FPGA implementation of Modified Booth's Algorithm for Multiplication

---

The aim here is to take you through the design and implementation steps of FPGA implementation for 8-bit binary multiplier. The algorithm used here uses Modified Booth's algorithm which approximately twice as fast as Booth's algorithm. You can compare this with 4-bit array multiplier design shown in this OLC by extending that to 8-bit. Refer to HDL coding issues given in the text Digital Principles and Applications, 6e by Leach, Malvino and Saha, TMH, 2006.

Algorithm:

Modified Radix-4 Booth's Algorithm is made use of for fast multiplication. The salient features of this algorithm are:

- Only *n*/2 clock cycles are needed for n-bit multiplication as compared to n clock cycles in Booth's algorithm.
- Isolated 0/1 are handled efficiently.
- For even *n*, the two's complement multipliers are handled automatically whereas for odd *n* an extension of sign bit is required.

Procedure:

For all odd values of *i* where *i* ranges from 1 to *n*-1 for *n*-bit multiplication (assuming *n* is even), the bits of the multiplicand are recoded using the formula

$$y_i = x_{i-1} + x_{i-2} - 2x_i$$

Then multiplication is done in normal way with the y_i that have been calculated. The following example illustrated the whole procedure:

A			01	00	01			17
X	$\times$		11	01	11			-9
Y			$0\bar{1}$	10	$0\bar{1}$			recoded multiplier
			$-A$	$+2A$	$-A$			operation
Add $-A$	+		10	11	11			
2-bit Shift		1	11	10	11	11		
Add $2A$	+	0	10	00	10			
			01	11	01	11		
2-bit Shift			00	01	11	01	11	
Add $-A$	+		10	11	11			
			11	01	10	01	11	-153

Input/Output Specification and Verilog Code:

- Input ranges from -128 to +127
- Negative numbers are represented using 2's-complement notation.
- Again, if the output is negative, then it is obtained in 2's-complement form.

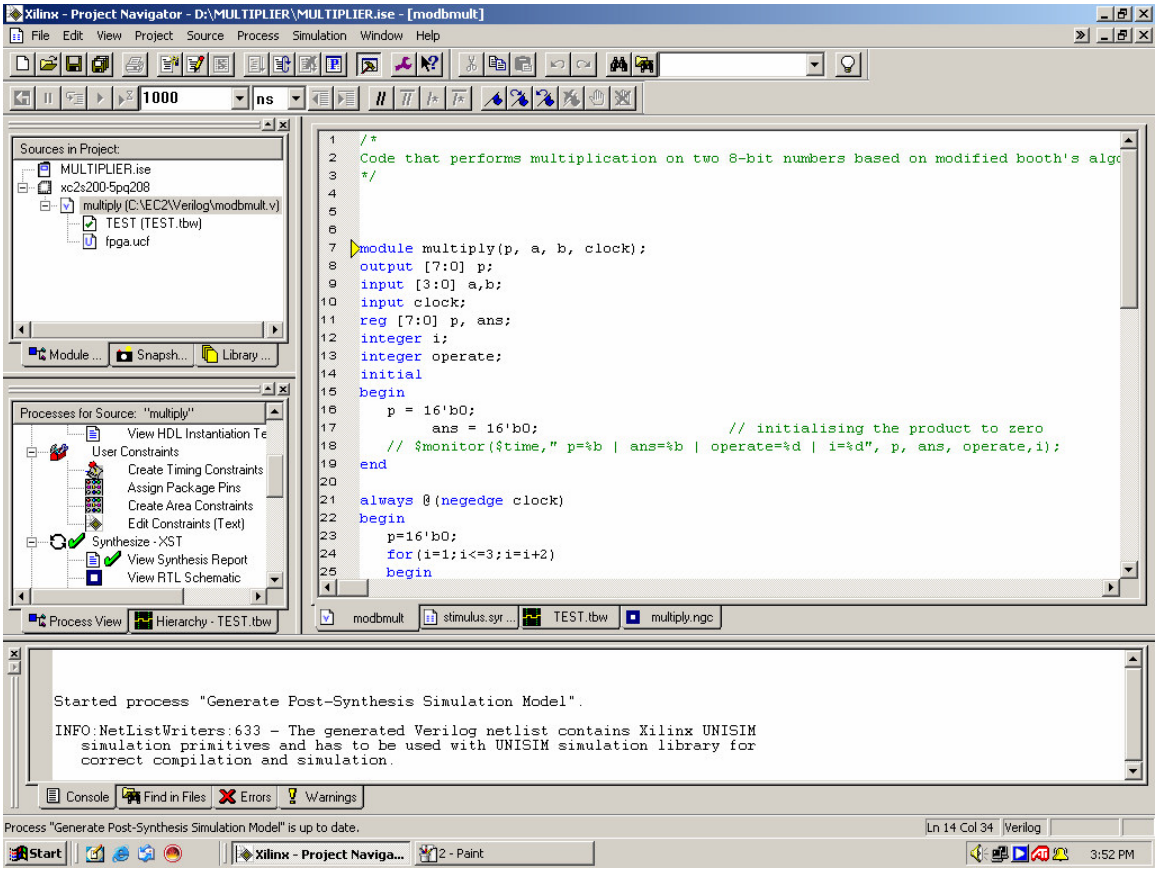
```
module multiply(p, a, b, clock);
output [15:0] p;
input [7:0] a,b;
input clock;
reg [15:0] p, ans;
integer i;
integer operate;
initial
begin
          p = 16'b0;                    // initialising the product to zero
          ans = 16'b0;
end
always @(negedge clock)
begin
        p=16'b0;
        for(i=1;i<=7;i=i+2)                  //loop for multiplication
        begin
                if(i==1)
                         operate = b[0]-b[1]-b[1];                    //b[-1] = 0  (assumed)
                else
                         operate = b[i-1] + b[i-2] - b[i] - b[i];
                case (operate)
                        1:                                              //multiplication by 1
                        begin
                                 ans = a;
                                 ans = ans<<(i-1);
                                 p = p + ans;
                        end
                        2:
                        begin
                                ans = a<<1;                  //multiplication by 2
                                ans = ans<<(i-1);
                                p = p + ans;

                        end
                        -1:                                             //multiplication by -1
                        begin
                                ans = ~a + 1;
                                ans = ans<<(i-1);
                                p = p + ans;
                        end
                        -2:                                             //multiplication by -2
                        begin
                                ans = a<<1;
                                ans = ~ans + 1;
                                ans = ans<<(i-1);
                                p = p + ans;
                        end
                endcase
        end
end
endmodule
```

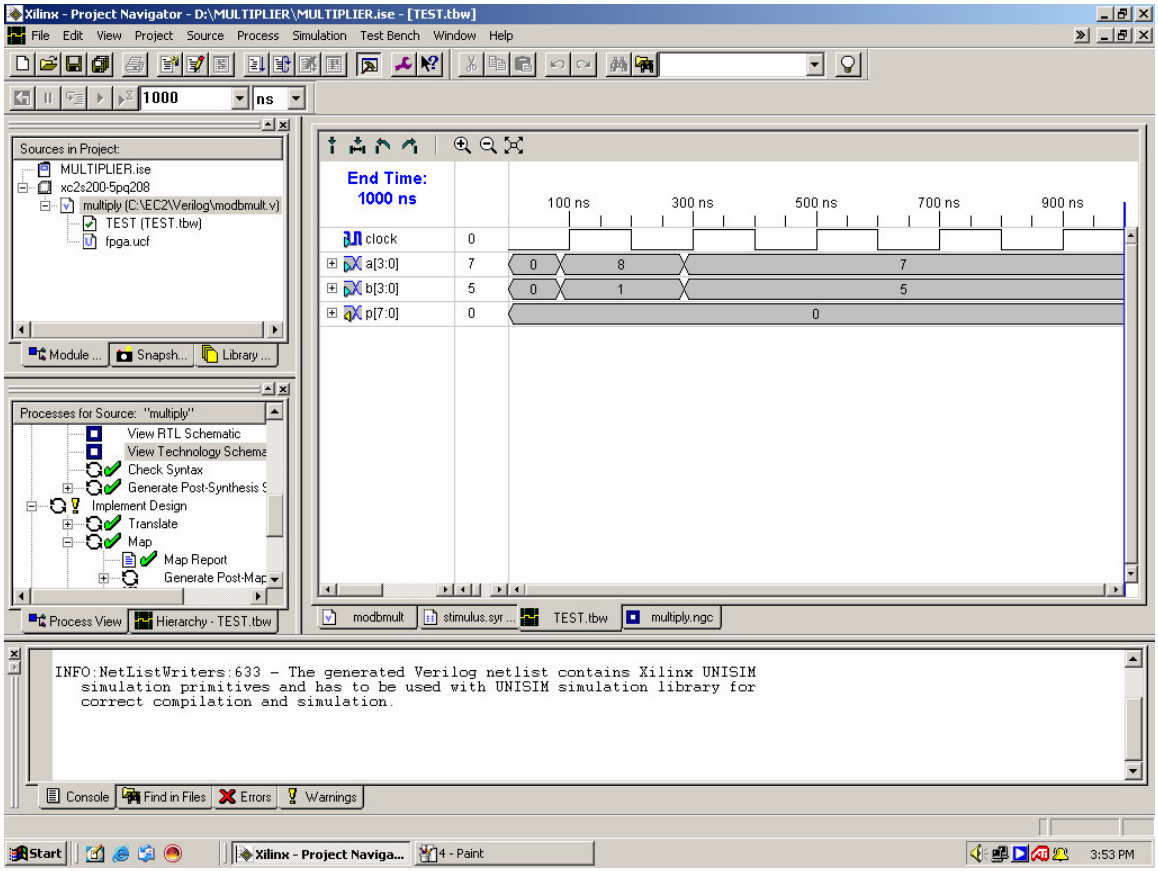
Simulation and Implementation in FPGA:

ModelSim is used for simulation of the code and Xilinx is used for the implementation of the code in FPGA.
Here are some screenshots at various stages of the implementation procedure:

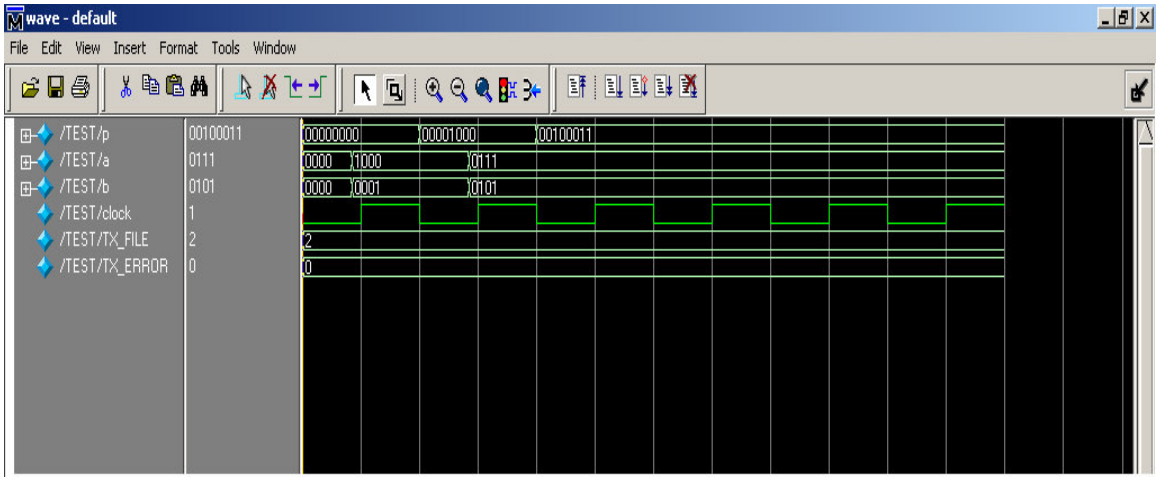
The Code:



Input Assignment for Simulation:

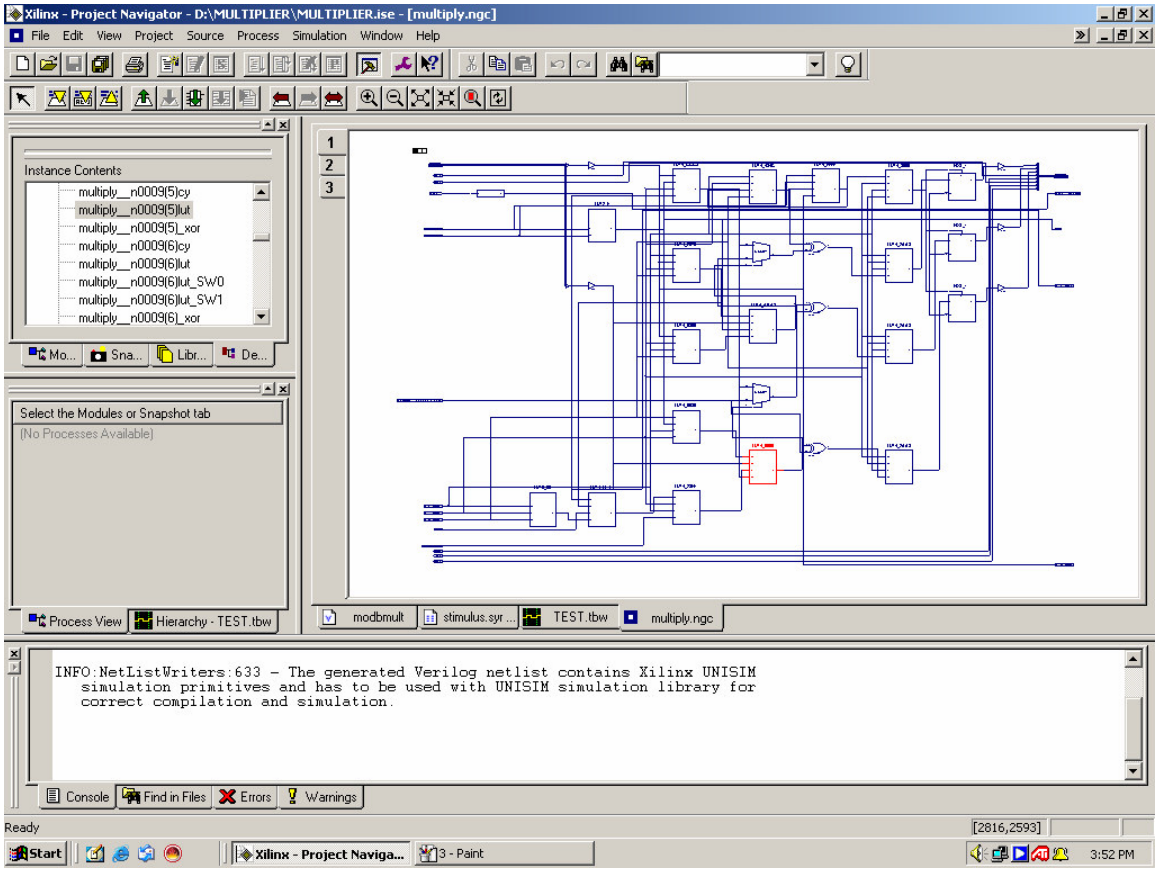


Output Waveform (simulated using ModelSim):

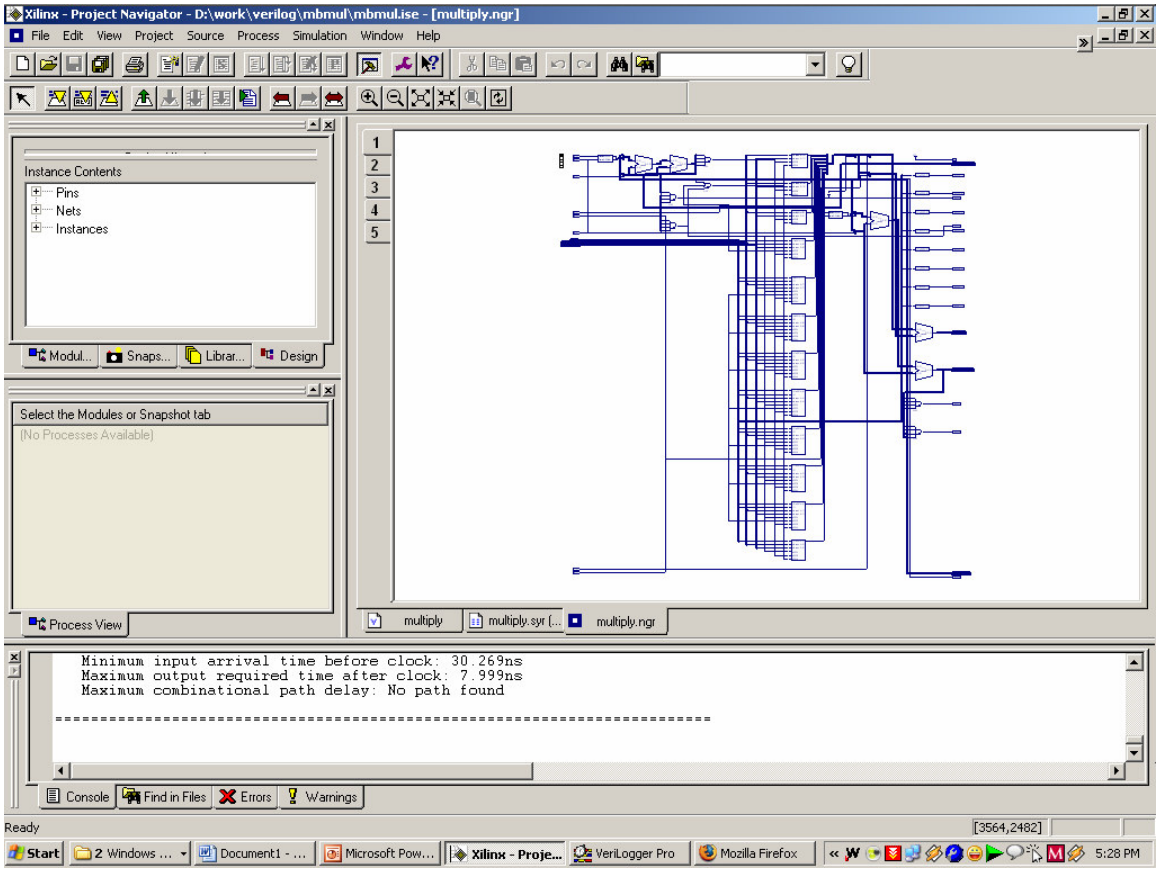


Synthesized Circuit:

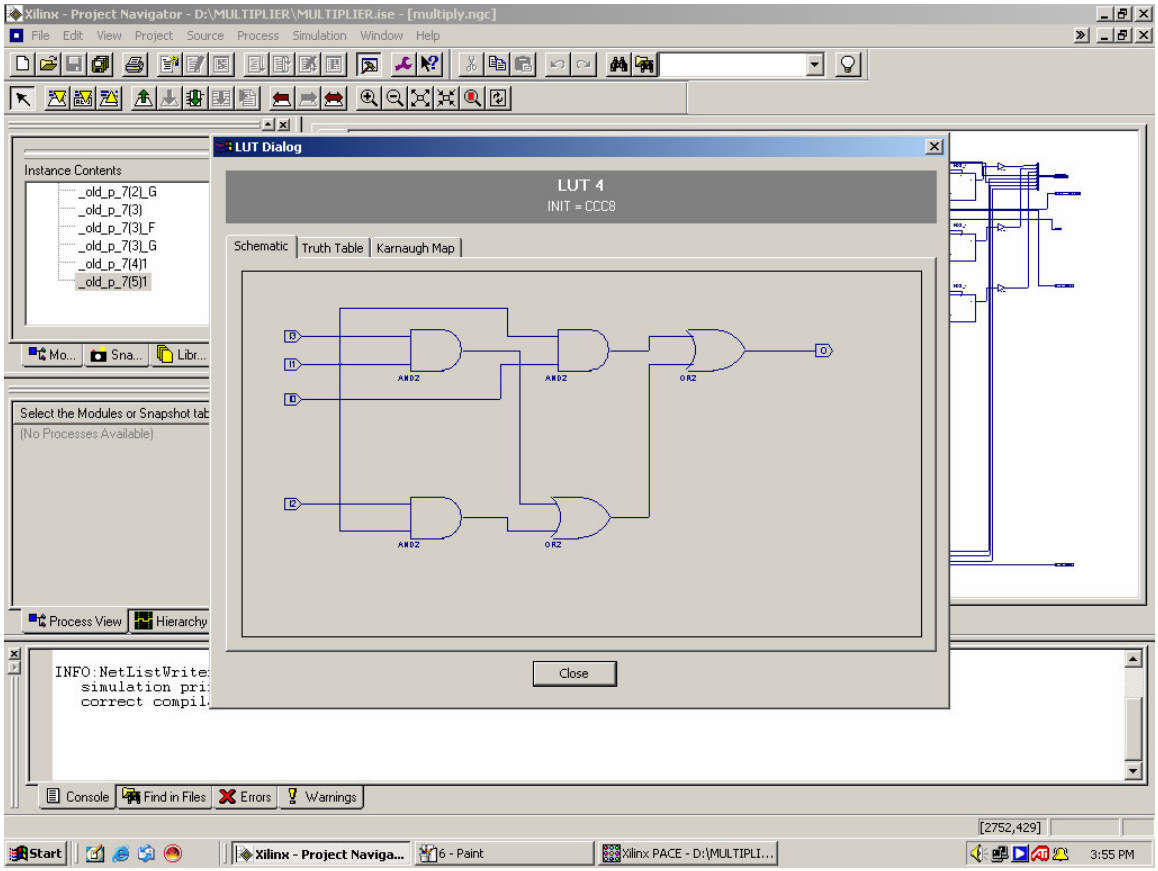
Technology Schema – Hardware Implementation



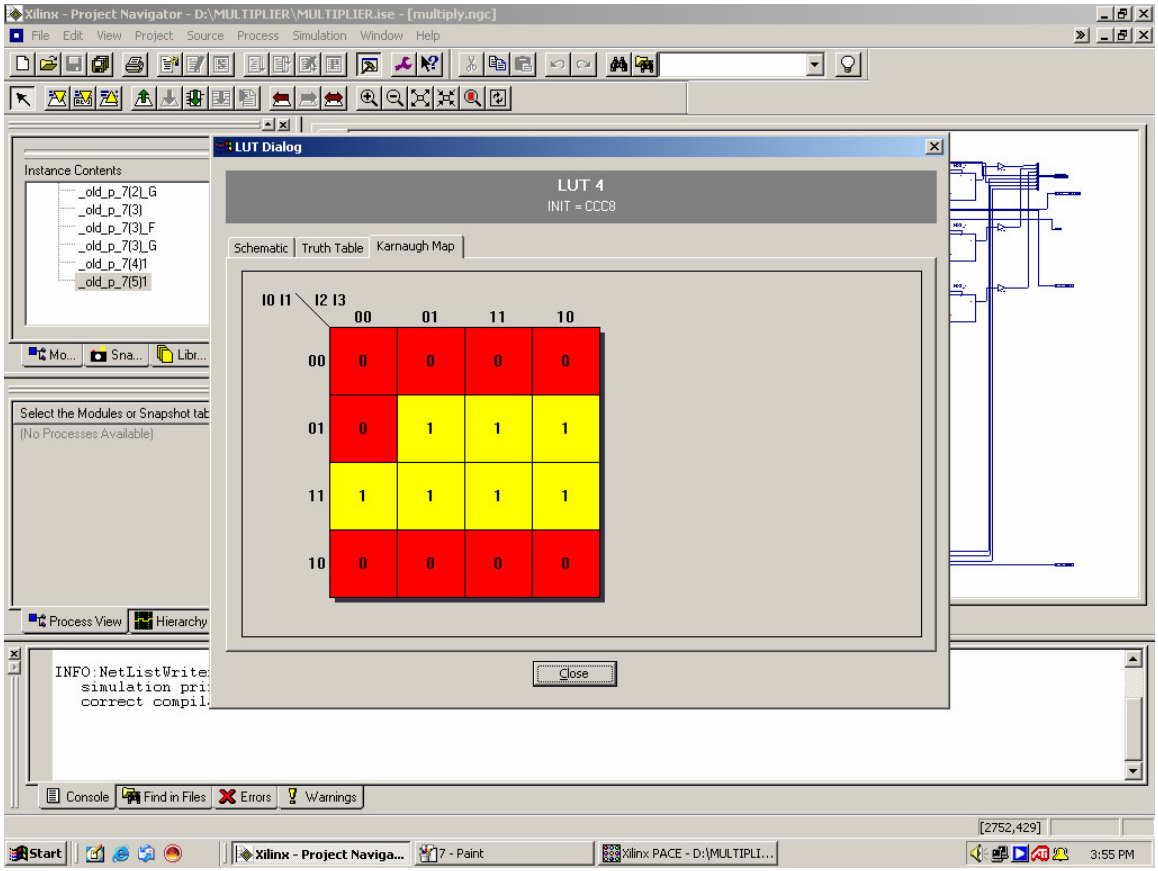
RTL Implementation:



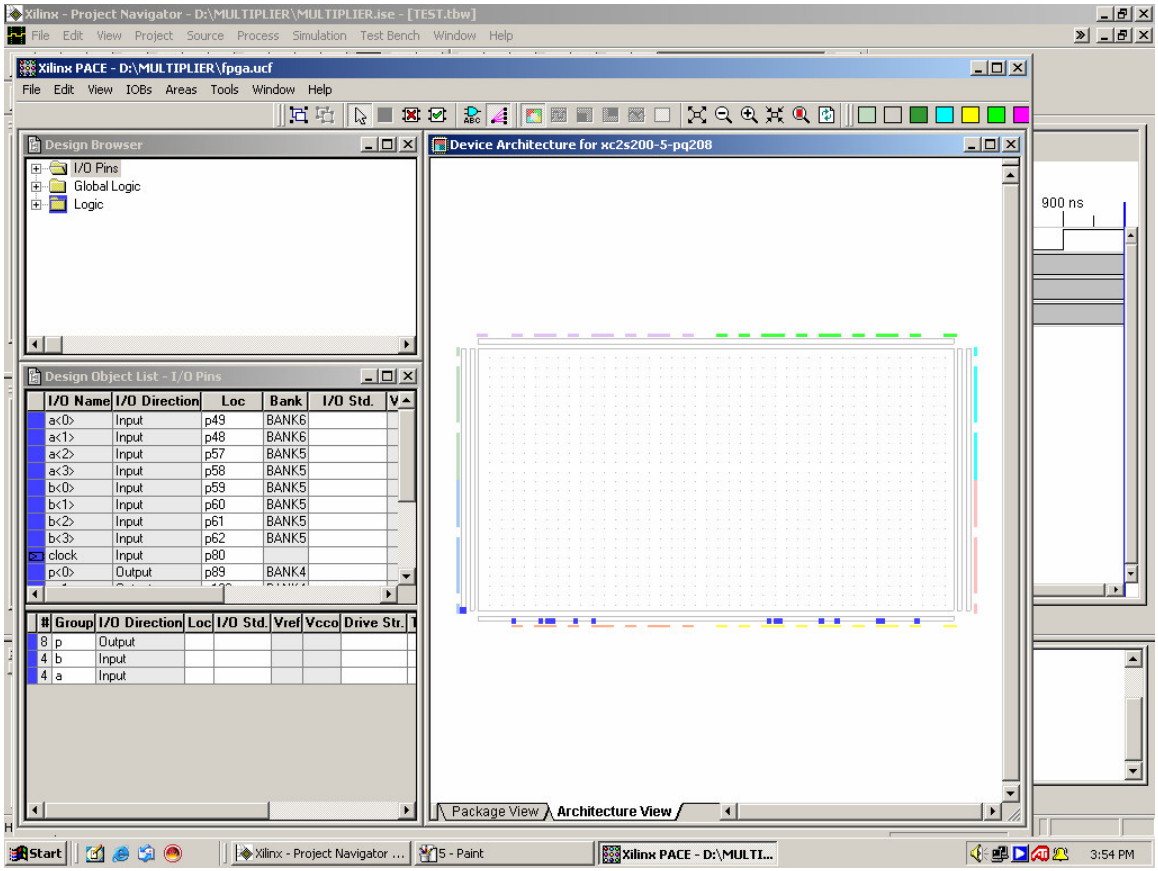
Gate level implementation of one of the blocks:



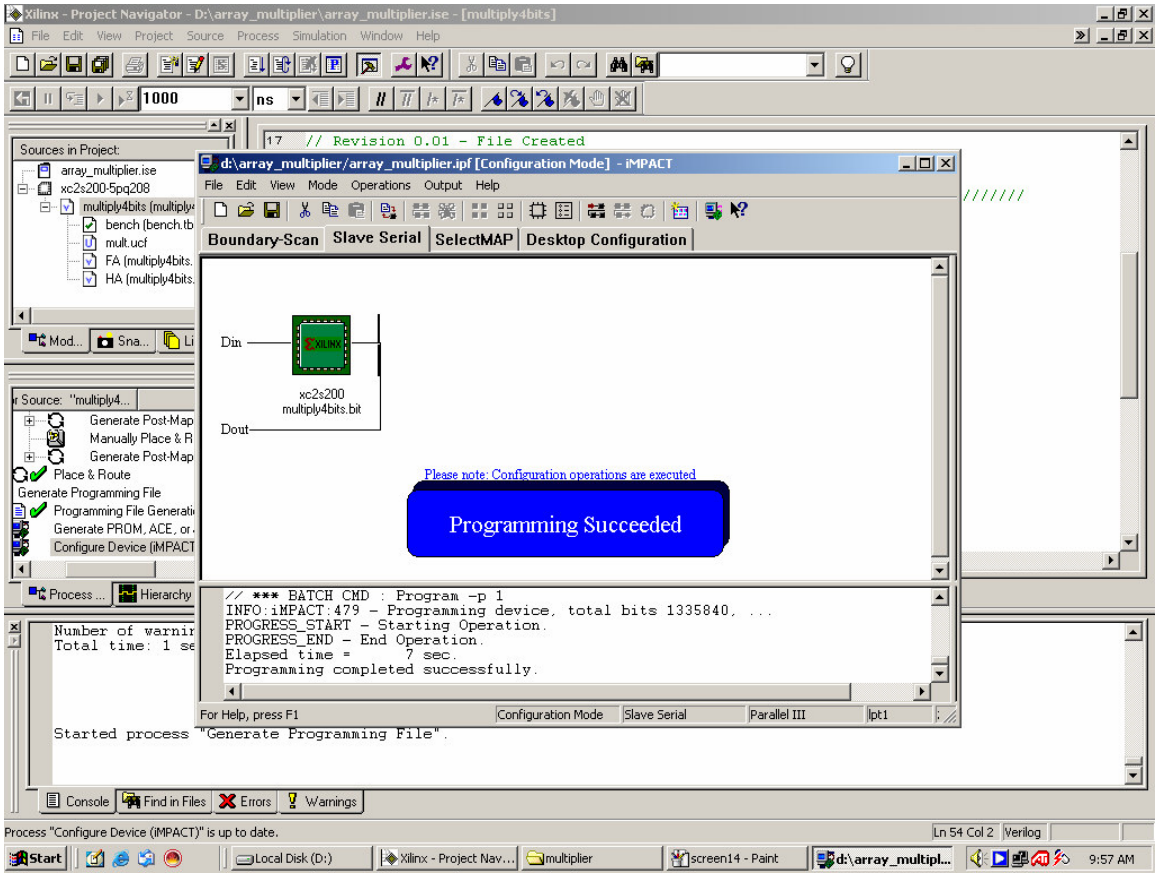
Karnaugh map for the above block:



Layout of FPGA and I/O port Assignments:



Programming the FPGA:



Conclusion:

The main advantage of using FPGA is that it gives high level of flexibility to the user to rapidly construct and test any hardware. FPGAs consist of a lot of gates out of which those that are needed get programmed by the application of suitable field. For example, the Spartan-2 FPGAs consists of 200,000 gates. Thus the number of gates used depends upon the hardware to be implemented and in most of the cases all the gates are not used. Therefore FPGAs are used for testing purposes only.

Hardware Description Languages (HDL) are useful to simulate a digital design before implementing it in the hardware. These languages provide the flexibility to describe the circuit in terms of its behaviour (behavioural modeling). The Compiler implements the code into the hardware.