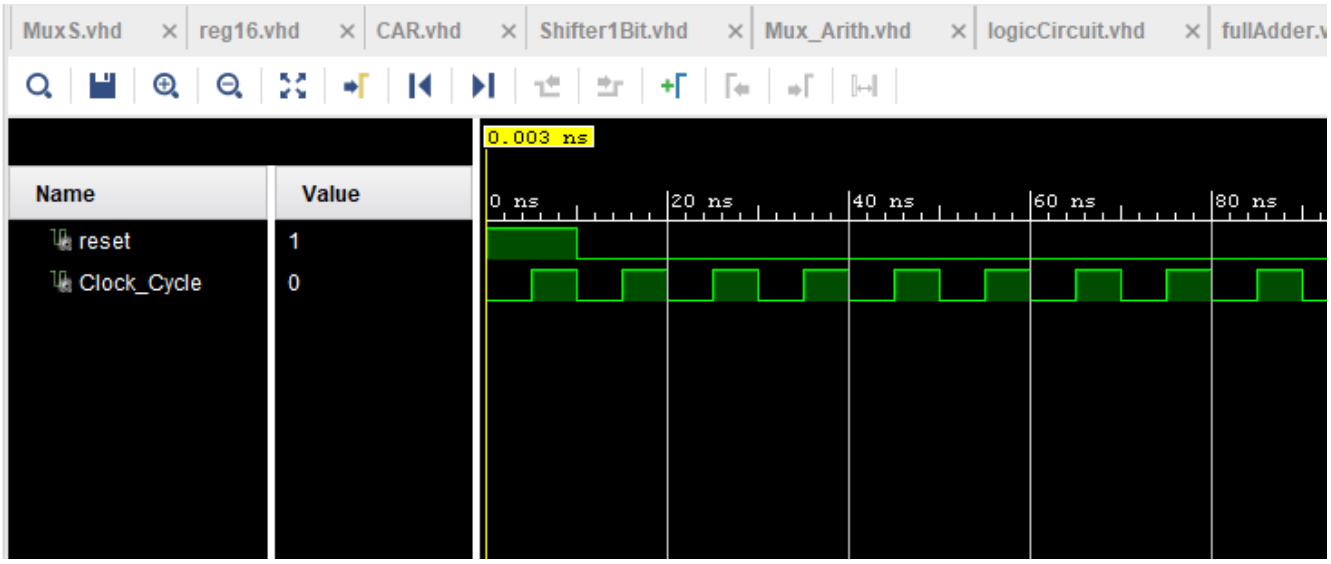
CS2022 Computer Architecture

Project 2 – Micro coded Instruction Set Processor

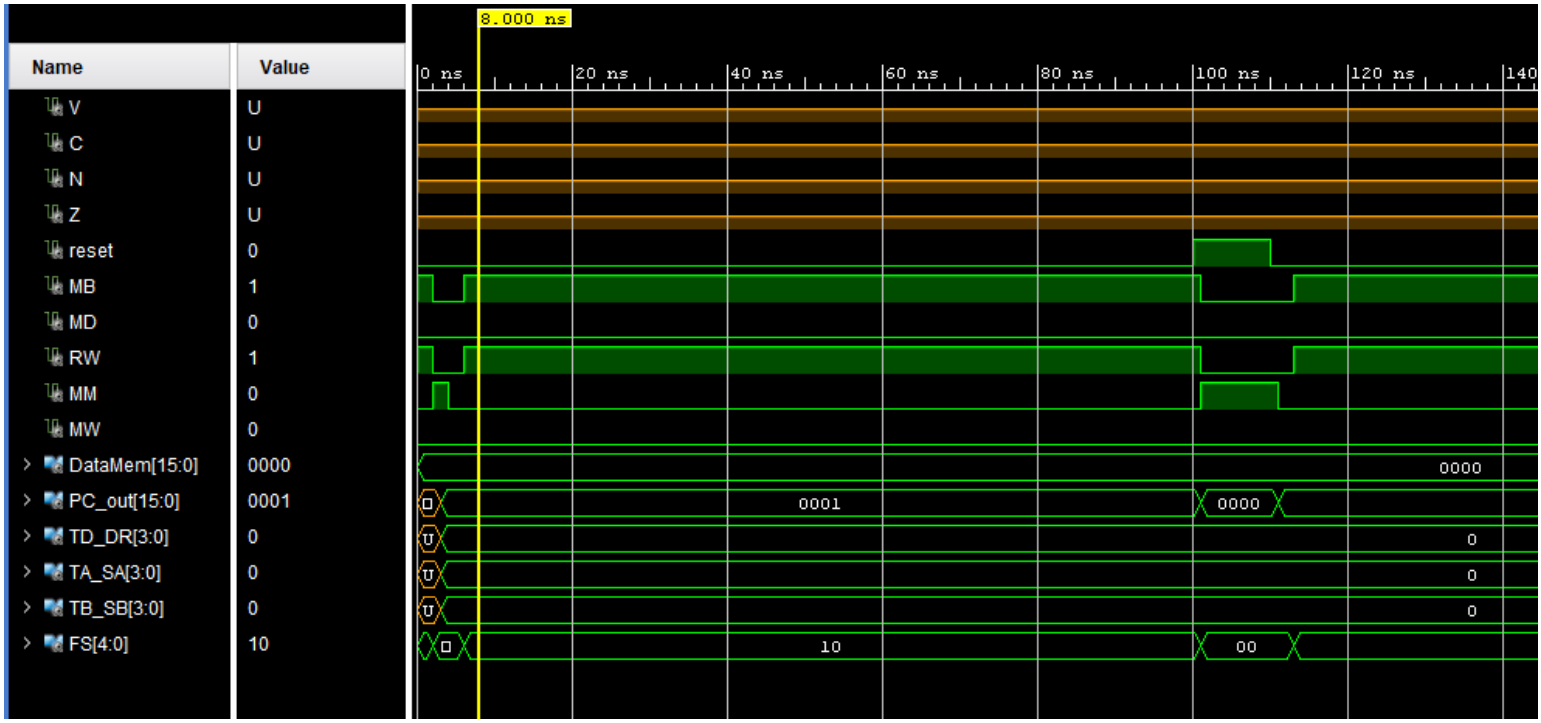
Name: Caleb Teo
Student Number: 15324741
Data: 05/04/2018

1 MICROPROCESSOR – MICROPROCESSOR.VHD



The Microprocessor has a reset and Clock Cycle which set the clock for the registers to load in new instructions. The instructions and their codes are listed below in the ControlMem and Mem section of the report.

2 MICROPROGRAMMED CONTROL - MICROPROGRAMMEDCONTROL.VHD



The DataMem was set at “0000” which means that the control word at control memory 0 was selected. That control word was the following

Code		Hex: C020304				Binary: 1100 0000 0010 0000 0011 0000 0100								
NA	MS	MC	IL	PI	PL	TD	TA	TB	MB	FS	MD	RW	MM	MW
11000000	001	0	0	0	0	0	0	0	1	10000	0	1	0	0

From 7ns we can check our values with the above table. We can see that the microprogrammed control works correctly. We have the MB and RW set to ‘1’ and MD, MM and MW set to ‘0’. We can also see that the PC value is ‘1’ as the CAR would set a default value of “C10C002” which would reset the PC value to “0000” and then increment by 1 I.e. “0001”. We can also see that FS is set to “10” which is correct to the table. We can also note the correct values of TD_DR, TA_SA, and TB_SB as they have the values ‘0’. We can note that the reset works correctly as at 100ns the reset is set ‘1’ and the control words are reset to zero.

3 CONTROL MEMORY – CONTROLMEM.VHD

From the above image, we can see that the Control Memory use IN_CAR and selects the memory address at “00”, “01”, “09”, “0C” and “50”. We can then look at the memory and check the correct control word.

```
begin
    memory_m : process(IN_CAR)
        variable control_mem : mem_array := (
            --0
            X"C020304", --0         --Stores value in Reg 0 = 0;
            X"C020304", --1         --Stores value in Reg 1 = 1;
            X"C020304", --2         --Stores value in Reg 2 = 2;
            X"C020304", --3         --Stores value in Reg 3 = 3;
            X"C020304", --4         --Stores value in Reg 4 = 4;
            X"C020304", --5         --Stores value in Reg 5 = 5;
            X"C020304", --6         --Stores value in Reg 6 = 6;
            X"C020304", --7         --Stores value in Reg 7 = 7;
            X"C020224", --8         --ADI R[dr] = R[SA] + zf[SB]
            X"C000014", --9         --INC
            X"C0000E4", --A         --NOT
            X"C000024", --B         --ADD
            X"0D0D00E", --C         --LDR1 R8 = M[R[SA]]
            X"C00000C", --D         --LDR2 R[SB] = R8
            X"0000000", --E
            X"0000000", --F
```

For “00” and “01” we can see that they are same and have following set bits:

Code		Hex: C020304				Binary: 1100 0000 0010 0000 0011 0000 0100								
NA	MS	MC	IL	PI	PL	TD	TA	TB	MB	FS	MD	RW	MM	MW
11000000	001	0	0	0	0	0	0	0	1	10000	0	1	0	0

Which compared to the image is correct to the table above. We can see the same for the memory at “09” and “0C”:

Code		Hex: C000014				Binary: 1100 0000 0000 0000 0000 0001 0100								
NA	MS	MC	IL	PI	PL	TD	TA	TB	MB	FS	MD	RW	MM	MW
11000000	000	0	0	0	0	0	0	0	0	00001	0	1	0	0
Code		Hex: 0D0D00E				Binary: 1100 0000 0010 0000 0011 0000 0100								
NA	MS	MC	IL	PI	PL	TD	TA	TB	MB	FS	MD	RW	MM	MW
00001101	000	0	1	1	0	1	0	0	0	00000	1	1	1	0

For the memory at address “50” is an empty address which means that the values of the control word are “0000000”.

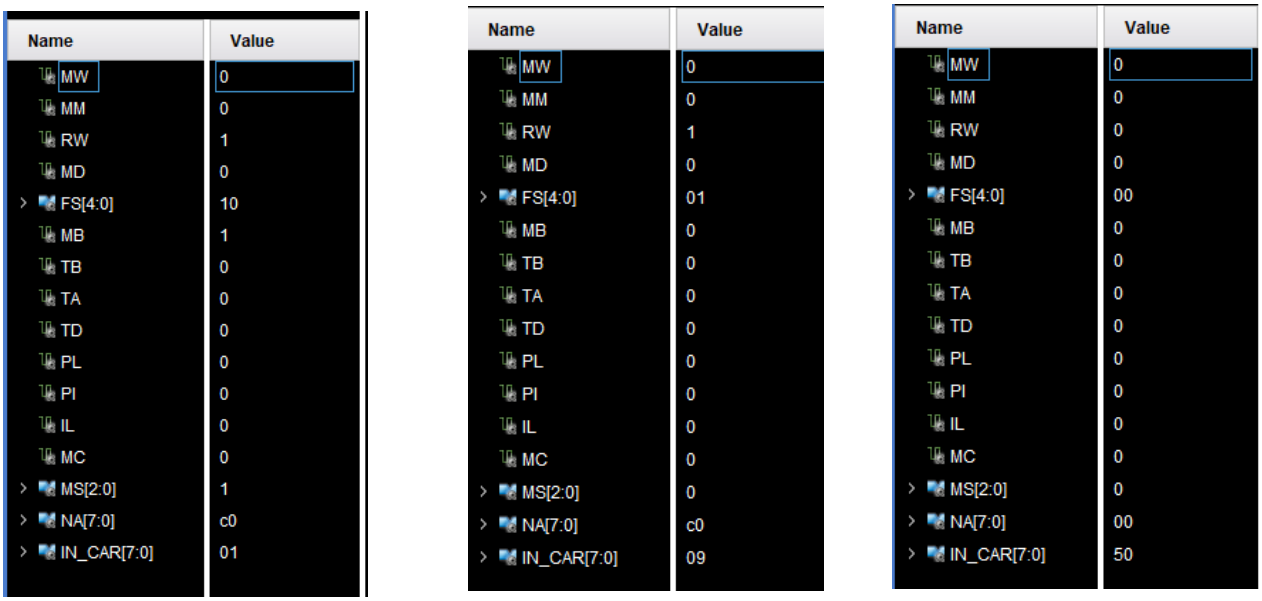
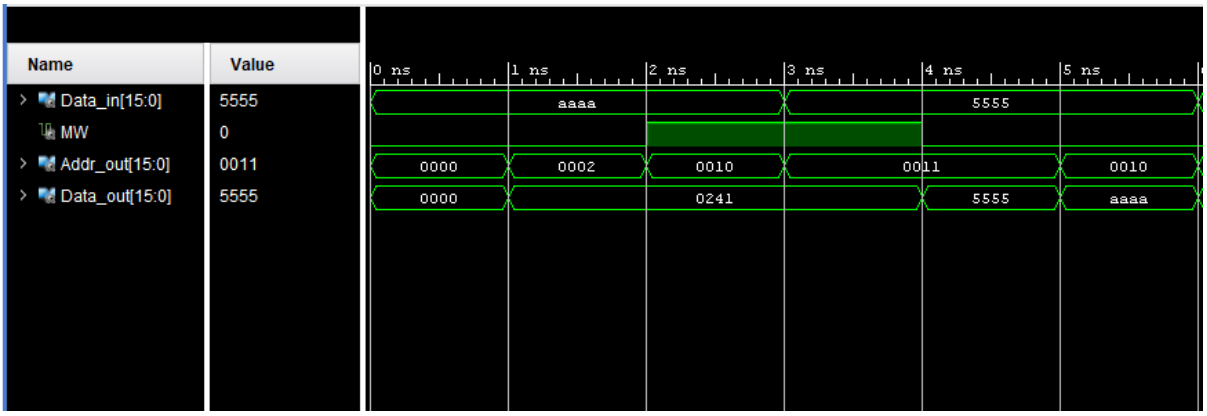


Figure 1: (From left to right) Values of "01", Values of "09" and Values of "50"

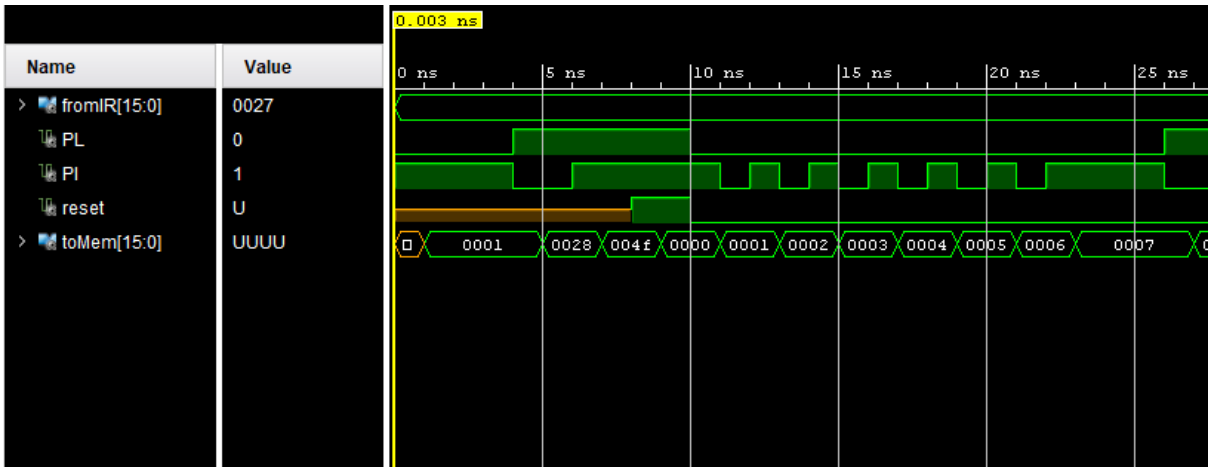
4 MEMORY – MEM.VHD



The image of the memory shows that the Addr_out is being selected as 0000 at first which maps to “0000” displayed in Data_out. The test bench then selects the address “0002” which is the memory is storing a value of “0241”. The MW is then set to ‘1’ which will then write the value from Data_in, “aaaa” to the Addr_out, “0010”. The test bench also stores “5555” in address “0011”. We can then check if the values “aaaa” and “5555” have been stored properly by setting MW = ‘0’. We can see that at 4ns and 5ns the values stored in “0010” and “0011” were stored correctly by inspecting the output in Data_out.

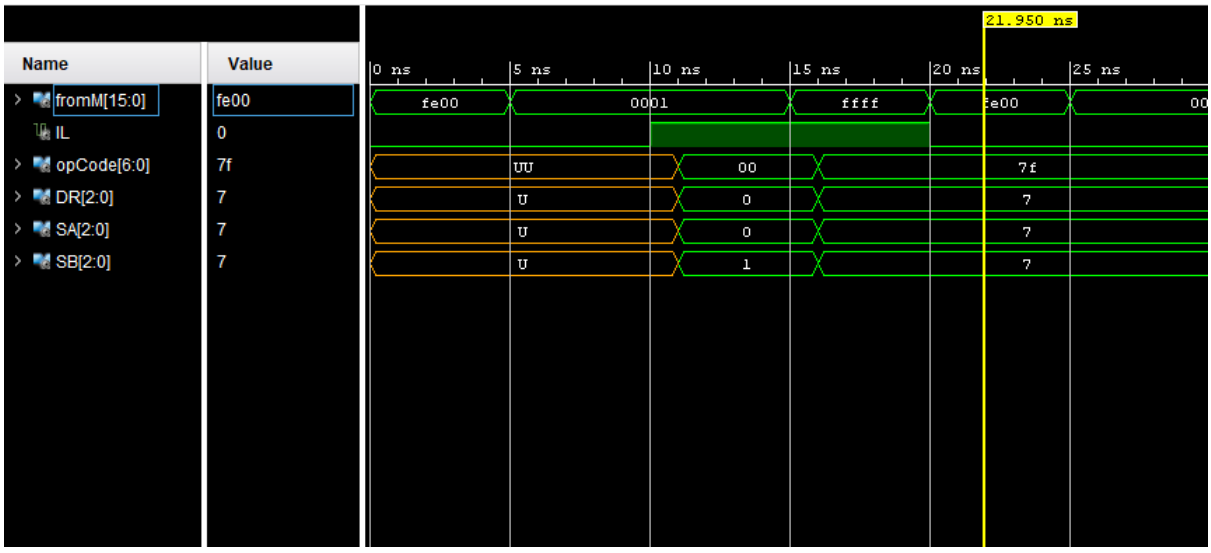
```
architecture Behavioral of Mem is
    type mem_array is array(0 to 511) of std_logic_vector (15 downto 0);
begin
    memory_m : process(Addr_out, MW)
        variable Mem : mem_array := (
            --00
            X"0000", X"0000", X"0241", X"0482",
            X"06C3", X"0904", X"0B45", X"0D86",
            X"0FC7", X"0000", X"0000", X"0000",
            X"0000", X"0000", X"0000", X"0000",
```

5 PROGRAM COUNTER – PC.VHD



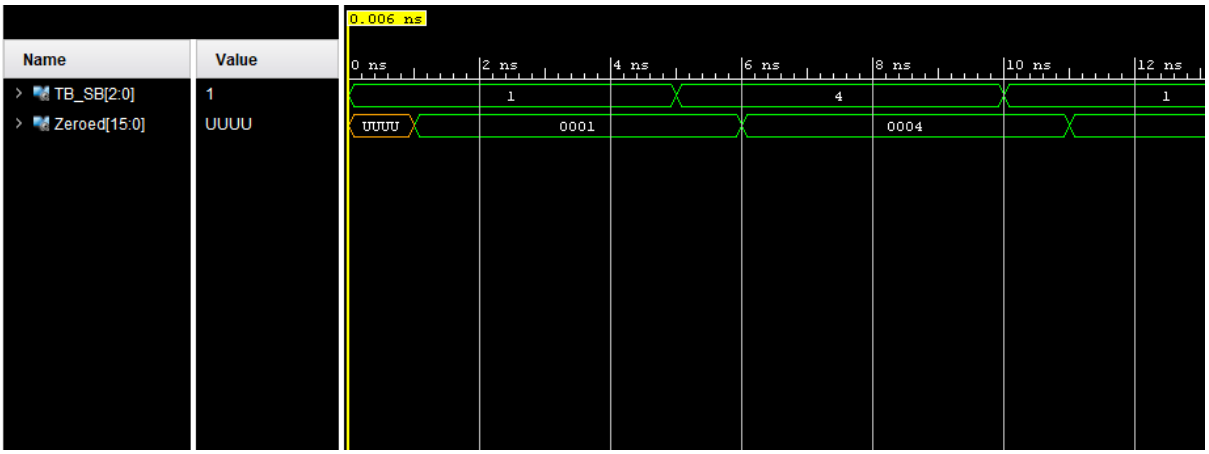
From the above image, we can see that function of the Program counter. When the value of PI = '1' then the PC increments from its original value "0000" by 1 i.e. to "0001". When the PL is set to '1' then the current value (0001) will be added to fromIR (0027) which gives a value of 0028 at 5ns. When we have PI = '1' and PL = '1' then we load only which gives us a value of "004F" (0027 + 0028). We can also see if we cycle the PI then we can increment the values e.g. "0000" -> "0007". The PC can also be reset when it is set to '1' (example at 10ns).

6 INSTRUCTION REGISTER – IR.VHD



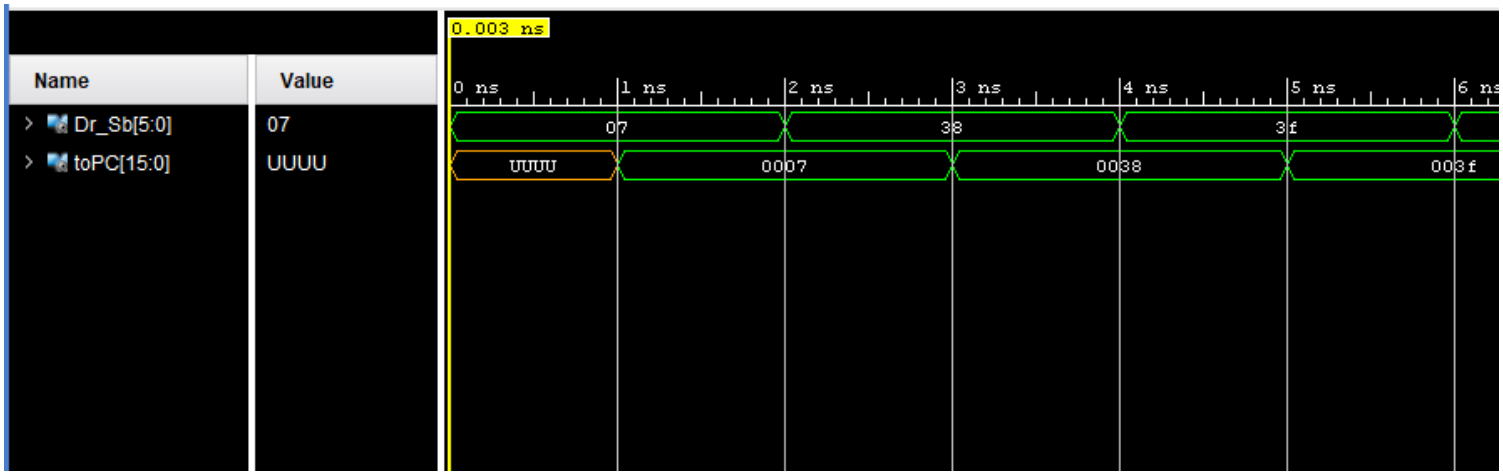
From the Instruction register, the value fromM = "FE00" and IL = '0', which shows that the values in the opcode, DR, SA and SB are not set. When IL = '1' at 10ns then the values of the opcode, DR, SA, and SB were set. We can also change the value of fromM = "ffff" and have the values of the opcode, DR, SA and SB change to the correct values.

7 ZEROFILL – ZEROFILL.VHD



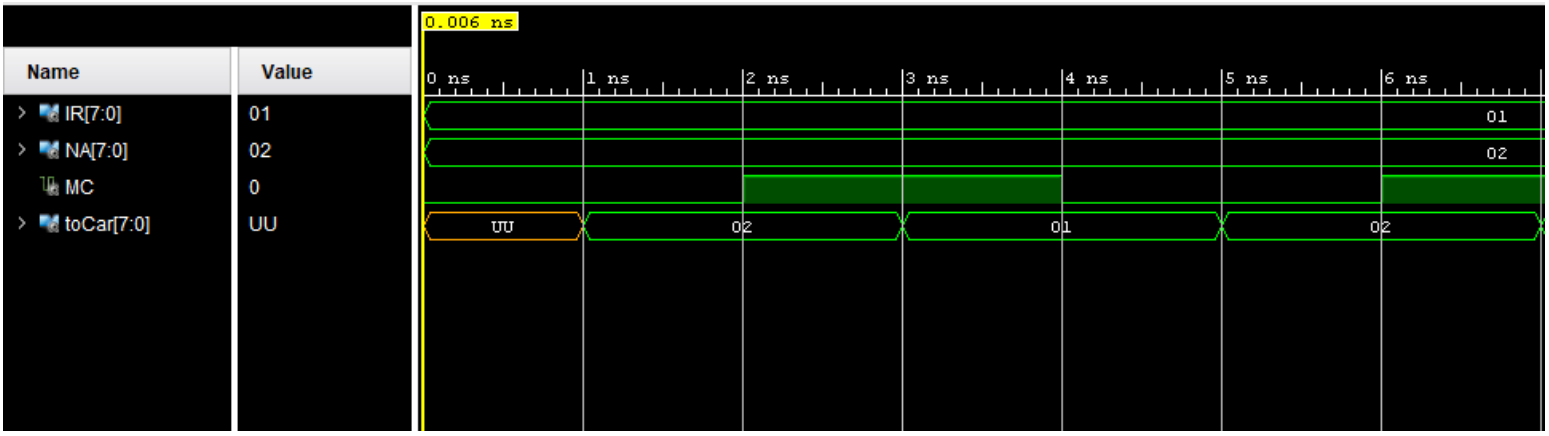
From the image above, the zero fill took in a 3 bit value and filled it with zeros to make a 16 bit value. The above has 2 examples, for a value of “001” and value of “100”.

8 EXTENDER – EXTENDER.VHD

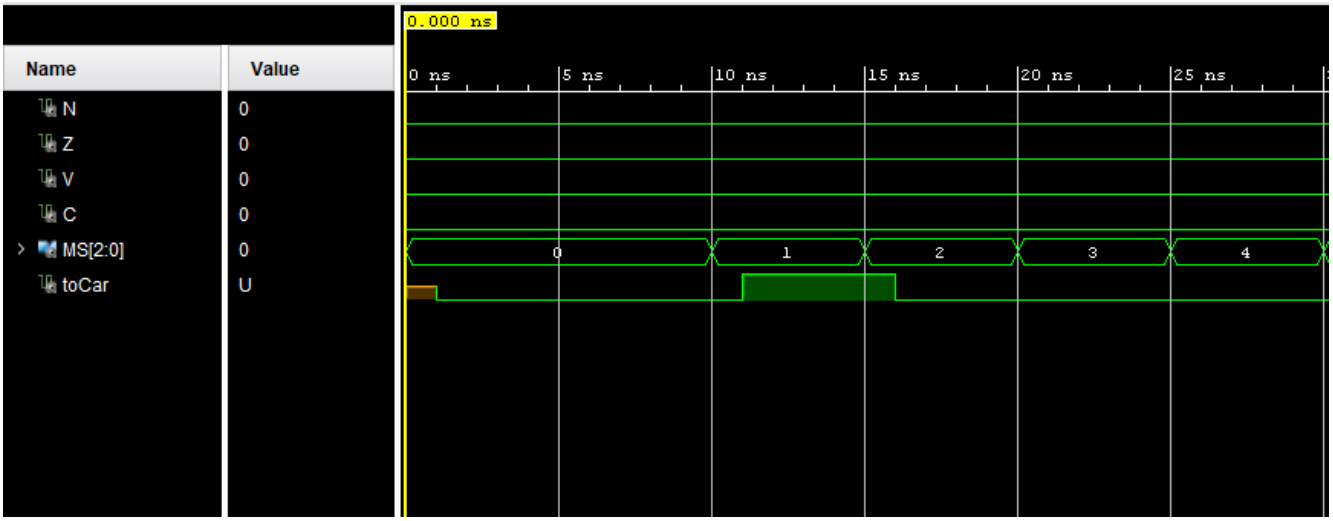


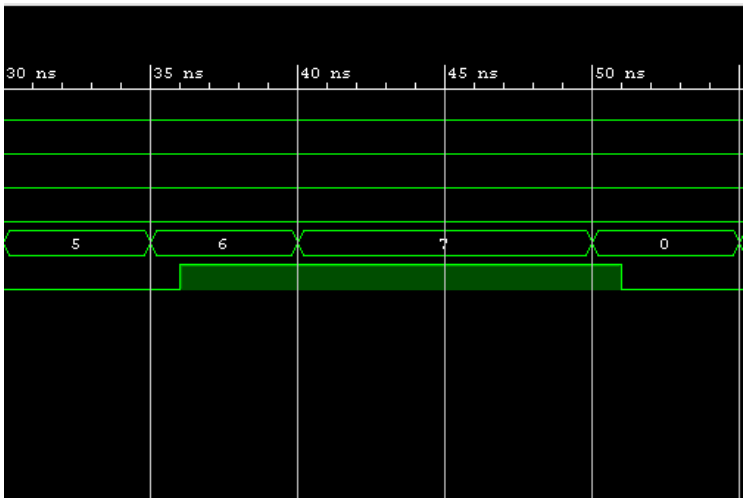
The extender took in a 6 bit value and extend it for the PC. The examples above shows the extender making the 6 bits into 16 bits for values of “000111”, “111000” and “111111”.

9 MULTIPLEXER C – MUXC.VHD



10 MULTIPLEXER S – MUXS.VHD

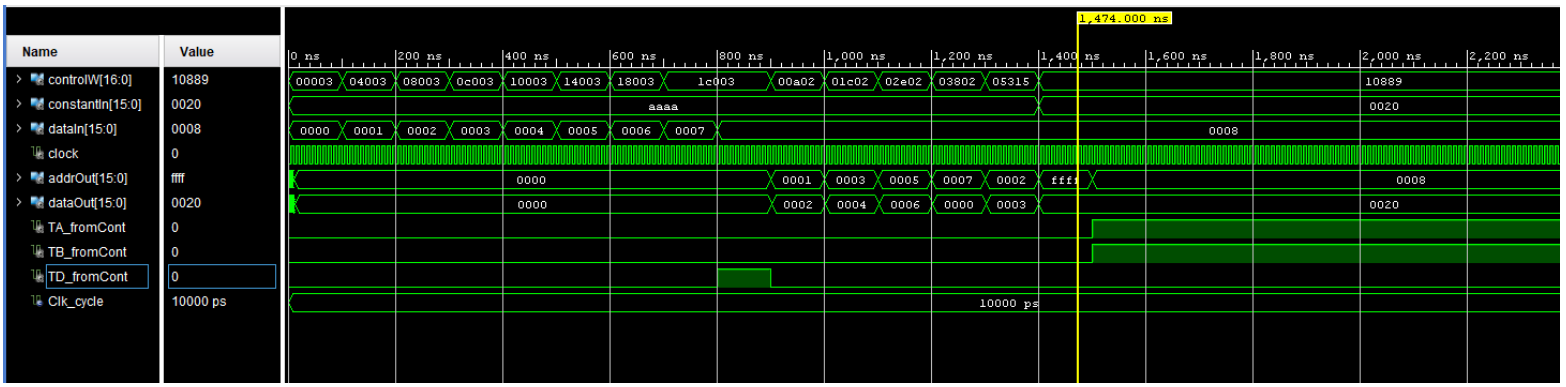




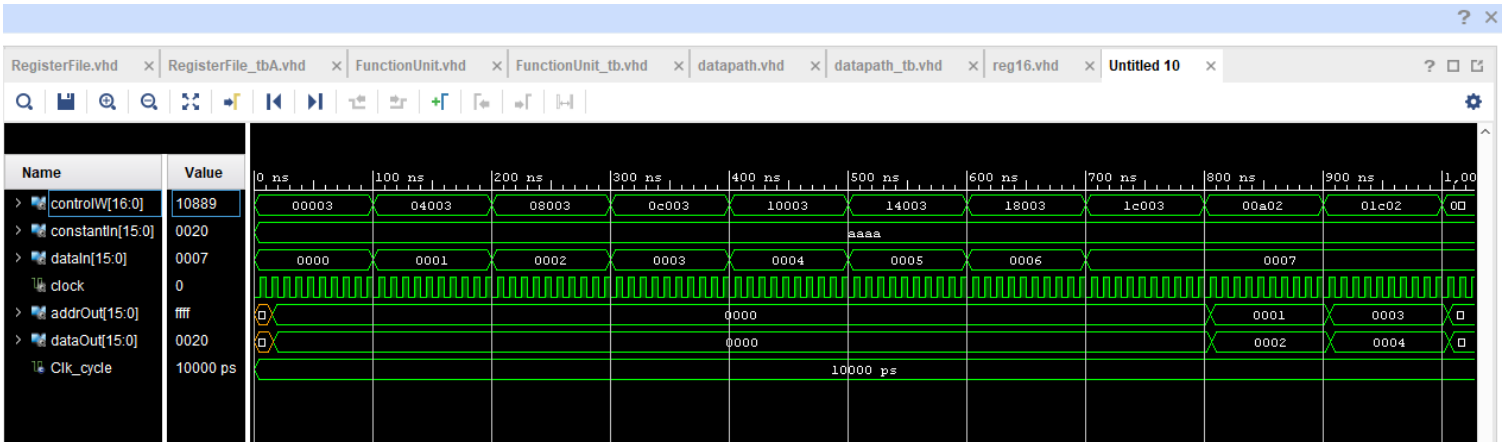
From the Multiplexer S we can see that when going through the values of MS we get the following behaviour:

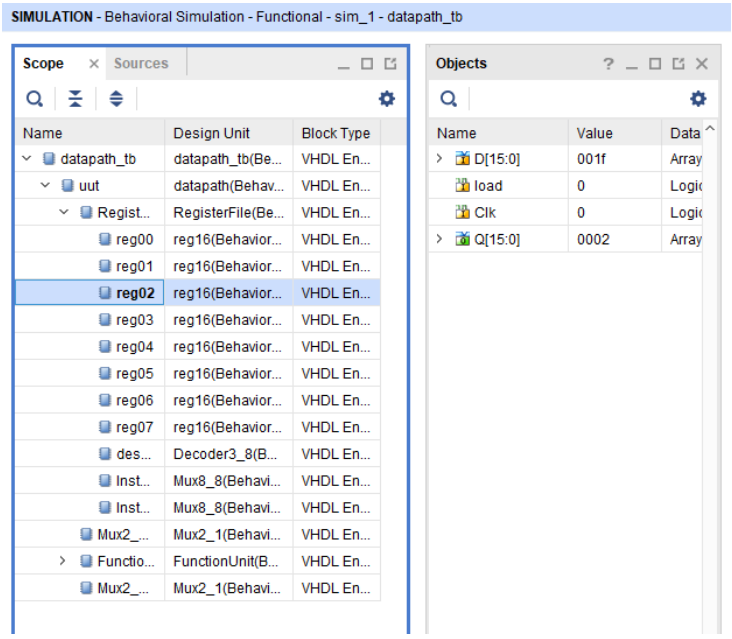
Selected Conditions		Value of MS	Value of toCar
Value 0	0	000	0
Value 1	1	001	1
C	0	010	0
V	0	011	0
Z	0	100	0
N	0	101	0
NOT C	1	110	1
NOT Z	1	111	1

11 MODIFIED DATA PATH – DATAPATH.VHD



This image above is the new Datapath adjusted for TA, TB and TD. The change can be see when TD_fromCont is set to 1 and the value “0008” is stored into R8. We can display this with the addr_out equalling “0008” at 1500ns when TA_fromCont = ‘1’. The following images explain the functionality of the data path.

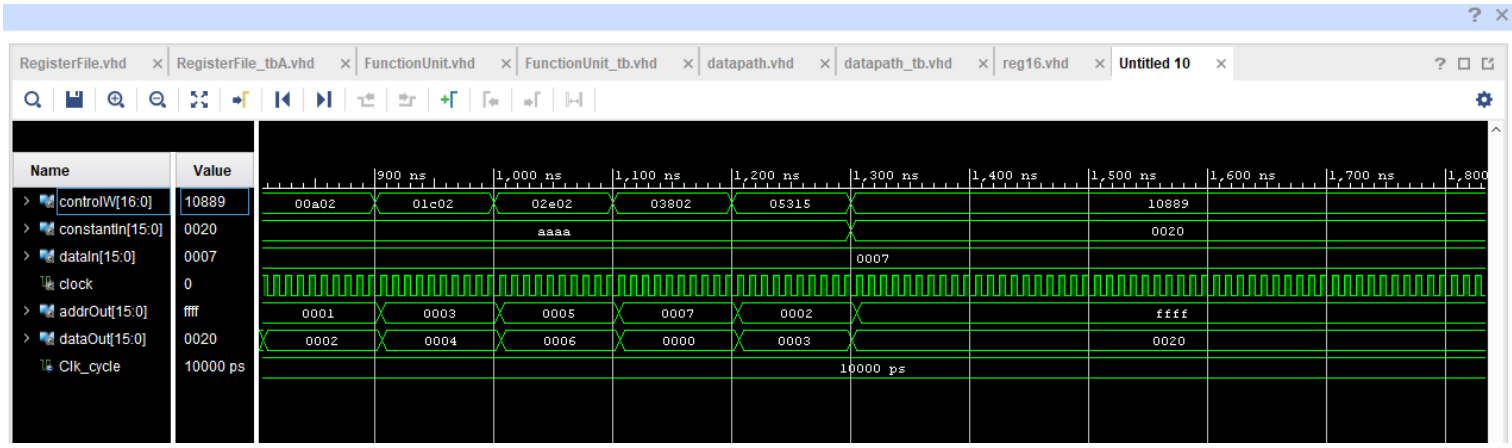


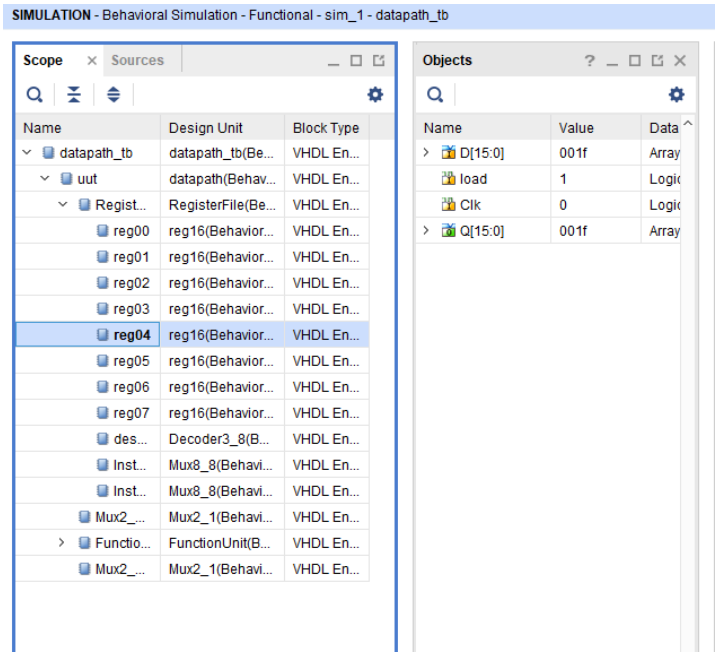


From the 2 screen shots (above and left), using the control word, the registers are being transferred with the value from dataIn. The Register are transferred the value corresponding to its register i.e. Register 0 = “0000”, Register 1 = “0001” and etc.

The control words used to transfer the values into the register are shown below:

Detail	Control World		DA	AA	BA	MB	FS	MD	RW
R0 <- dataIn	00000000000000011		000	000	000	0	00000	1	1
R1 <- dataIn	00100000000000011		001	000	000	0	00000	1	1
R2 <- dataIn	01000000000000011		010	000	000	0	00000	1	1
R3 <- dataIn	01100000000000011		011	000	000	0	00000	1	1
R4 <- dataIn	10000000000000011		100	000	000	0	00000	1	1
R5 <- dataIn	10100000000000011		101	000	000	0	00000	1	1
R6 <- dataIn	11000000000000011		110	000	000	0	00000	1	1
R7 <- dataIn	11100000000000011		111	000	000	0	00000	1	1





From the 2 other screenshots shown here (above and left), we can see that function unit adding selecting addrOut and dataOut (i.e. selecting the register for the A and B input to the function unit). We can see this as addrOut and dataOut are shown in pairs with “0001” and “0002” until “0007” and “0000”.

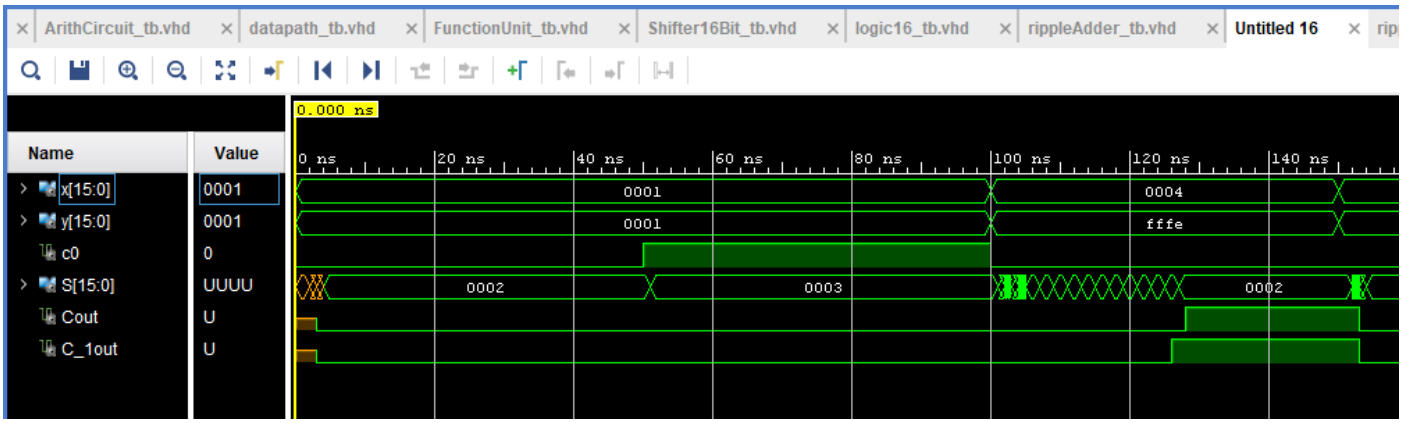
At 1.2us the addrOut and dataOut change to “0002” and “0003” which are register 2 and 3. Register 2 is subtracted by register 3. The result is stored in register 1. This gives the results “ffff.” This result from register 1 is then used at 1.3us to add with the constantIn value = “0020.” The result of this is stored in register 4 (shown in the screenshot on left).

Detail	Control World		DA	AA	BA	MB	FS	MD	RW
R1 -> addrOut R2 -> dataOut	00000101000000010		000	001	010	0	00000	1	0
R3 -> addrOut R4 -> dataOut	00001110000000010		000	011	100	0	00000	1	0
R5 -> addrOut R6 -> dataOut	00010111000000010		000	101	110	0	00000	1	0
R7 -> addrOut R0 -> dataOut	00011100000000010		000	111	000	0	00000	1	0
R1 <- R2 – R3	00101001100010101		001	010	011	0	00101	0	1
R4 <- R1 - constantIn	10000100010001001		100	001	000	1	00010	0	1

12 Full Adder – fullAdder.vdh

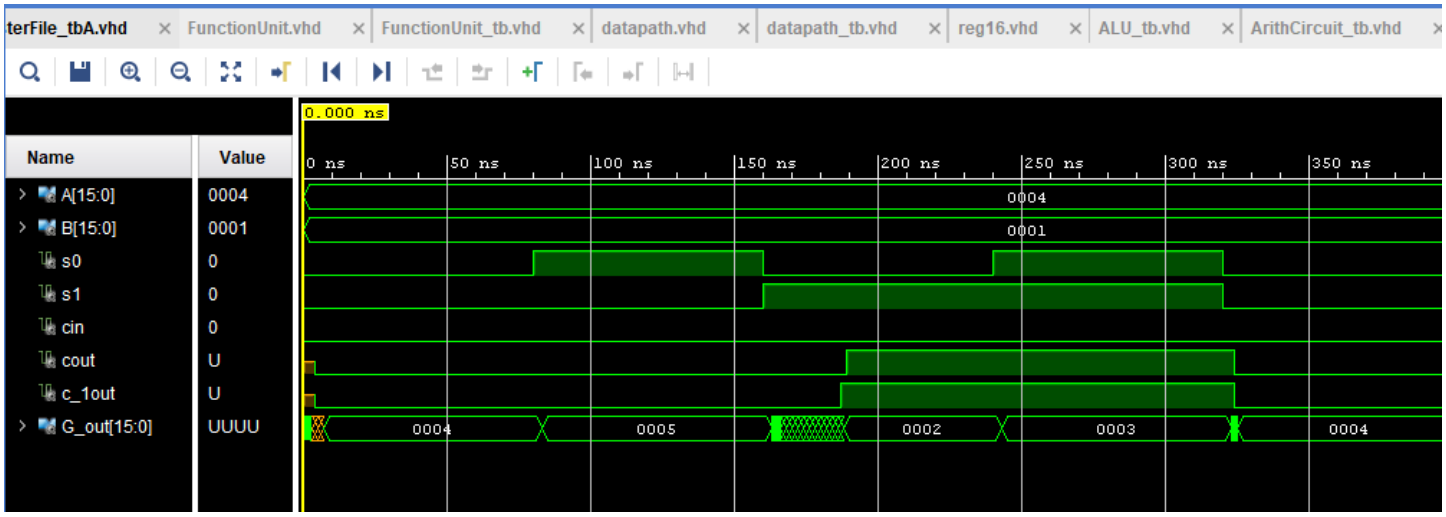
Full Adder test bench showing that when adding 0 and 1 without a carry results in sum = '1'. When a c_in = '1' then the sum = '0' and c_out (carry out) = '1'.

13 Ripple adder - rippleAdder.vdh



From the ripple adder above we can see that x = "0001" and y = "0001". When added the result S = "0002". When there is a carry c0 = '1' then the result S = "0003". Another example with x = "0004" (decimal: 4) and y = "fffe" (decimal: -2) shows that when adding the result S = "0002" (decimal: 2).

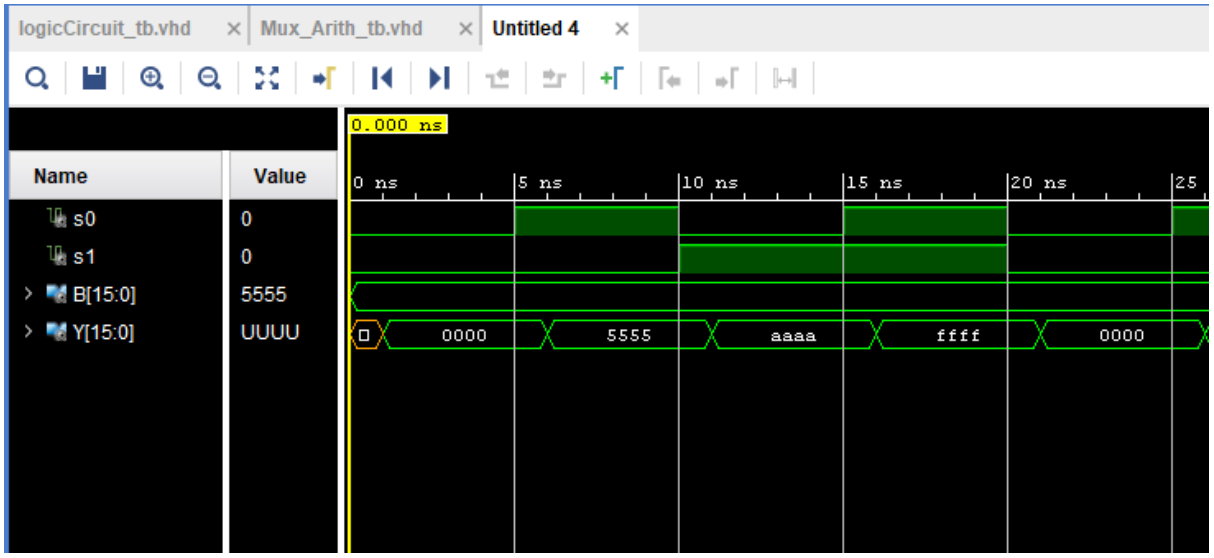
14 ARITHMETIC CIRCUIT – ARITHCIRCUIT.VHD



From the arithmetic circuit we have s0 and s1 instructing the circuit to behave in the following behaviour:

S1	S0	Logic
0	0	G_out = A
0	1	G_out = A+B
1	0	G_out = A + B`
1	1	G_out = A - 1

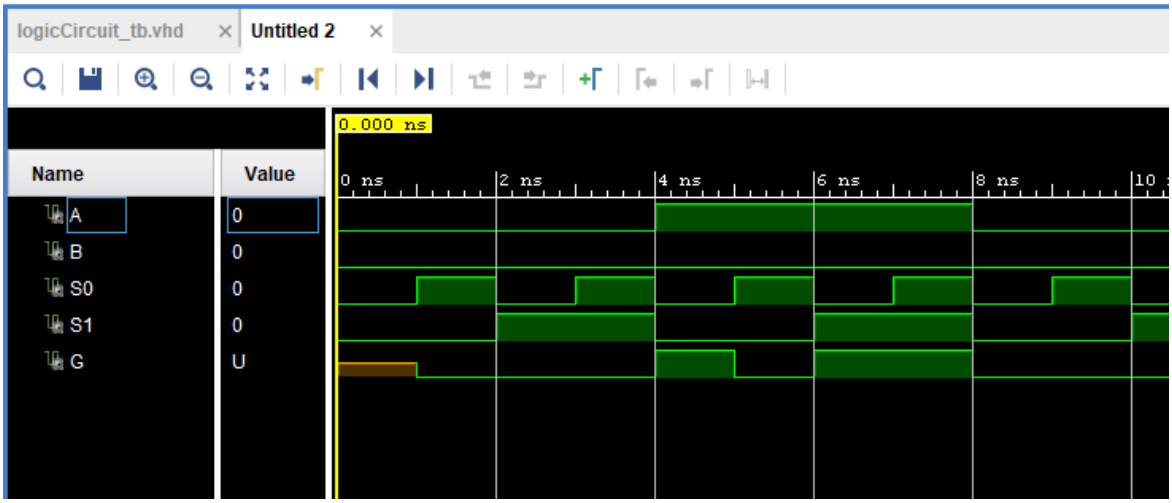
15 MULTIPLEXER FOR ARITHMETIC CIRCUIT – MUX_ARITH.VDH



The multiplexer for the arithmetic circuit follows the following table:

S1	S0	Logic
0	0	Y = 0000
0	1	Y = B
1	0	Y = B`
1	1	Y = 1111

16 Logic Circuit 1 Bit – logicCircuit.vhd

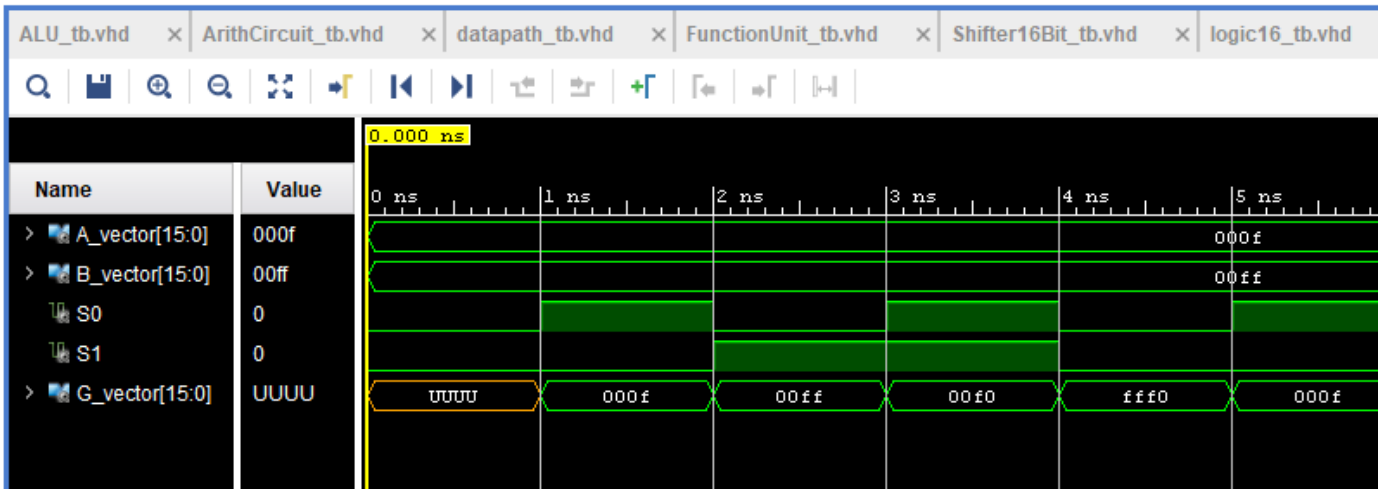


For the logic circuit 1 bit slide, the results follow the following table:

S1	S0	Logic
0	0	G = A and B
0	1	G = A or B
1	0	G = A xor B
1	1	G = not A

We can see that the results are that G = ‘0’ when S0 = ‘0’, S1 = ‘0’ (AND Gate), S0 = ‘1’ and S1 = ‘0’ (OR Gate) and S0 = ‘0’ and S1 = ‘1’ (XOR Gate). G = ‘1’ when S0 = ‘1’ and S1 = ‘1’ (Not A).

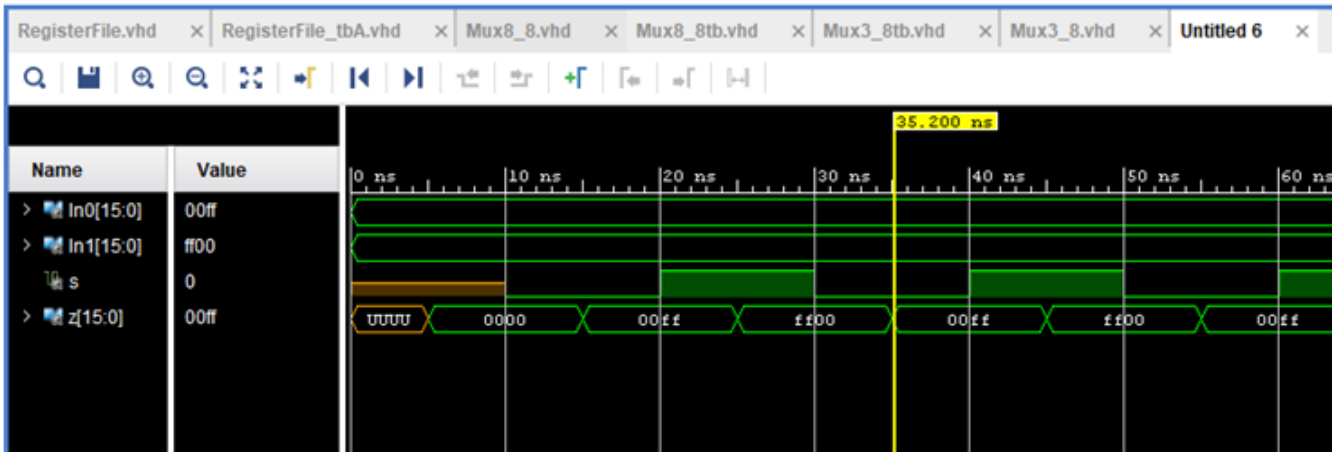
17 Logic Circuit 16 Bits – logic16.vhd



From the logic Circuit 16 bits we can see that A = “000f” and B = “00ff” and with the we can see the results below:

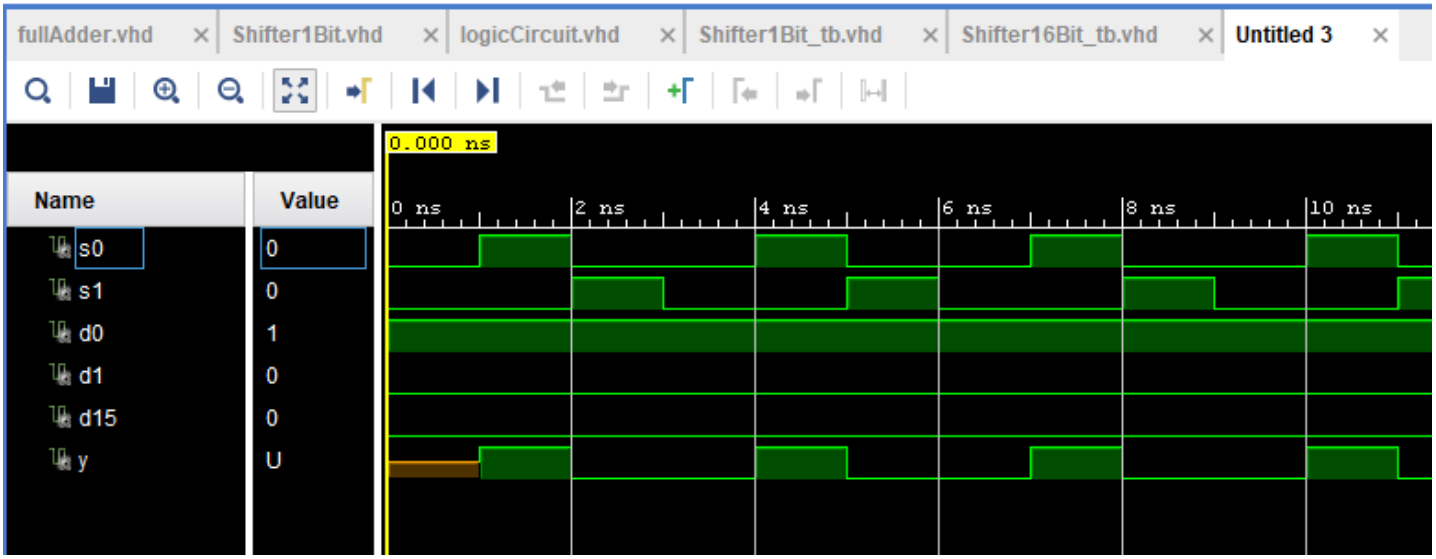
S1	S0	A	B	Logic Gate	G
0	0	000f	00ff	AND	000f
0	1	000f	00ff	OR	00ff
1	0	000f	00ff	XOR	00f0
1	1	000f	00ff	NOT A	fff0

18 Multiplexer 2 to 1 – Mux2_1.vhd



This multiplexer is the same implementation of project 1A. The results z = In0 when s = 0 and z = In1 when s = 1.

19 SHIFTER 1 BIT – SHIFTER1BIT.VHD

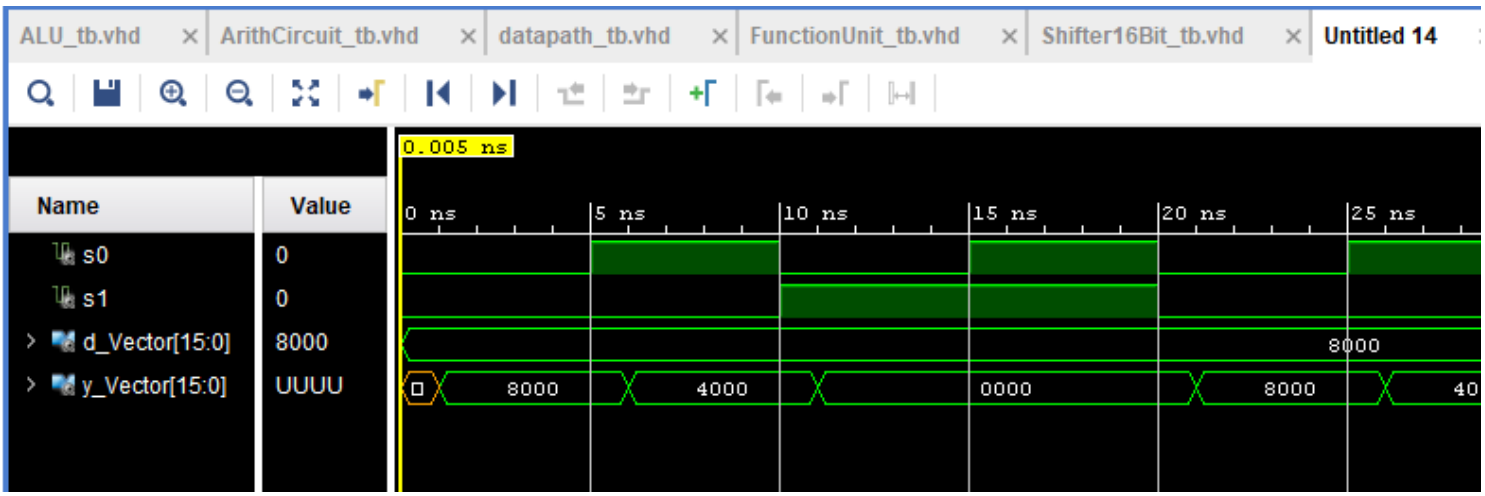


The 1 bit shifter will take behave with the following table:

S0	S1	Result
0	0	d0
0	1	d1
1	0	d15
1	1	0

D0, d1, and d15 map to the current bit and the left bit and right bit, where d0 is the current bit and d1 is the left bit and d15 is the right bit.

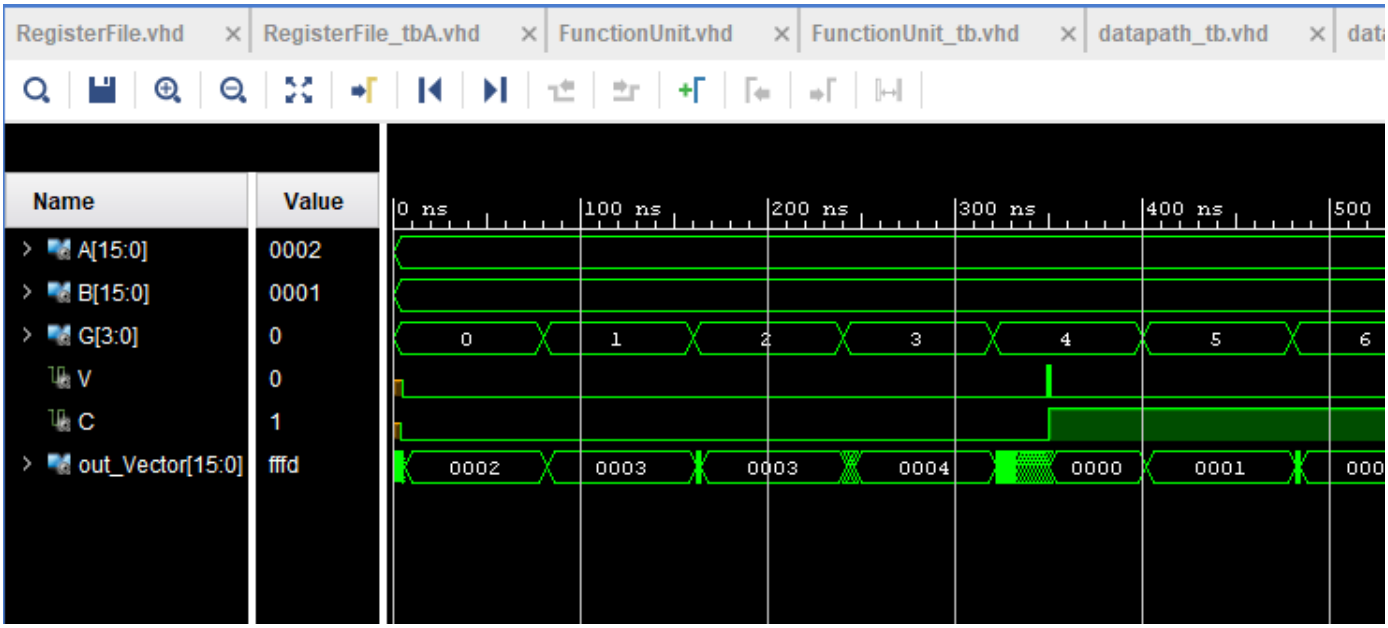
20 SHIFTER 16 BIT – SHIFTER16BIT.VHD

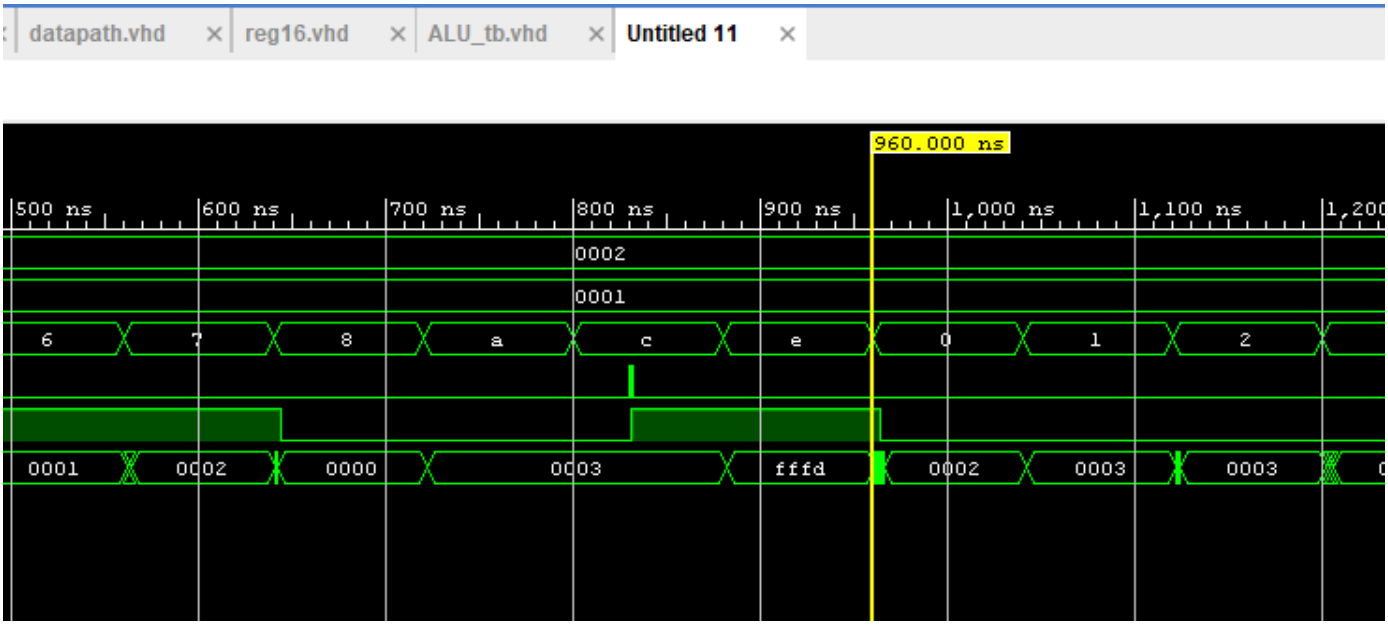


From the 16 bit shifter, it can be seen that the input is d_Vector = “8000” in hex. The shifter follows the following behaviour and logic:

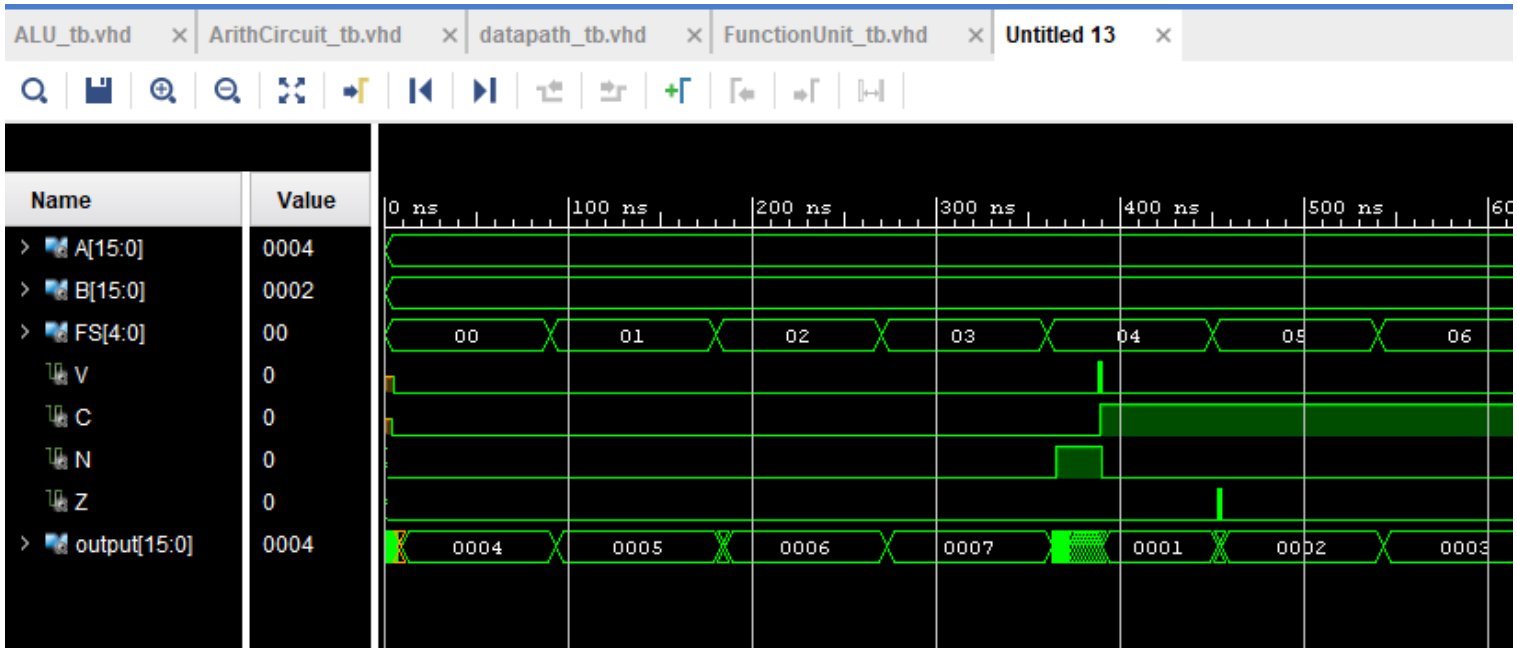
S0	S1	Logic	Result
0	0	Transfer	y_Vector = “8000”
0	1	Shift Right	y_Vector = “4000”
1	0	Shift Left	y_Vector = “0000”

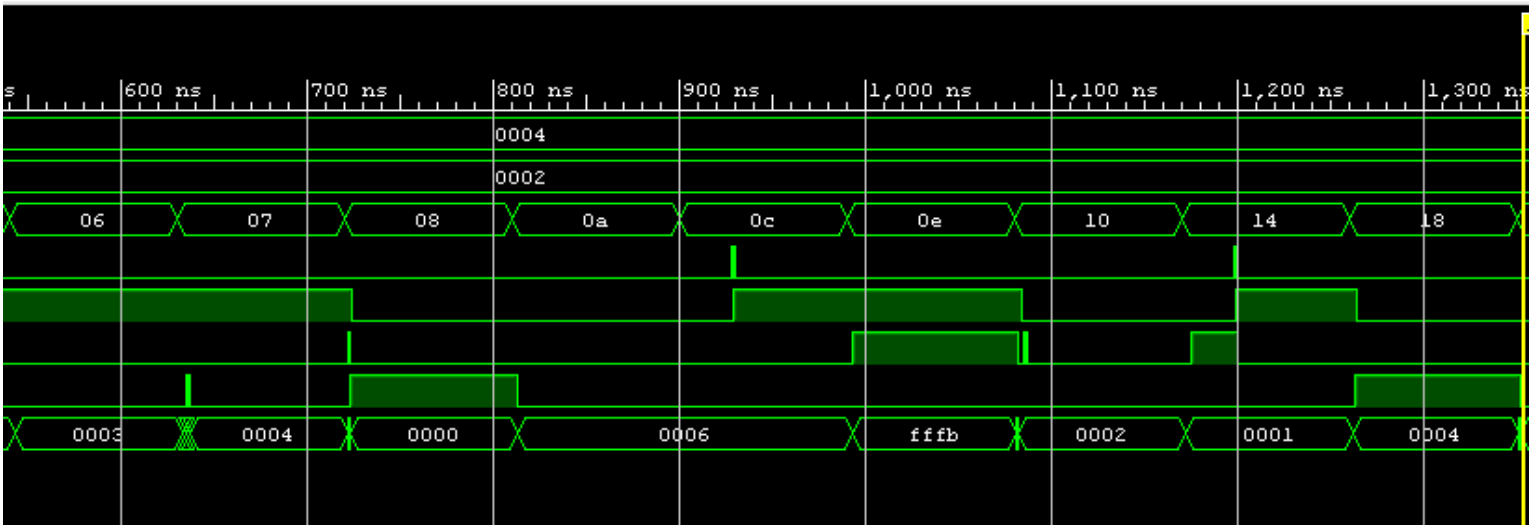
21 ARITHMETIC LOGIC UNIT – ALU.VDH





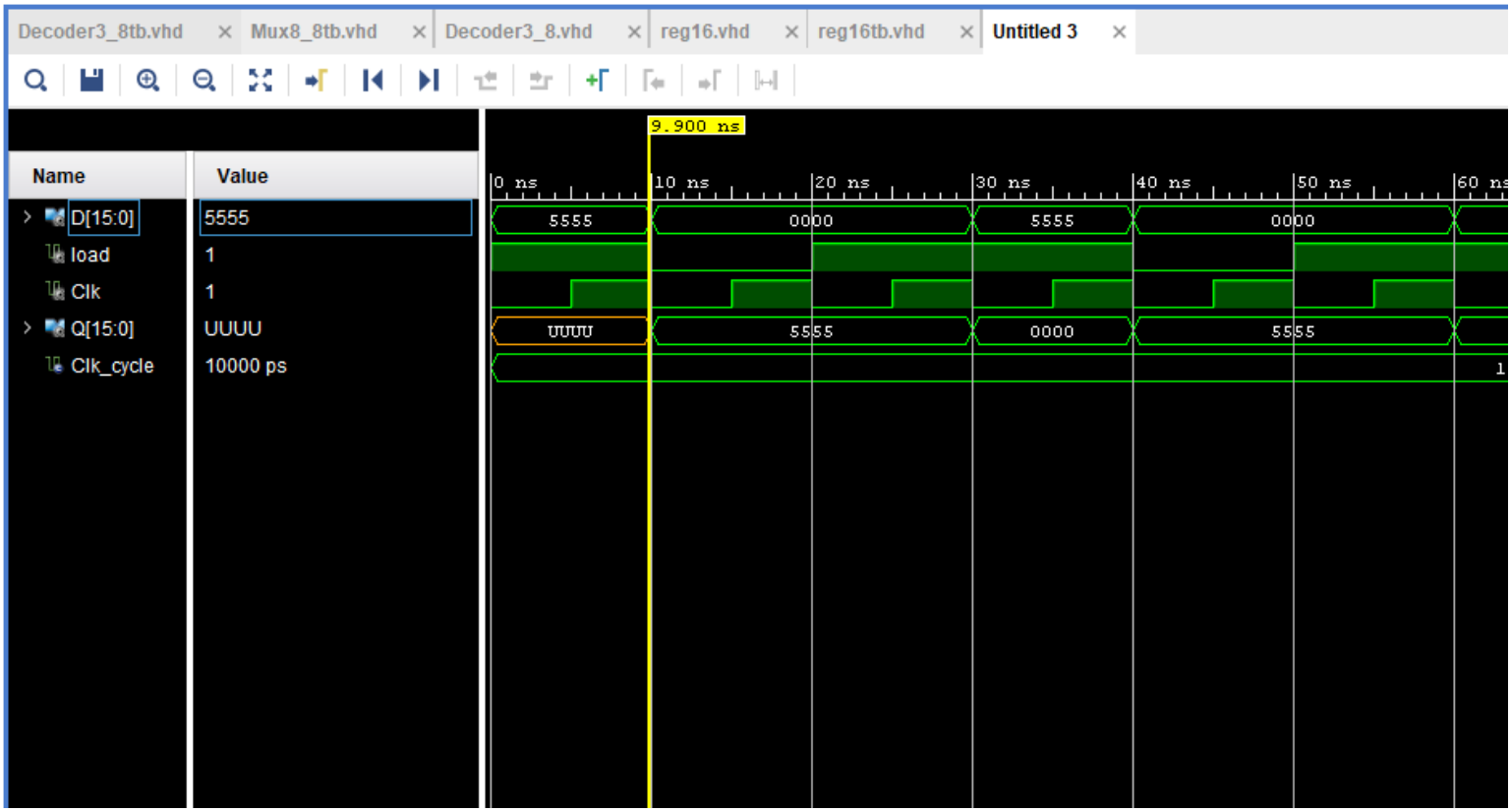
22 FUNCTION UNIT – FUNCTIONUNIT.VHD





23 Modified Register File – RegisterFile.vhd

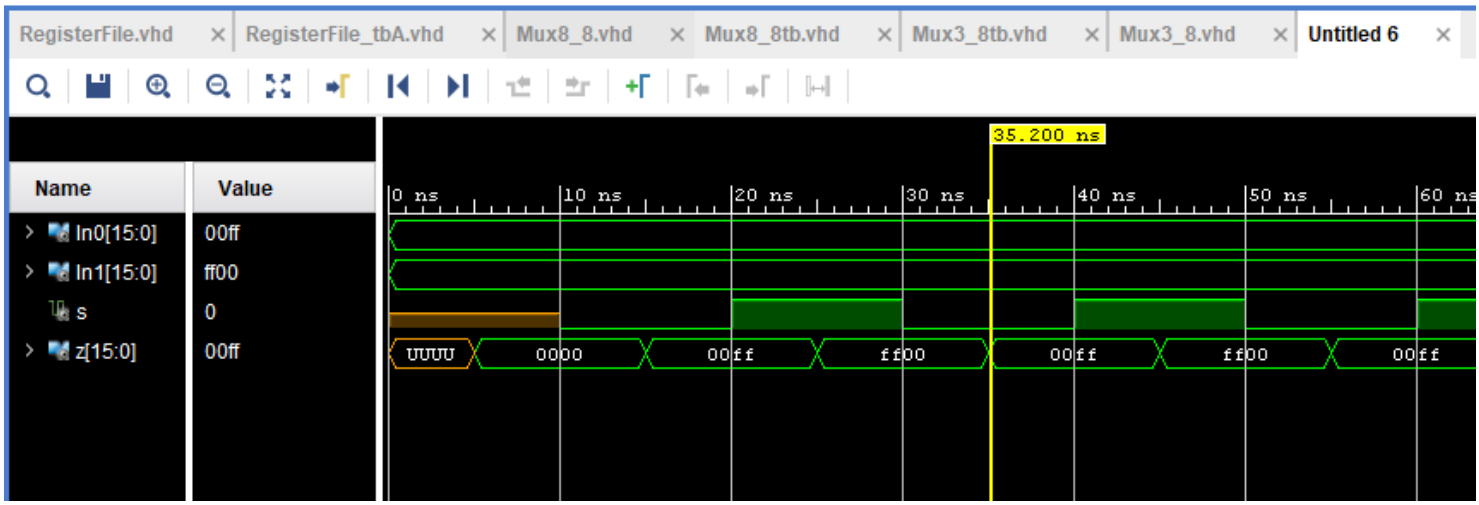
24 Single Register – reg16.vhd



25 MODIFIED MULTIPLEXER – MUX_8_8.VDH

Setting the value of in8 = 0080 and outputting it to z when sT = ‘1’.

26 MULTIPLEXER – MUX3_8.VDH



From the Mu3_8 at 30ns we can see that the output ‘z’ = “ff00”. This changes as the select, ‘s’ = 0 and after 5ns (at 35.20ns) that the output ‘z’ = “00ff” which is ‘In0’. We can see that it changes back to “ff00” as the ‘s’ changes to 1. This is correct as s = 0 maps to In0 and s = 1 maps to In1.

27 MODIFIED DECODER – DECODER3_8.VDH

When TA is set to ‘1’ then D8 will be set to ‘1’ regardless of A0 – A2.