CS-288: LOGIC DESIGN LAB

Documentation for Assignment-1 CS-288

Submitted By:

Astha Agarwal (110050018)

Ashish Sonone (110050022)

Anand Soni (110050037)

Mridul Jain (110040083)

Vikash Challa (110050077)

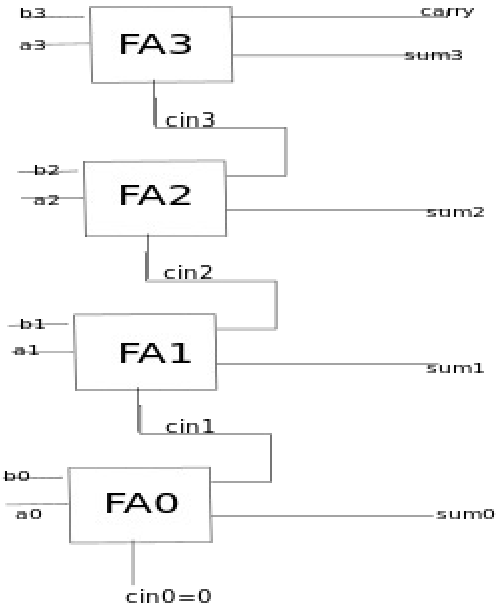
Adder

We have used four full-adder entities to implement the four bit adder logic.

Inputs- Two 4-bit STD_LOGIC_VECTORs a and *b*

Outputs- One 4-bit STD_LOGIC_VECTOR *sum,*

One 1-bit STD_LOGIC carry.



4-bit adder

We start adding the LSB bits with initial carry zero using full adder entity. The resulting carry is forwarded to the next set of input bits. The intermediate sums are assigned to the corresponding sum-vector('sum') bits. And the final carry goes to the carry bit.

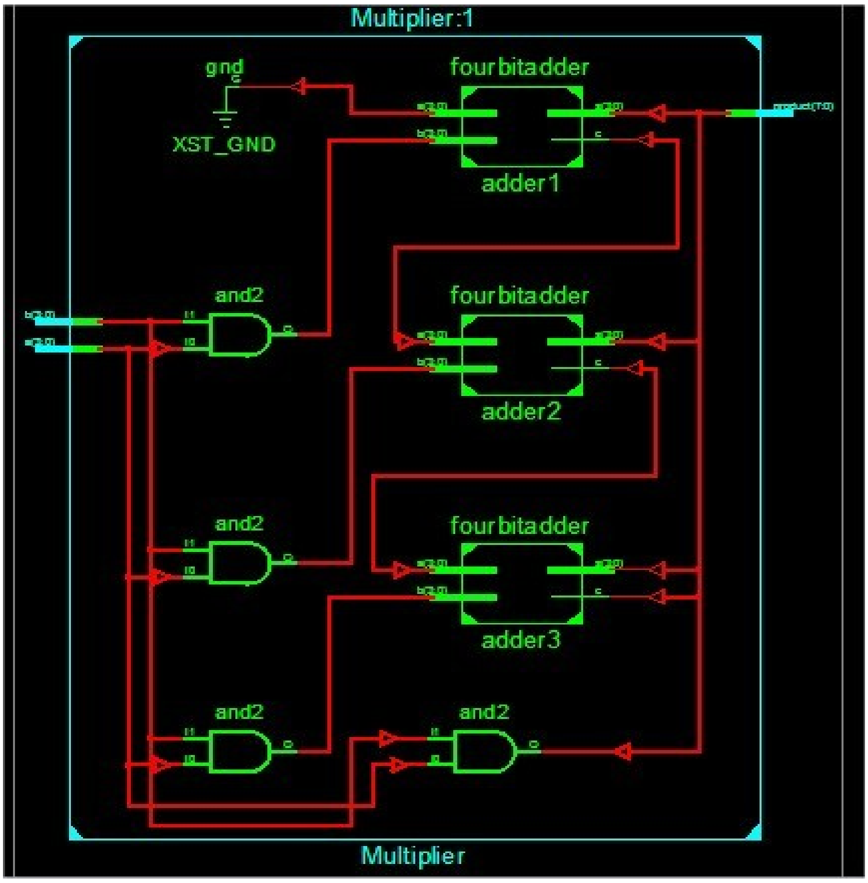
Multiplier

We have used four four bit adder entities to implement the four bit multiplier logic.

Inputs- Two 4-bit STD_LOGIC_VECTORs a and *b*

Outputs- One 8-bit STD_LOGIC_VECTOR product,

We start multiplying ('ANDing') the bits from 'b' with each bit of 'a' and store the corresponding values as 4-bit STD_LOGIC_VECTOR signals (Partial multiplication). Next, we feed the two signals as inputs to the first adder and store the corresponding sum in another 4-bit STD_LOGIC_VECTOR signal. After multiplying Now, we 'AND' the input vector 'b' bitwise with the third bit of 'a', store the corresponding values and feed them to the second adder with the output from the first adder. We continue like this, generating intermediate signals and then assign the output 'product' values from the three adder outputs. The circuit diagram below explains the process further.



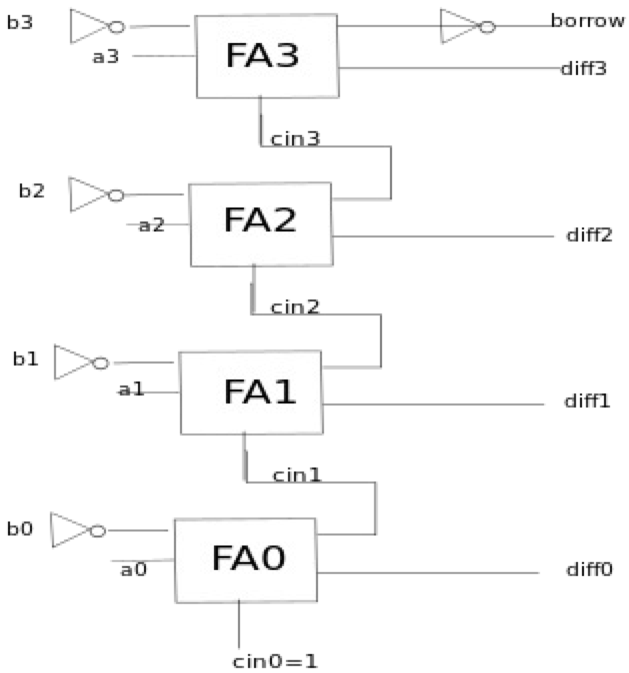
Subtractor

We have used four full-adder entities and NOT gates to implement the four bit adder logic.

Inputs- Two 4-bit STD_LOGIC_VECTORs a and *b*

Outputs- One 4-bit STD_LOGIC_VECTOR diff, One 1-bit STD_LOGIC borr.

We start subtarcting the LSB bits with initial borrow ('borr') zero using full adder entity. We implemented the subtractor using complement method. 'b' is complemented and added to 'a' with initial carry 1. The resulting carry is forwarded to the next set of input bits. The intermediate sums are assigned to the corresponding difference-vector('diff') bits. And the final carry is complemented to give the borrow bit.



4-bit subtractor

Bit Shift

Inputs- 4-bit STD_LOGIC_VECTOR In1,

Integer In2, 1-bit STD_LOGIC dir.

Outputs- 4-bit STD_LOGIC_VECTOR *out1.*

The architecture is sequential, inside a process with dependencies *In1* and dir*.* First, In1 is stored in a buffer variable (of type STD_LOGIC_VECTOR). Then, inside a for loop, we shift bits 'In2' times according to the value of 'dir'. If *dir* is 1, then we shift left. Else if '*dir*' is 0, then we shift right. Finally, out1 is assigned the value of the updated buffer.

For example: Shifting left 1-bit '1011', the output is '0110'. The LSB is assigned zero.

Bit Rotator

Inputs- 4-bit STD_LOGIC_VECTOR *In1*,

Integer In2, 1-bit STD_LOGIC dir.

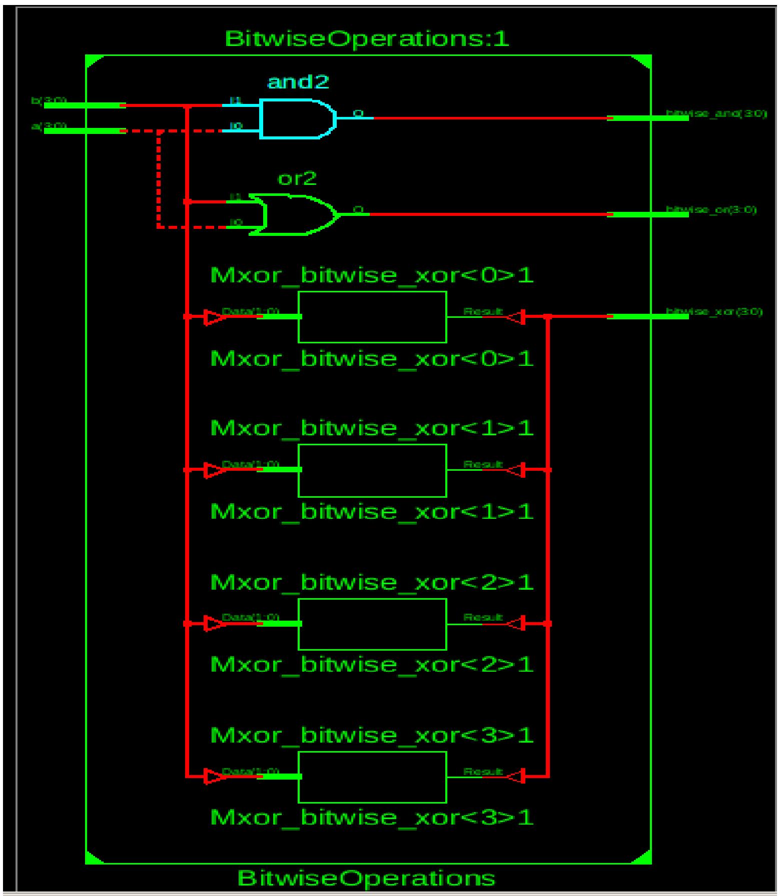
Outputs- 4-bit STD_LOGIC_VECTOR *out1.*

The architecture is sequential, inside a process with dependencies *In1* and dir*. First, In1 is stored in a buffer variable (of type STD_LOGIC_VECTOR). Then, inside a for loop, we rotate bits* '*In2*' *times according to the value of* '*dir*'*.* If 'dir' is 1, we rotate left else, we rotate right. Also, we have used a 'temp' variable of type STD_LOGIC to temporarily store MSB or LSB depending on the value of 'dir'.

Bitwise operations

INPUT- Two 4-bit STD_LOGIC_VECTORs *a* and *b*

OUTPUT- Three 4-bit STD_LOGIC_VECTOR *bitwise_and, bitwise_or, bitwise_xor.*

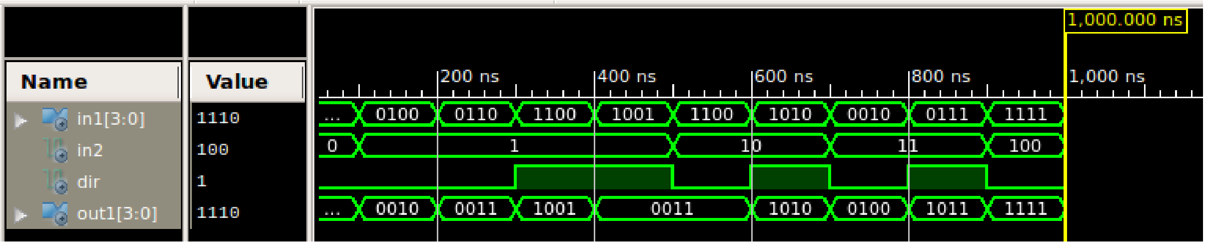


In the architecture, basically we assign to each index of the *output* vector, the result of bitwise AND applied to both the corresponding input bits of a and *b*, that is, for i=0,1,2,3 we assign *bitwise_and*(i) <= a(i) AND *b*(i).

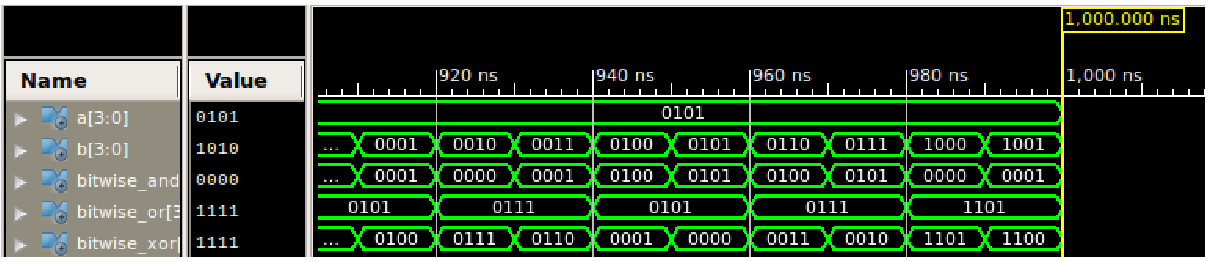
Similarly, for other two bitwise operations.

Results Captured

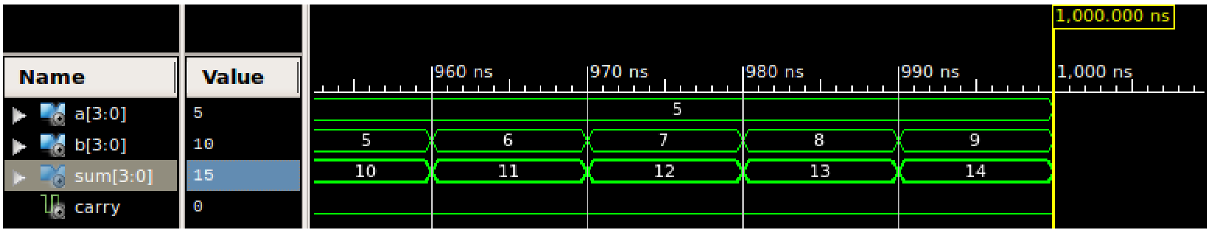
Bit Rotation:



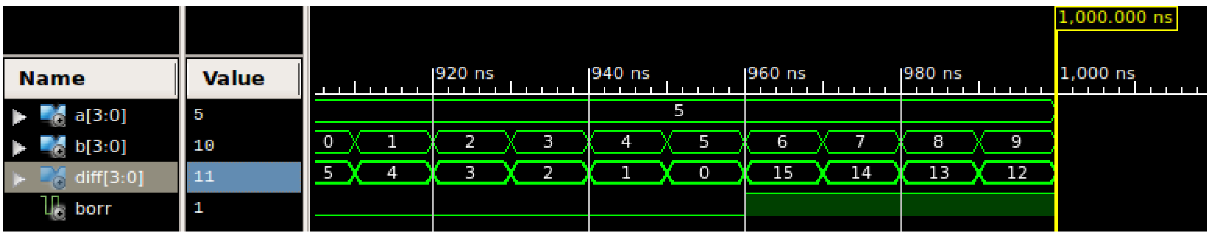
Bitwise:



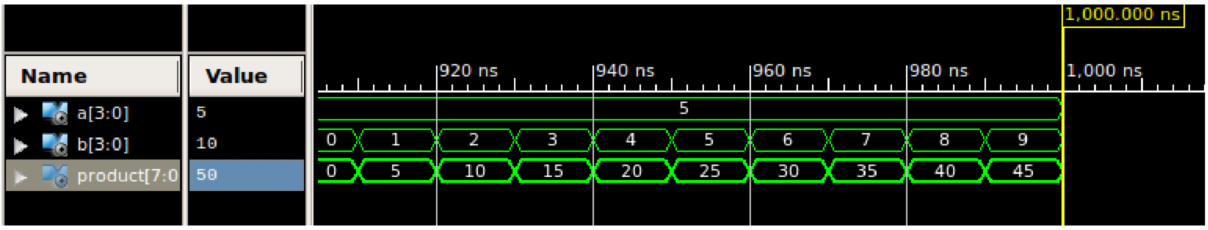
Four bit Adder:



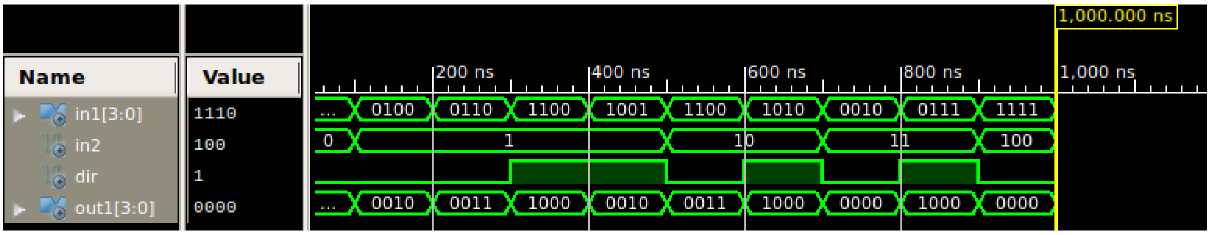
Four bit Subtractor:



Multiplier:



Bit Shift:



The above screenshots clearly show the results of all the operations as expected. Thus, we have successfully implemented the calculator.