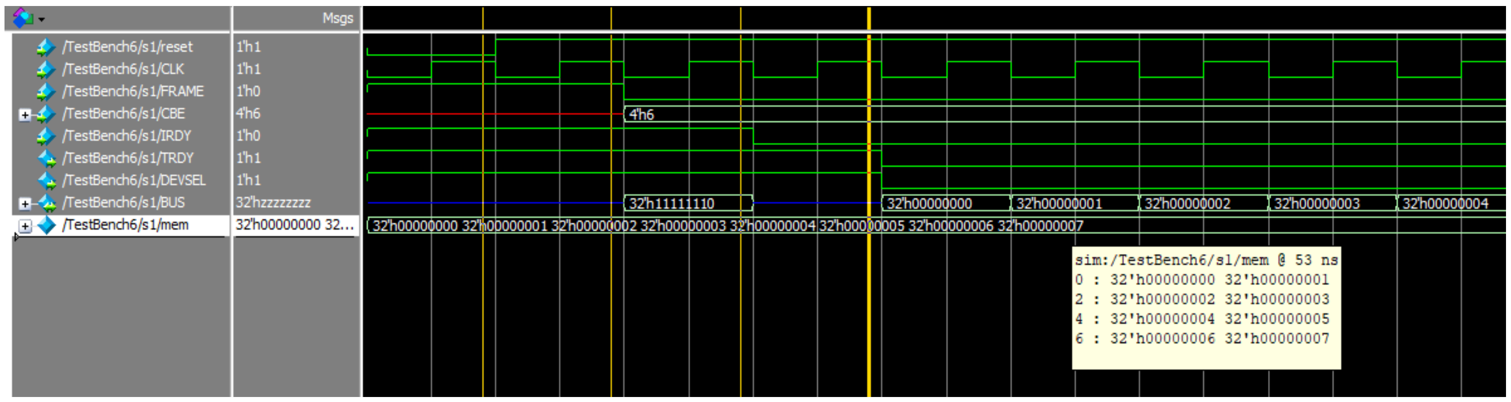
```
sim:/TestBench5/s1/mem @ 53 ns
0 : 32'h00000000 32'h00000001
2 : 32'h00000002 32'h00000003
4 : 32'h00000004 32'h00000005
6 : 32'h00000006 32'h00000007
```

Fig. 5

```verilog
module TestBench5();
reg reset;
reg CLK,FRAME,IRDY;
reg [3:0] CBE;
wire TRDY,DEVSEL;
wire [31:0] BUS;
parameter add=32'h11111111;
reg cond; // Writing Condition
reg[31:0] buffer;

assign BUS = (cond ) ? buffer : (32'hzzzzzzzz);

initial
begin
$monitor($time, , ,  " %d   %b  %b  %b ", BUS, FRAME, TRDY, DEVSEL);

reset=1;
CLK=0;
IRDY=1;
FRAME=1;
cond=0;
reset=0;
#10
reset=1;
#10
////////// DEFAULT  \\\\\\\\

cond=1;
CBE=4'b0110;
buffer[31:0]= add;
buffer[2:0]= 3'b011;
FRAME=0;
#10
cond=0;
IRDY=0;
#30
IRDY=1;
#10
IRDY=0;
#20
FRAME=1;
#10
IRDY=1;

///////// DEFAULT \\\\\\\\

FRAME=1'bz;
buffer=32'hzzzzzzzz;
CBE=4'bzzzz;
#10
FRAME=1;
CBE=4'b1111;
end

always
begin
#5
CLK= ~ CLK;
end

slave s1 ( reset,CLK, TRDY,IRDY, BUS,CBE, FRAME, DEVSEL);

endmodule
```

- **Test bench 6**

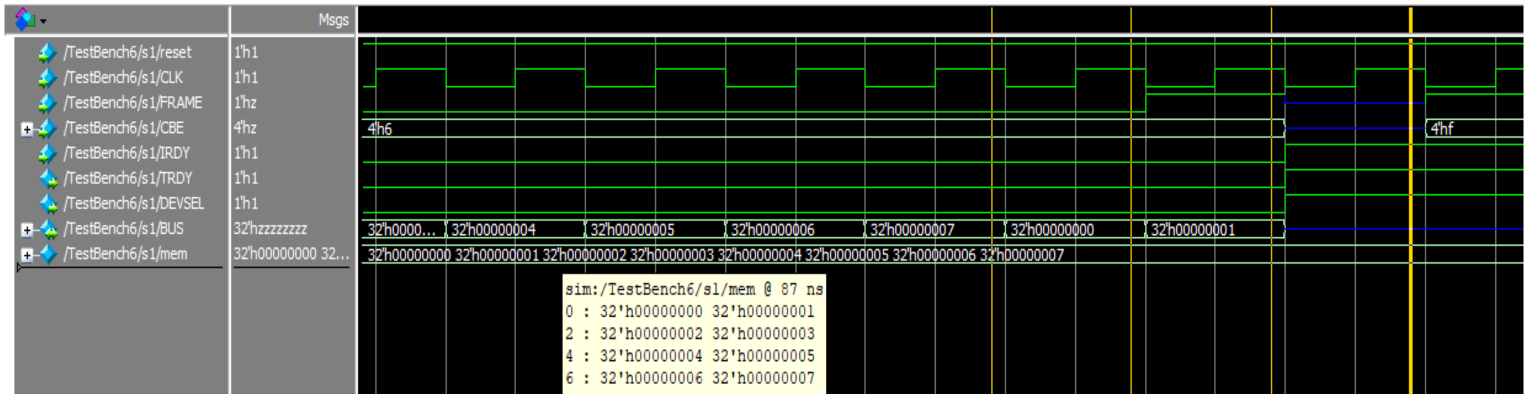
The slave reads from mem[0] for 10 cycles then [i] resets and starts from the beginning. (By default mem[i] = i)


4'h6
32'h11111110
32'h00000000
32'h00000001
32'h00000002
32'h00000003
32'h00000004
32'h00000000 32'h00000001 32'h00000002 32'h00000003 32'h00000004 32'h00000005 32'h00000006 32'h00000007

```
                      Msgs
/TestBench6/s1/reset  1'h1
/TestBench6/s1/CLK    1'h1
/TestBench6/s1/FRAME  1'h0
/TestBench6/s1/CBE    4'h6
/TestBench6/s1/IRDY   1'h0
/TestBench6/s1/TRDY   1'h1
/TestBench6/s1/DEVSEL 1'h1
/TestBench6/s1/BUS    32'hzzzzzzzz
/TestBench6/s1/mem    32'h00000000 32...
```

```
sim:/TestBench6/s1/mem @ 53 ns
0 : 32'h00000000 32'h00000001
2 : 32'h00000002 32'h00000003
4 : 32'h00000004 32'h00000005
6 : 32'h00000006 32'h00000007
```

Fig. 6.1


4'h6
4'hf
32'h0000...
32'h00000004
32'h00000005
32'h00000006
32'h00000007
32'h00000000
32'h00000001
32'h00000000 32'h00000001 32'h00000002 32'h00000003 32'h00000004 32'h00000005 32'h00000006 32'h00000007

```
                      Msgs
/TestBench6/s1/reset  1'h1
/TestBench6/s1/CLK    1'h1
/TestBench6/s1/FRAME  1'hz
/TestBench6/s1/CBE    4'hz
/TestBench6/s1/IRDY   1'h1
/TestBench6/s1/TRDY   1'h1
/TestBench6/s1/DEVSEL 1'h1
/TestBench6/s1/BUS    32'hzzzzzzzz
/TestBench6/s1/mem    32'h00000000 32...
```

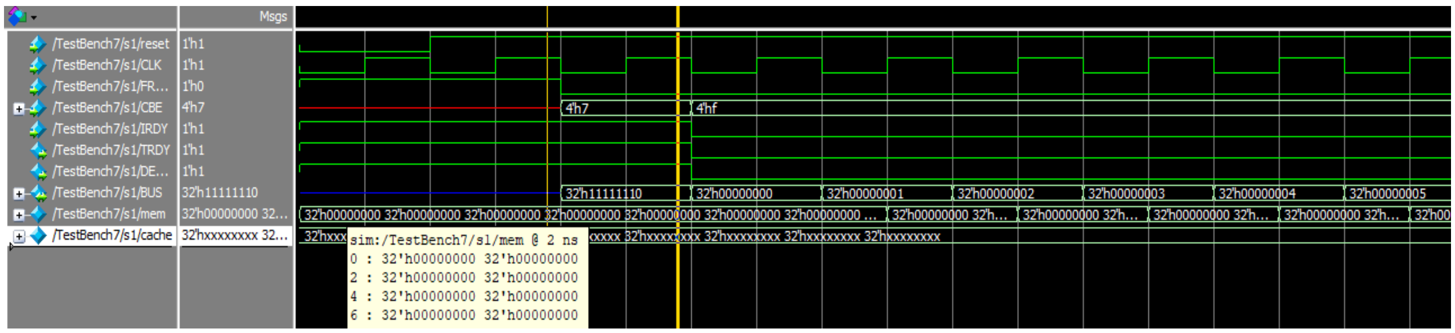
```
sim:/TestBench6/s1/mem @ 87 ns
0 : 32'h00000000 32'h00000001
2 : 32'h00000002 32'h00000003
4 : 32'h00000004 32'h00000005
6 : 32'h00000006 32'h00000007
```

Fig. 6.2

```
module TestBench6();

////////// DEFAULT  \\\\\\\\

reg reset;
reg CLK,FRAME,IRDY;
reg [3:0] CBE;
wire TRDY,DEVSEL;
wire [31:0] BUS;
parameter add=32'h11111111;
reg cond; // Writing Condition
reg[31:0] buffer;

assign BUS = (cond ) ? buffer : (32'hzzzzzzzz);

initial
begin
$monitor($time, , ,  " %d   %b  %b  %b ", BUS, FRAME, TRDY, DEVSEL);

reset=1;
CLK=0;
IRDY=1;
FRAME=1;
cond=0;
reset=0;
#10
reset=1;
#10
////////// DEFAULT  \\\\\\\\

//code
cond=1;
CBE=4'b0110;
buffer[31:0]= add;
buffer[2:0]= 3'b000;
FRAME=0;
#10
cond=0;
IRDY=0;
#100
FRAME=1;
#10
IRDY=1;

//code

///////// DEFAULT \\\\\\\\


FRAME=1'bz;
buffer=32'hzzzzzzzz;
CBE=4'bzzzz;
#10
FRAME=1;
CBE=4'b1111;
end

always
begin
#5
CLK= ~ CLK;
end

slave s1 ( reset,CLK, TRDY,IRDY, BUS,CBE, FRAME, DEVSEL);

endmodule
```

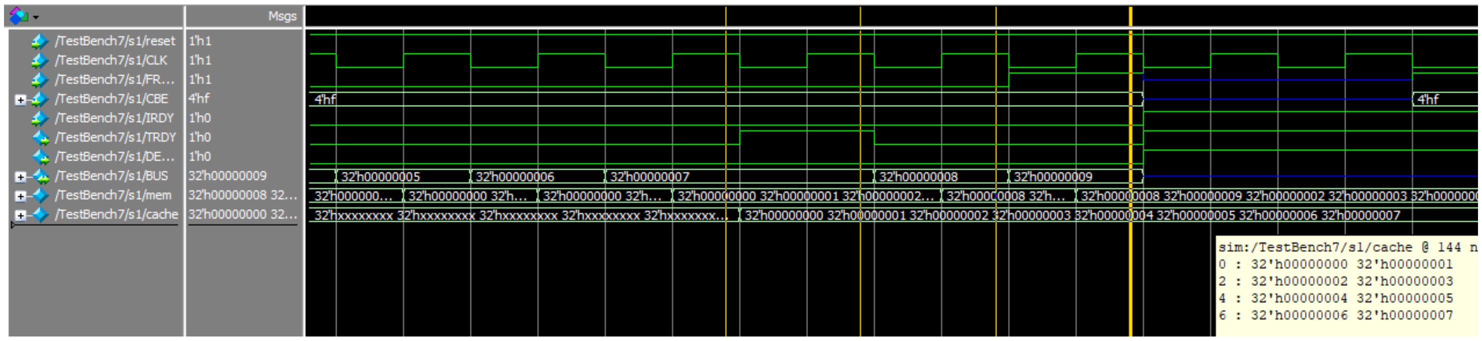
- **Test bench 7**

The slave writes for 10 cycles (By default mem= 0) and shows TRDY delay.



```
sim:/TestBench7/s1/mem @ 2 ns
0 : 32'h00000000 32'h00000000
2 : 32'h00000000 32'h00000000
4 : 32'h00000000 32'h00000000
6 : 32'h00000000 32'h00000000
```

Fig. 7.1


Msgs
/TestBench7/s1/reset 1'h1
/TestBench7/s1/CLK 1'h1
/TestBench7/s1/FR... 1'h1
/TestBench7/s1/CBE 4'hf
/TestBench7/s1/IRDY 1'h0
/TestBench7/s1/TRDY 1'h0
/TestBench7/s1/DE... 1'h0
/TestBench7/s1/BUS 32'h00000009
/TestBench7/s1/mem 32'h00000008 32...
/TestBench7/s1/cache 32'h00000000 32...
4'hf
32'h00000005
32'h00000006
32'h00000007
32'h00000008
32'h00000009
4'hf

```
sim:/TestBench7/s1/cache @ 144 n
0 : 32'h00000000 32'h00000001
2 : 32'h00000002 32'h00000003
4 : 32'h00000004 32'h00000005
6 : 32'h00000006 32'h00000007
```

Fig. 7.2

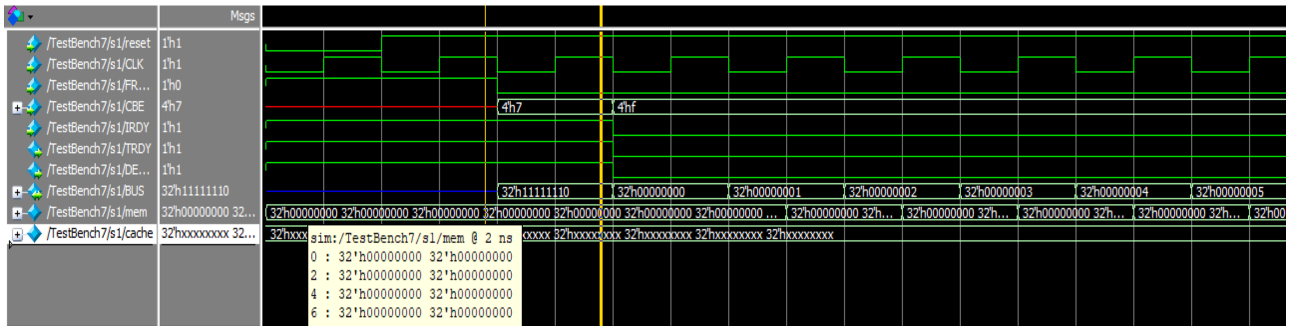
```
module TestBench7();

        ////////// DEFAULT  \\\\\\\\

        reg reset;
        reg CLK,FRAME,IRDY;
        reg [3:0] CBE;
        wire TRDY,DEVSEL;
        wire [31:0] BUS;
        parameter add=32'h11111111;
        reg cond; // Writing Condition
        reg[31:0] buffer;

        assign BUS = (cond ) ? buffer : (32'hzzzzzzzz);

        initial
        begin
        $monitor($time, , ,  " %d   %b  %b  %b ", BUS, FRAME, TRDY, DEVSEL);

        reset=1;
        CLK=0;
        IRDY=1;
        FRAME=1;
        cond=0;
        reset=0;
        #10
        reset=1;
        #10
        ////////// DEFAULT  \\\\\\\\

        //code
        cond=1;
        FRAME=0;
        buffer[31:0]= add;
        buffer[2:0]= 3'b000;
        CBE=4'b0111;

        #10
        buffer=0;
        CBE=4'b1111;
        IRDY=0;


        #10
        buffer=1;
        CBE=4'b1111;

        #10
        buffer=2;
        CBE=4'b1111;
        #10
        buffer=3;
        CBE=4'b1111;
        #10
        buffer=4;
        CBE=4'b1111;
        #10
        buffer=5;
        CBE=4'b1111;
        #10
        buffer=6;
        CBE=4'b1111;
        #10
        buffer=7;
```

```
CBE=4'b1111;
#20
buffer=8;
CBE=4'b1111;
#10

///////// DEFAULT \\\\\\\\

FRAME=1;
buffer=9;
CBE=4'b1111;
#10
IRDY=1;
FRAME=1'bz;
buffer=32'hzzzzzzzz;
CBE=4'bzzzz;

#10
FRAME=1;
CBE=4'b1111;









//code

///////// DEFAULT \\\\\\\\


FRAME=1'bz;
buffer=32'hzzzzzzzz;
CBE=4'bzzzz;
#10
FRAME=1;
CBE=4'b1111;

end

always
begin
#5
CLK= ~ CLK;
end


slave s1 ( reset,CLK, TRDY,IRDY, BUS,CBE, FRAME, DEVSEL);

endmodule
///////// DEFAULT \\\\\\\\
```

- **Test bench 8**

The slave writes for 10 cycles then reads for 5 cycles from mem[4] then IRDY is asserted for 2 cycles then the slave reads again for 2 more clock cycles.

Fig. 8.1

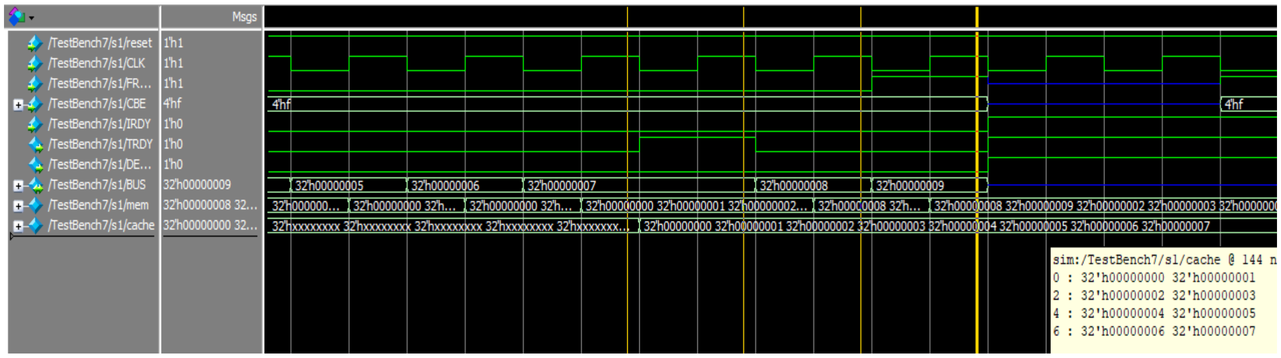


```
/TestBench7/s1/reset   1'h1
/TestBench7/s1/CLK     1'h1
/TestBench7/s1/FR...   1'h0
/TestBench7/s1/CBE     4'h7
/TestBench7/s1/IRDY    1'h1
/TestBench7/s1/TRDY    1'h1
/TestBench7/s1/DE...   1'h1
/TestBench7/s1/BUS     32'h11111110
/TestBench7/s1/mem     32'h00000000 32...
/TestBench7/s1/cache   32'hxxxxxxxx 32...
```

```
sim:/TestBench7/s1/mem @ 2 ns
0 : 32'h00000000 32'h00000000
2 : 32'h00000000 32'h00000000
4 : 32'h00000000 32'h00000000
6 : 32'h00000000 32'h00000000
```

- The cache takes the values inside the memory

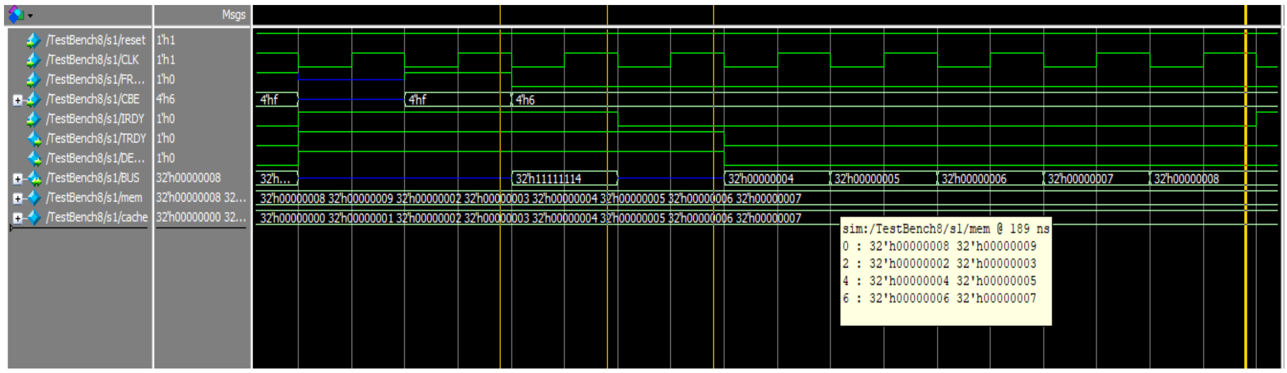
Fig. 8.2



```
/TestBench7/s1/reset   1'h1
/TestBench7/s1/CLK     1'h1
/TestBench7/s1/FR...   1'h1
/TestBench7/s1/CBE     4'hf
/TestBench7/s1/IRDY    1'h0
/TestBench7/s1/TRDY    1'h0
/TestBench7/s1/DE...   1'h0
/TestBench7/s1/BUS     32'h00000009
/TestBench7/s1/mem     32'h00000008 32...
/TestBench7/s1/cache   32'h00000000 32...
```

```
sim:/TestBench7/s1/cache @ 144 n
0 : 32'h00000000 32'h00000001
2 : 32'h00000002 32'h00000003
4 : 32'h00000004 32'h00000005
6 : 32'h00000006 32'h00000007
```

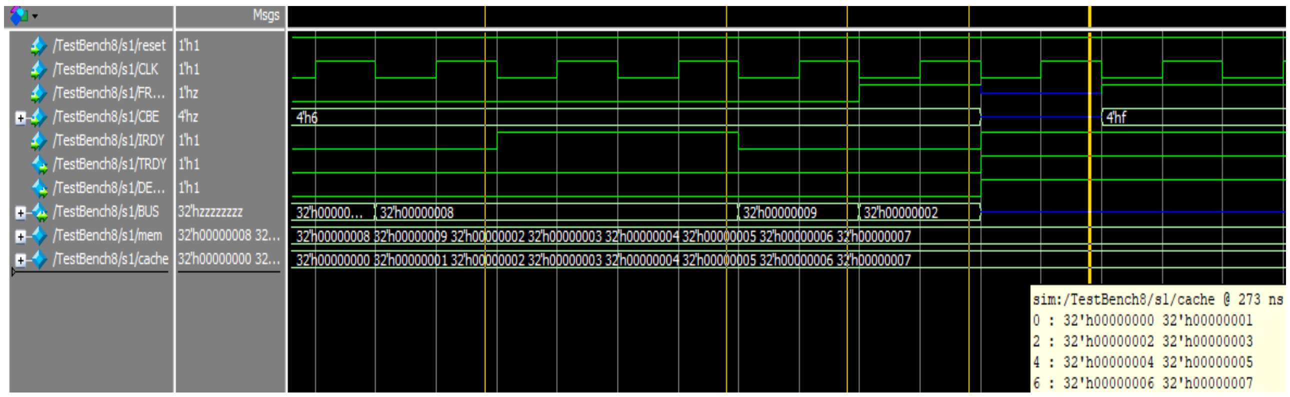
Fig. 8.3



```
/TestBench8/s1/reset   1'h1
/TestBench8/s1/CLK     1'h1
/TestBench8/s1/FR...   1'h0
/TestBench8/s1/CBE     4'h6
/TestBench8/s1/IRDY    1'h0
/TestBench8/s1/TRDY    1'h0
/TestBench8/s1/DE...   1'h0
/TestBench8/s1/BUS     32'h00000008
/TestBench8/s1/mem     32'h00000008 32...
/TestBench8/s1/cache   32'h00000000 32...
```

```
sim:/TestBench8/s1/mem @ 189 ns
0 : 32'h00000008 32'h00000009
2 : 32'h00000002 32'h00000003
4 : 32'h00000004 32'h00000005
6 : 32'h00000006 32'h00000007
```

Fig. 8.4



```
/TestBench8/s1/reset   1'h1
/TestBench8/s1/CLK     1'h1
/TestBench8/s1/FR...   1'hz
/TestBench8/s1/CBE     4'hz
/TestBench8/s1/IRDY    1'h1
/TestBench8/s1/TRDY    1'h1
/TestBench8/s1/DE...   1'h1
/TestBench8/s1/BUS     32'hzzzzzzzz
/TestBench8/s1/mem     32'h00000008 32...
/TestBench8/s1/cache   32'h00000000 32...
```

```
sim:/TestBench8/s1/cache @ 273 ns
0 : 32'h00000000 32'h00000001
2 : 32'h00000002 32'h00000003
4 : 32'h00000004 32'h00000005
6 : 32'h00000006 32'h00000007
```

```
module TestBench8();

////////// DEFAULT  \\\\\\\\

reg reset;
reg CLK,FRAME,IRDY;
reg [3:0] CBE;
wire TRDY,DEVSEL;
wire [31:0] BUS;
parameter add=32'h11111111;
reg cond; // Writing Condition
reg[31:0] buffer;

assign BUS = (cond ) ? buffer : (32'hzzzzzzzz);

initial
begin
$monitor($time, , ,  " %d   %b  %b  %b ", BUS, FRAME, TRDY, DEVSEL);

reset=1;
CLK=0;
IRDY=1;
FRAME=1;
cond=0;
reset=0;
#10
reset=1;
#10
////////// DEFAULT  \\\\\\\\
//code
cond=1;
FRAME=0;
buffer[31:0]= add;
buffer[2:0]= 3'b000;
CBE=4'b0111;

#10
buffer=0;
CBE=4'b1111;
IRDY=0;


#10
buffer=1;
CBE=4'b1111;

#10
buffer=2;
CBE=4'b1111;
#10
buffer=3;
CBE=4'b1111;
#10
buffer=4;
CBE=4'b1111;
#10
buffer=5;
CBE=4'b1111;
#10
buffer=6;
CBE=4'b1111;
#10
buffer=7;
CBE=4'b1111;
#20
buffer=8;
```

```
CBE=4'b1111;
#10

FRAME=1;
buffer=9;
CBE=4'b1111;
#10
IRDY=1;
FRAME=1'bz;
buffer=32'hzzzzzzzz;
CBE=4'bzzzz;

#10
FRAME=1;
CBE=4'b1111;
#10





//read
cond=1;
CBE=4'b0110;
buffer[31:0]= add;
buffer[2:0]= 3'b100;
FRAME=0;
#10
cond=0;
IRDY=0;
#10
#50
IRDY=1;
#20
IRDY=0;
#10

FRAME=1;
#10
IRDY=1;
FRAME=1'bz;
buffer=32'hzzzzzzzz;
CBE=4'bzzzz;

#10
FRAME=1;
CBE=4'b1111;
end

always
begin
#5
CLK= ~ CLK;
end


slave s1 ( reset,CLK, TRDY,IRDY, BUS,CBE, FRAME, DEVSEL);

endmodule
```

4. Implementation Enhancements

- Support the case when the initiator is not ready the transaction wait till the IRDY is DE-asserted.

- Support the case when writing more than 8 times:
    - TRDY is asserted for 1 cycle.
    - Copy the data from the main memory to another temporary memory.
    - Then TRDY is DE-asserted and the transaction continues.