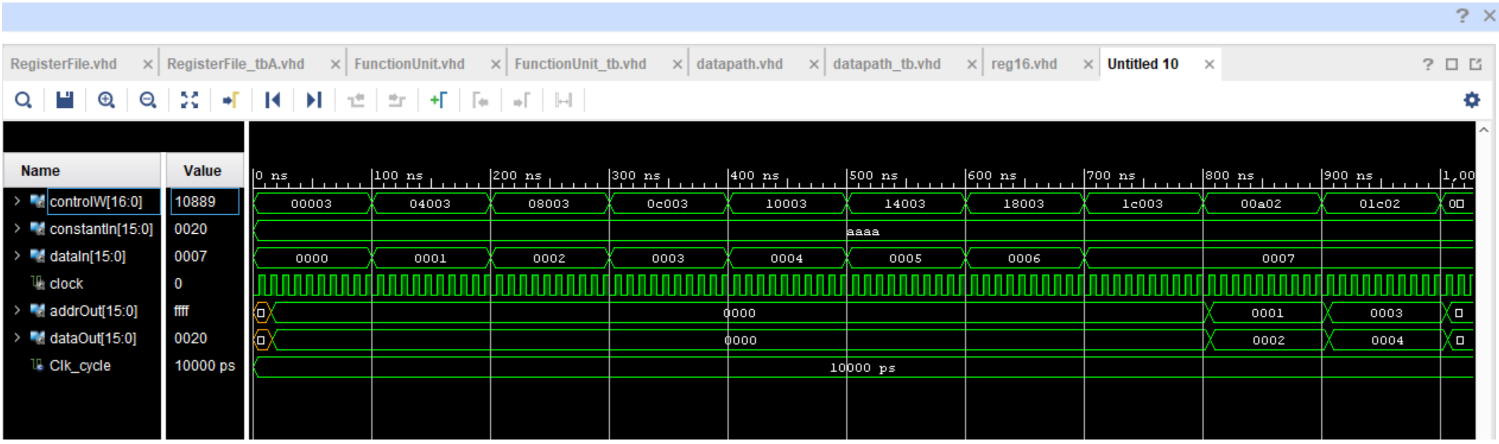
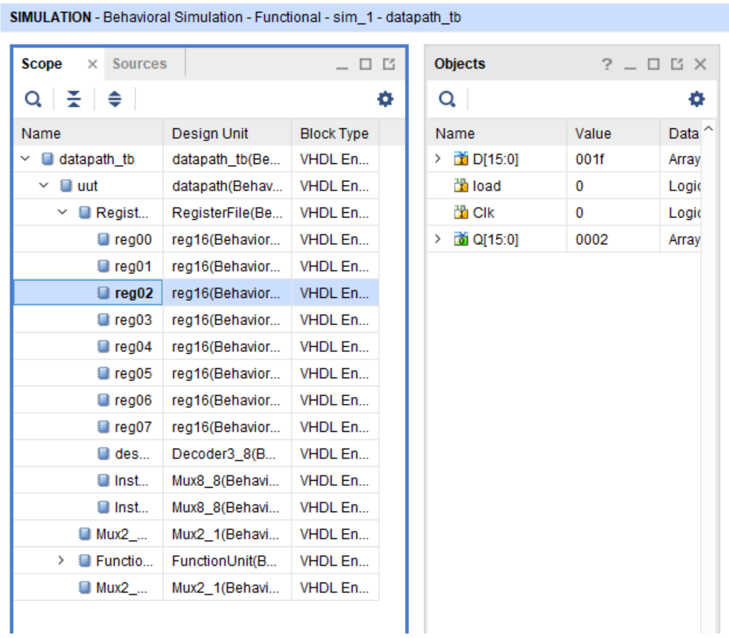
CS2022 Computer Architecture

Project 1 – Datapath Design Part B

Name: Caleb Teo
Student Number: 15324741
Data: 26/02/2018

1 DATA PATH – DATAPATH.VHD

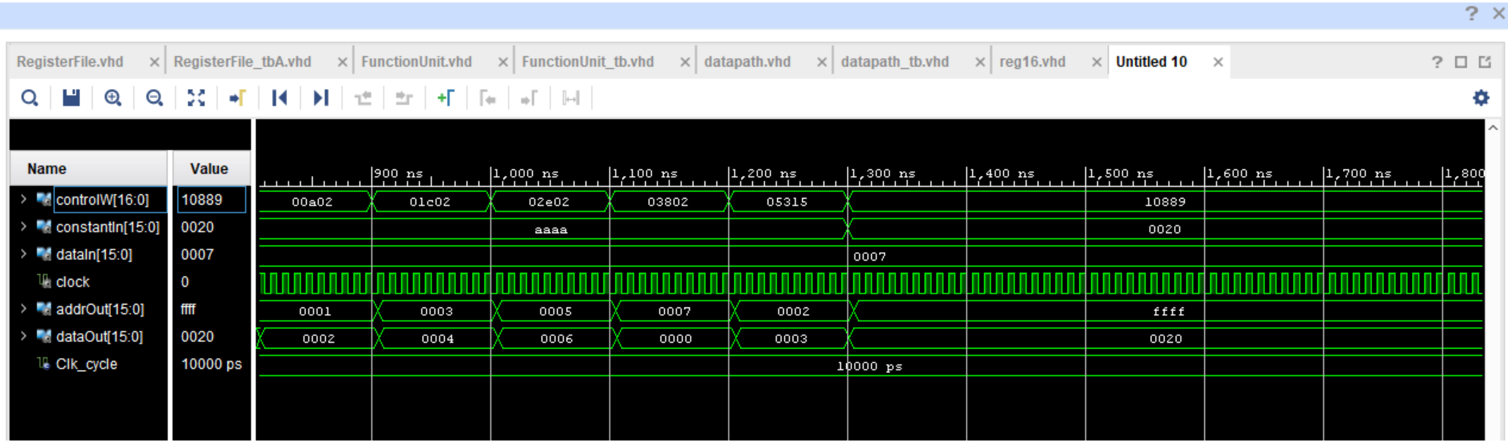


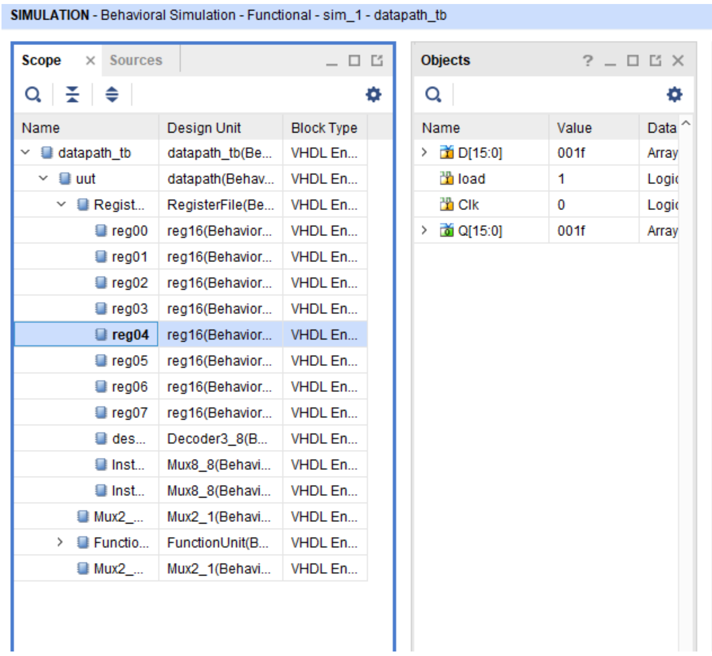


From the 2 screen shots (above and left), using the control word, the registers are being transferred with the value from dataIn. The Register are transferred the value corresponding to its register i.e. Register 0 = "0000", Register 1 = "0001" and etc.

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



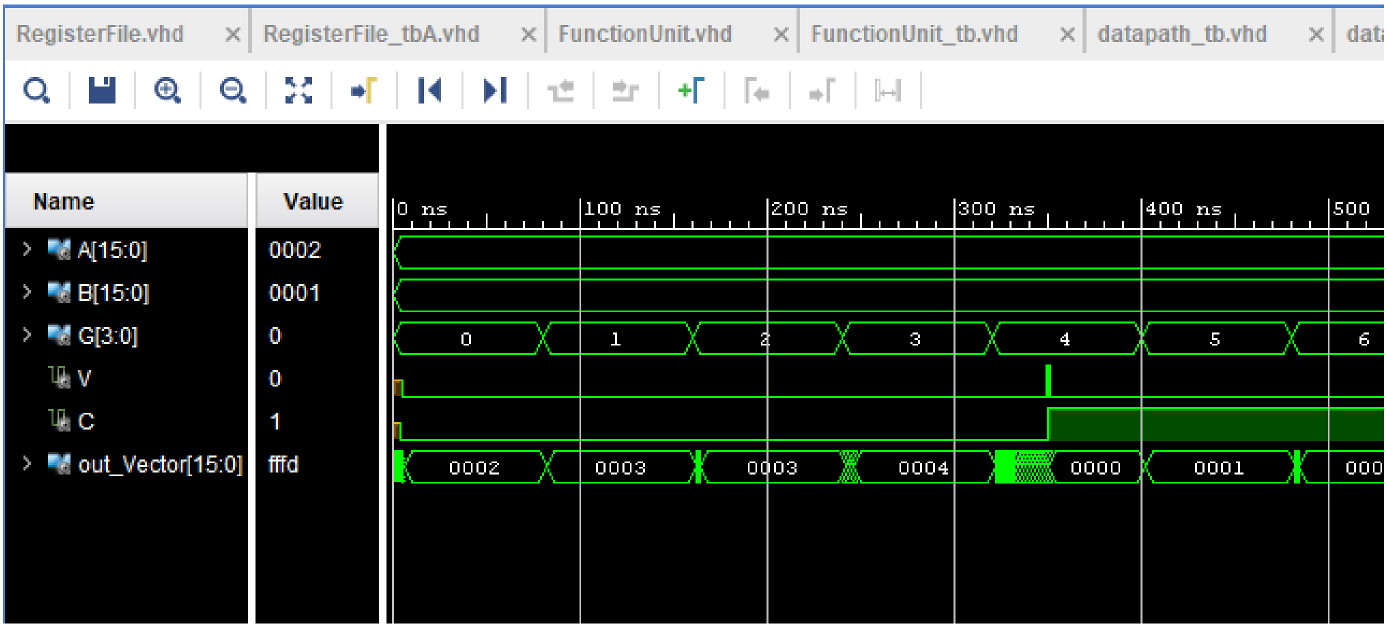


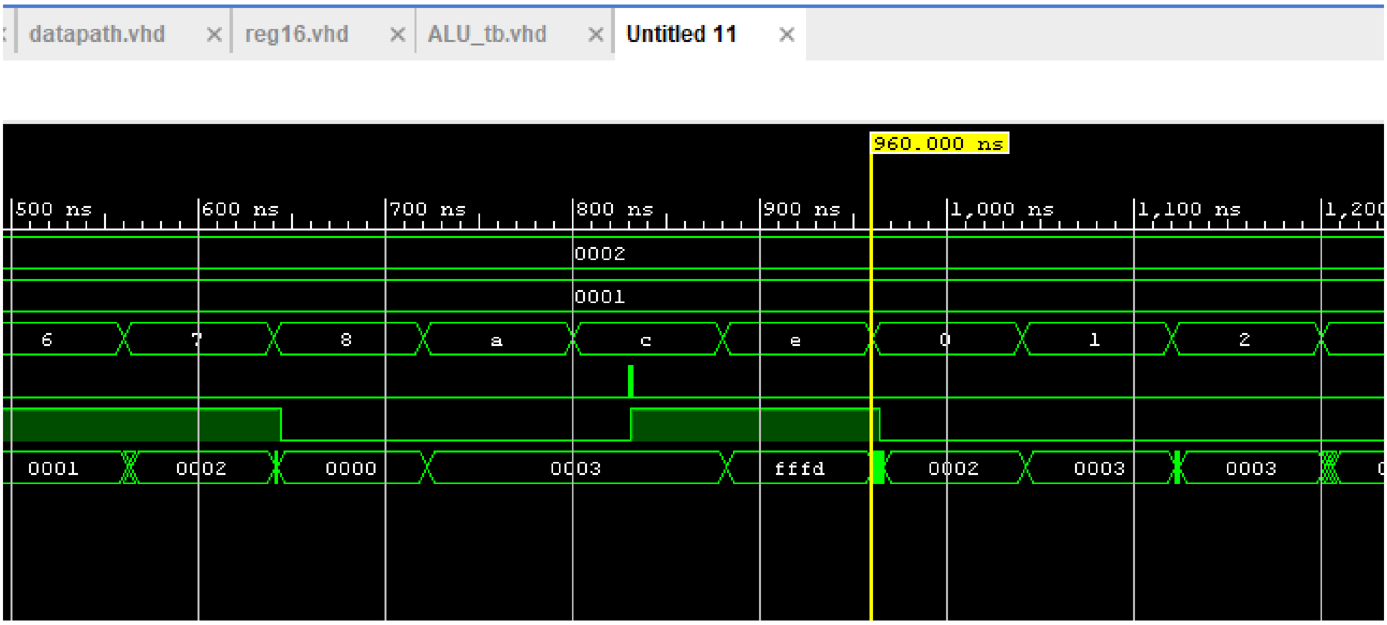
From the 2 other screenshots shown here (above and left), we can see that function unit adding selecting addrOut and dataOut (i.e. selecting the register for the A and B input to the function unit). We can see this as addrOut and dataOut are shown in pairs with "0001" and "0002" until "0007" and "0000".

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

2 ARITHMETIC LOGIC UNIT – ALU.VDH



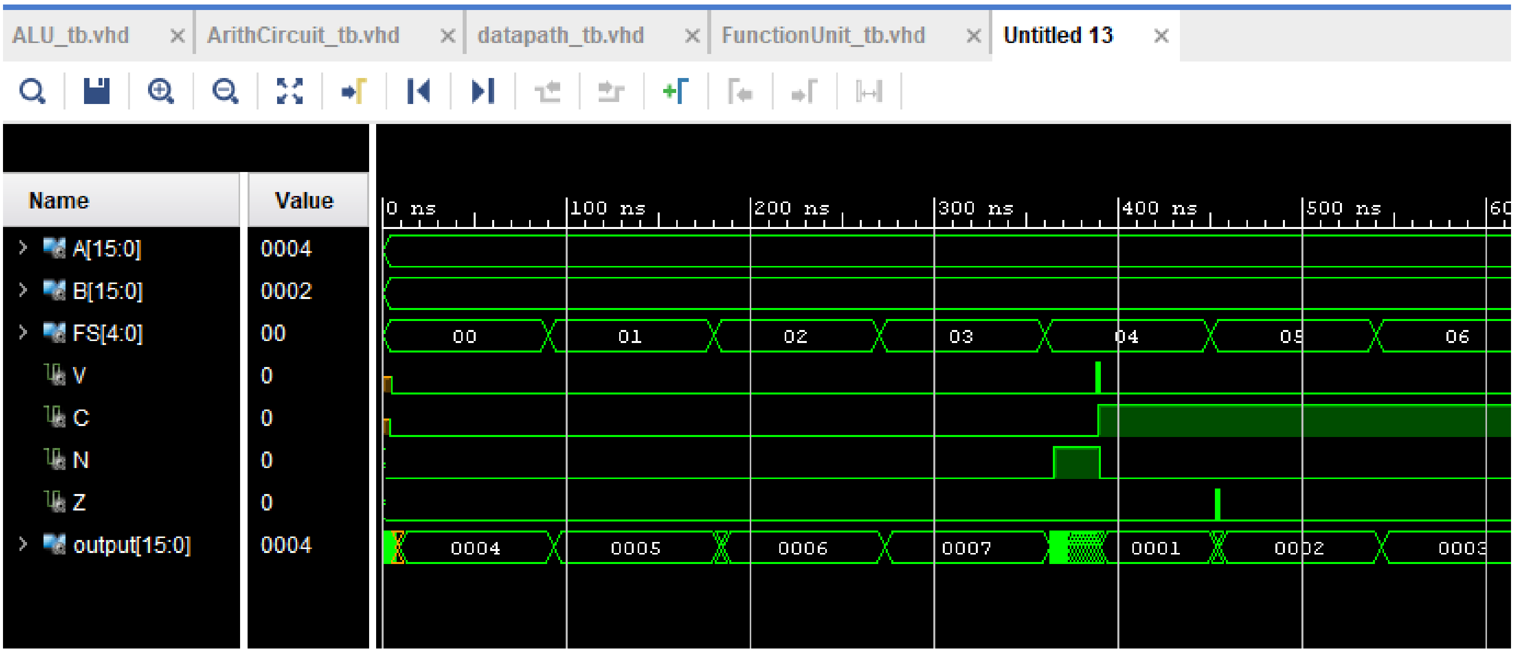


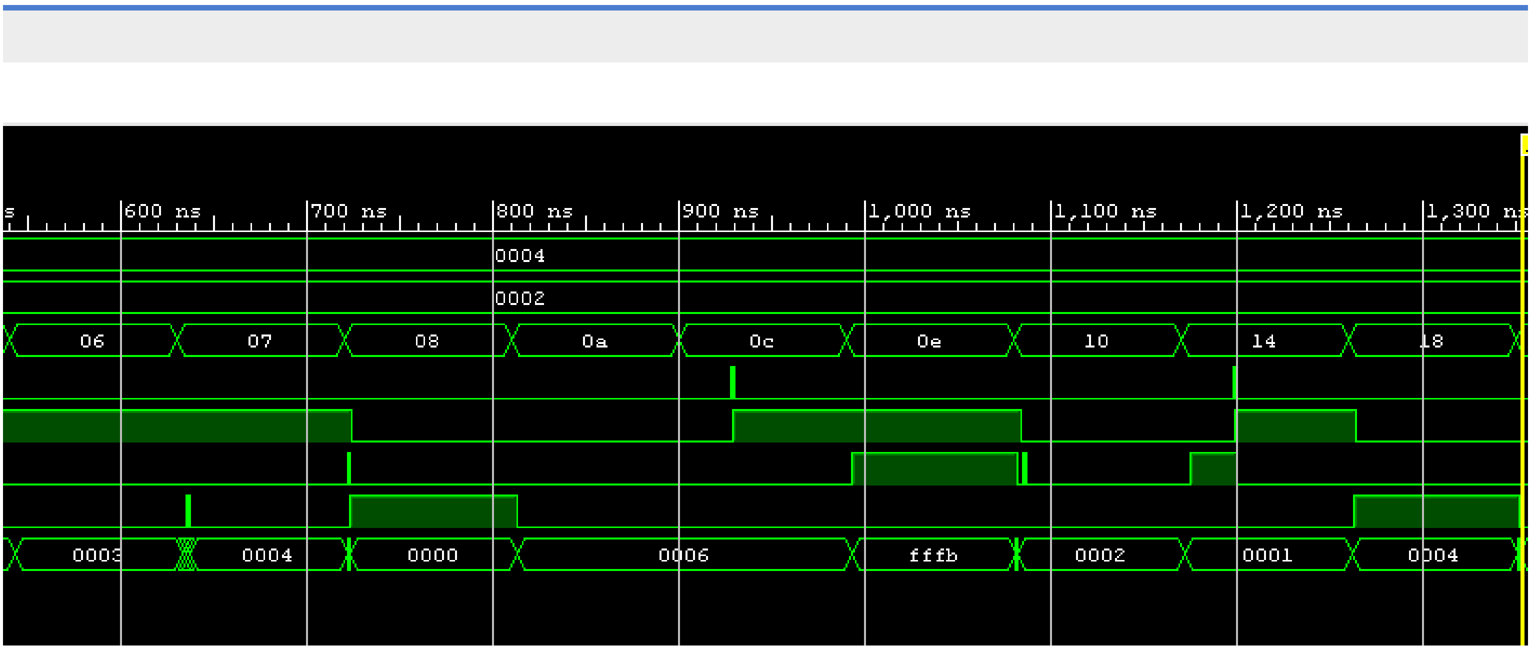
From the ALU above we can see that G goes from “0000” to “1110”, which will should the corresponding results:

Logic	G	A	B	Result (out)	V	C
Out <- A	0000	0002	0001	0002	0	0
Out <- A + 1	0001	0002	0001	0003	0	0
Out <- A + B	0010	0002	0001	0003	0	0
Out <- A+B+1	0011	0002	0001	0004	0	0
Out<- A+B`	0100	0002	0001	0000	0	1
Out<- A+B`+1	0101	0002	0001	0001	0	1

Out <- A – 1	0110	0002	0001	0001	0	1
Out <- A	0111	0002	0001	0002	0	1
Out <- A^B	1000	0002	0001	0000	X	X
Out <- AᵛB	1010	0002	0001	0003	X	X
Out <- A ⊕ B	1100	0002	0001	0003	X	X
Out <- not A	1110	0002	0001	fffd	X	X

3 FUNCTION UNIT – FUNCTIONUNIT.VHD





From the Function Unit we can see the results follow the table from the FS code definition in the project description (Table 1).

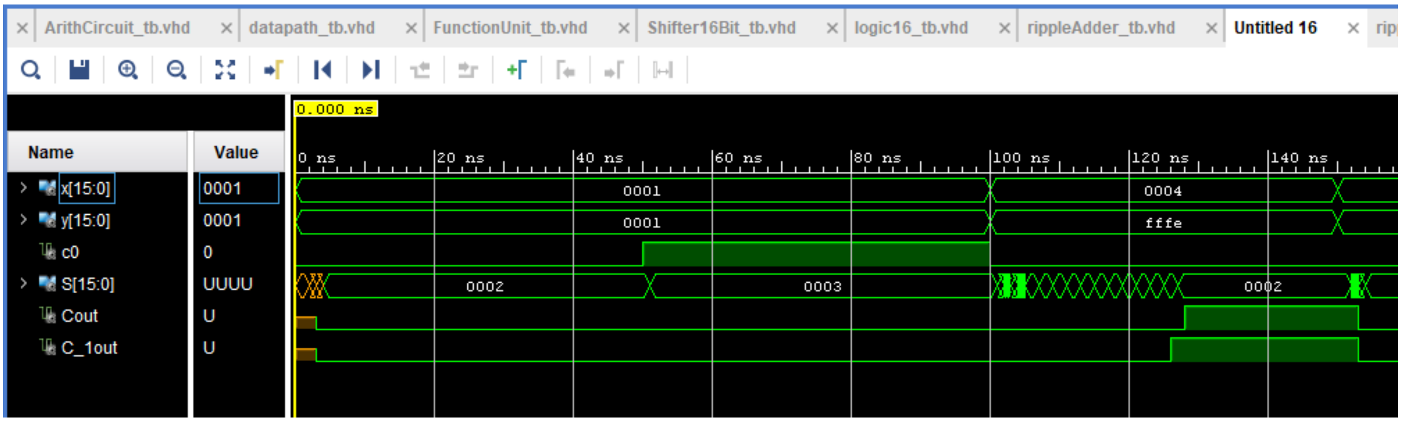
Table 1: FS code definition

FS	MF Select	G Select	H Select	Micro-operation
00000	0	0000	00	$F = A$
00001	0	0001	00	$F = A + 1$
00010	0	0010	00	$F = A + B$
00011	0	0011	00	$F = A + B + 1$
00100	0	0100	01	$F = A + \bar{B}$
00101	0	0101	01	$F = A + \bar{B} + 1$
00110	0	0110	01	$F = A - 1$
00111	0	0111	01	$F = A$
01000	0	1000	00	$F = A \wedge B$
01010	0	1010	10	$F = A \vee B$
01100	0	1100	10	$F = A \oplus B$
01110	0	1110	10	$F = \bar{A}$
10000	1	0000	00	$F = B$
10100	1	0100	01	$F = srB$
11000	1	1000	10	$F = slB$

4 FULL ADDER – FULLADDER.VDH

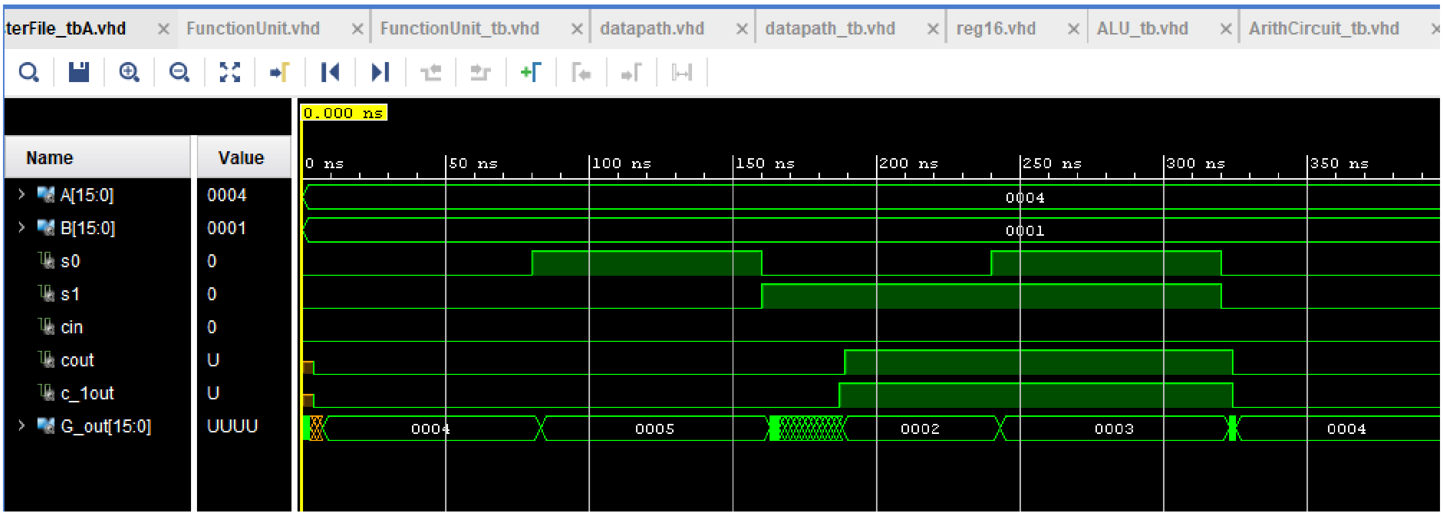
Full Adder test bench showing that when adding 0 and 1 without a carry results in sum = ‘1’. When a c_in = ‘1’ then the sum = ‘0’ and c_out (carry out) = ‘1’.

5 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).

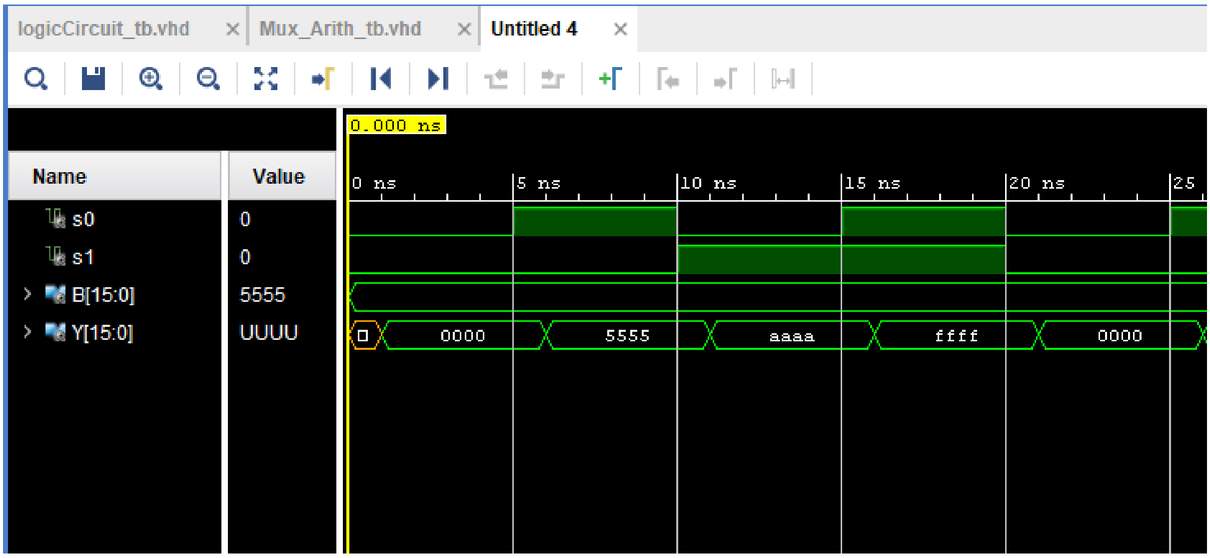
6 Arithmetic Circuit – ArithCircuit.vhd



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

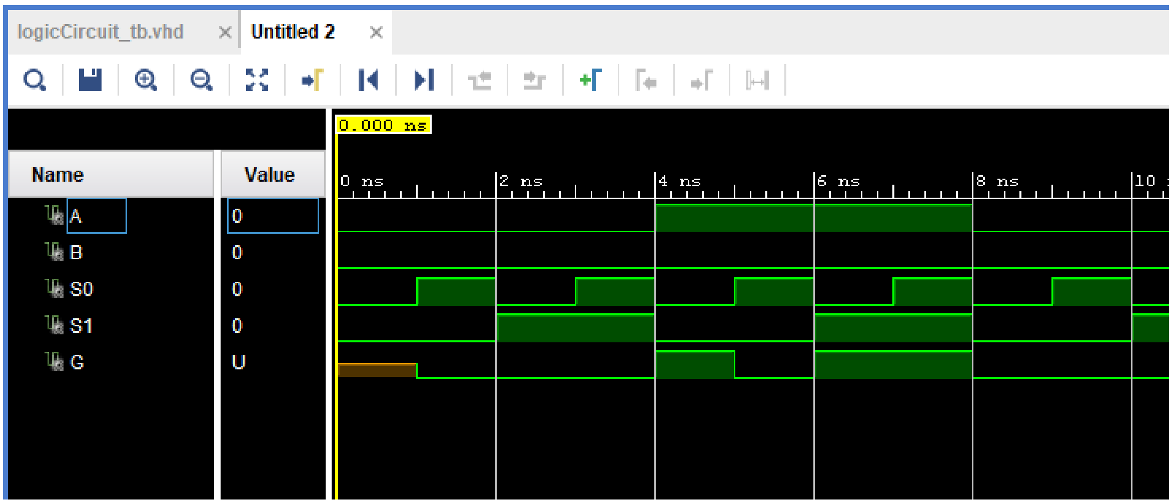
7 MULTIPLEXER FOR ARITHMETIC CIRCUIT – MUX_ARITH.VDH



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

8 LOGIC CIRCUIT 1 BIT – LOGICCIRCUIT.VHD



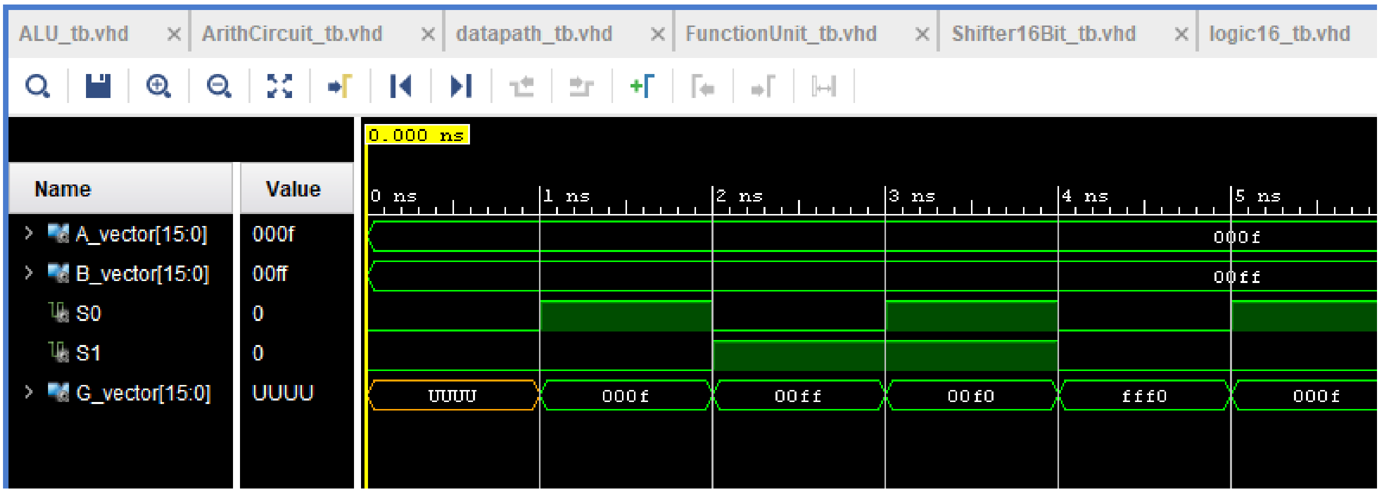
For the logic circuit 1 bit slide, the results follow the following table:

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).

9 LOGIC CIRCUIT 16 BITS – LOGIC16.VHD



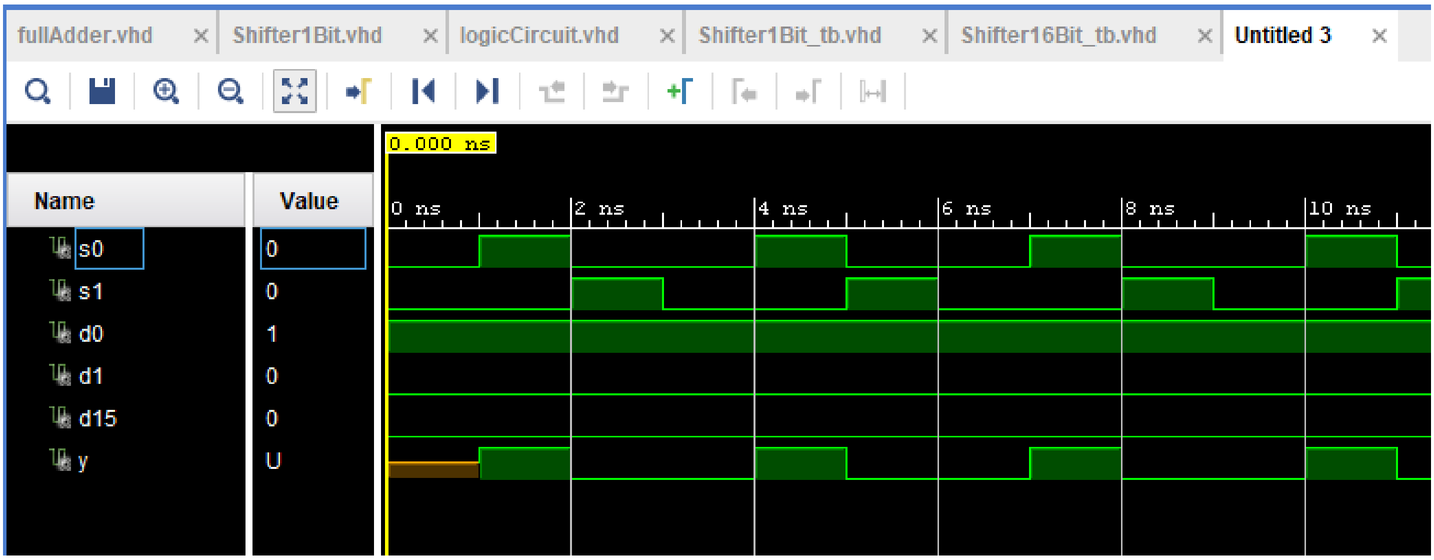
From the logic Circuit 16 bits we can see that A = “000f” and B = “00ff” and with the we can see the results below:

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

10 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.

11 Shifter 1 Bit – Shifter1Bit.vhd

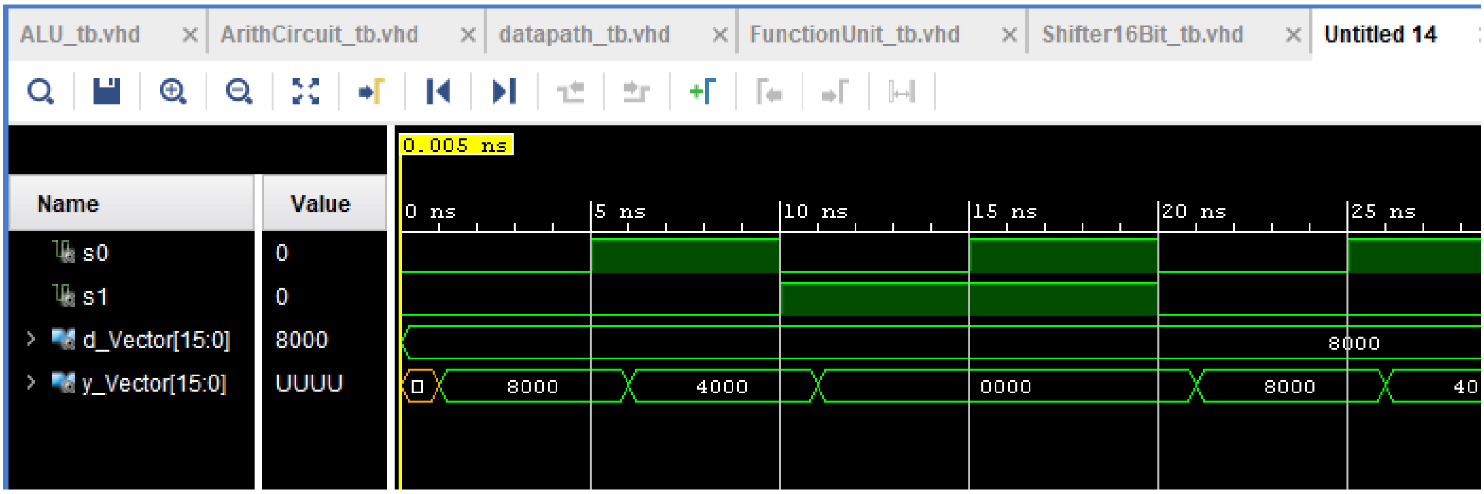


The 1 bit shifter will take behave with the following table:

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.

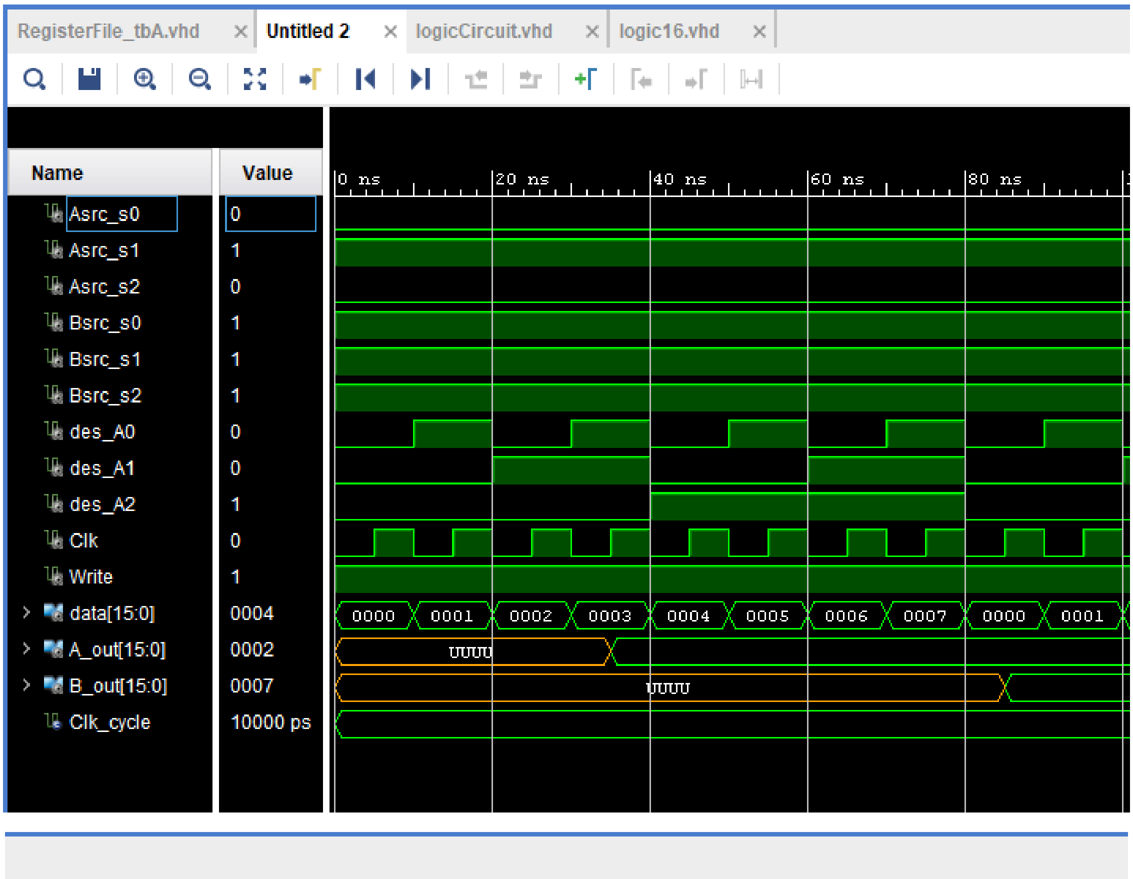
12 Shifter 16 Bit – Shifter16Bit.vhd

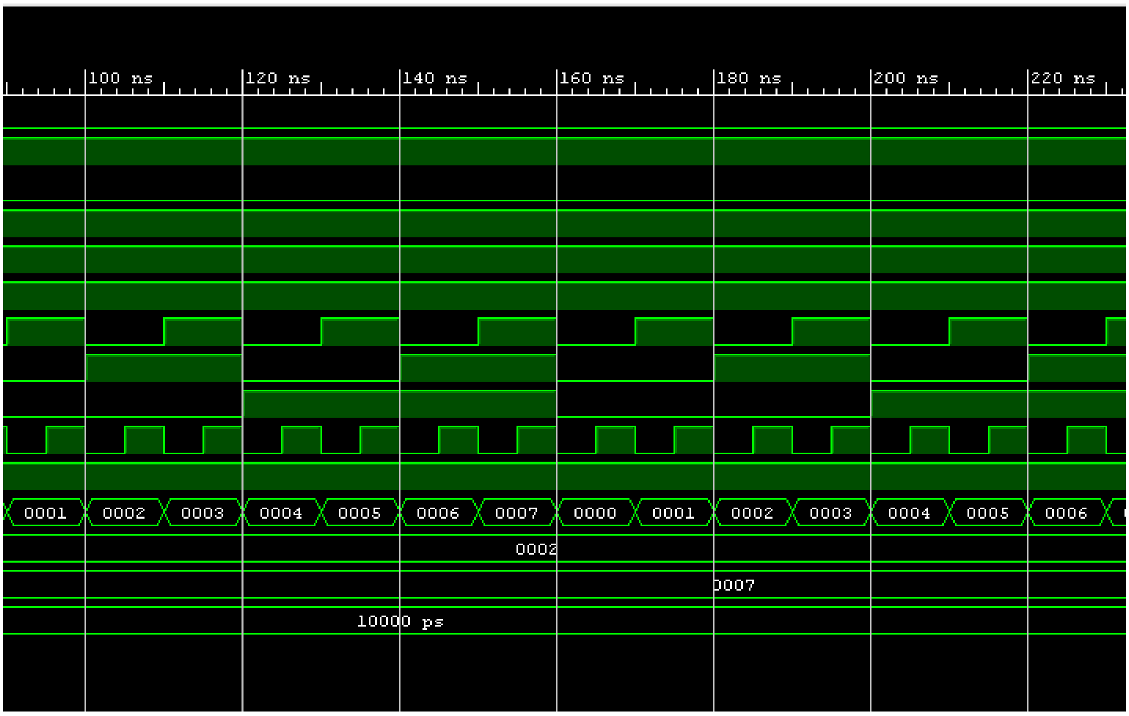


From the 16 bit shifter, it can be seen that the input is d_Vector = "8000" in hex. The shifter follows the following behaviour and logic:

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

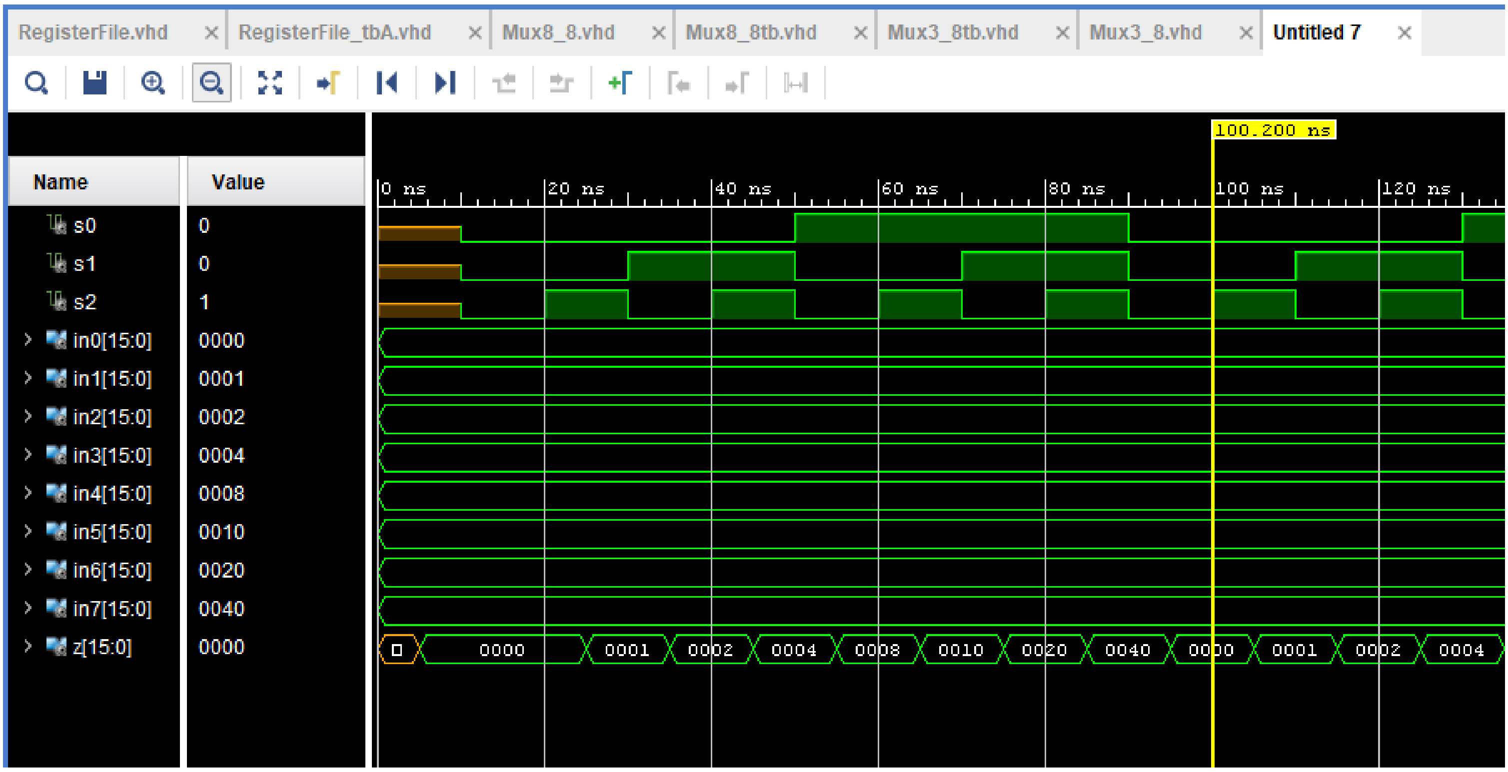
13 Register File – RegisterFile.vhd



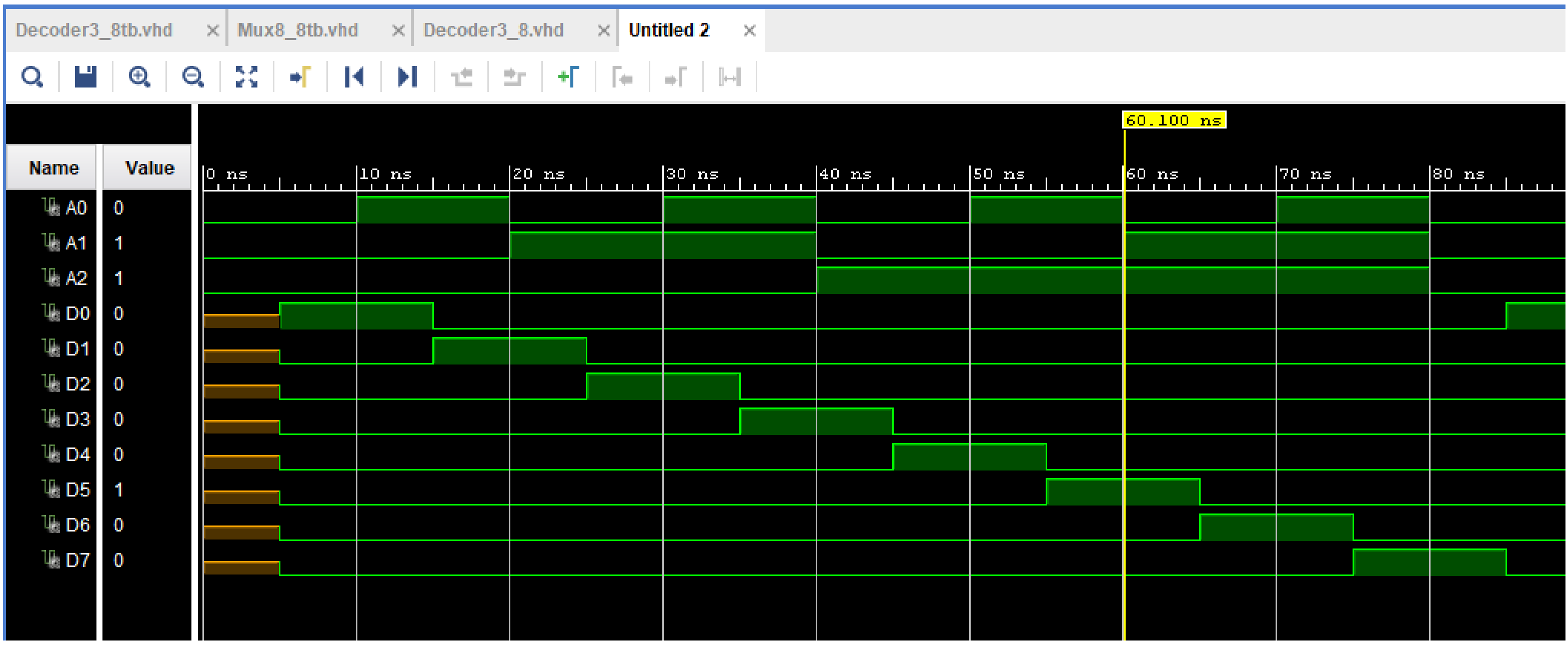


The above register file is updated with the second B Mux and outputs depending on Asrc and Bsrc.

14 MULTIPLEXER – MUX_8_8.VDH



15 DECODER – DECODER3_8.VDH



16 Single Register – reg16.vhd