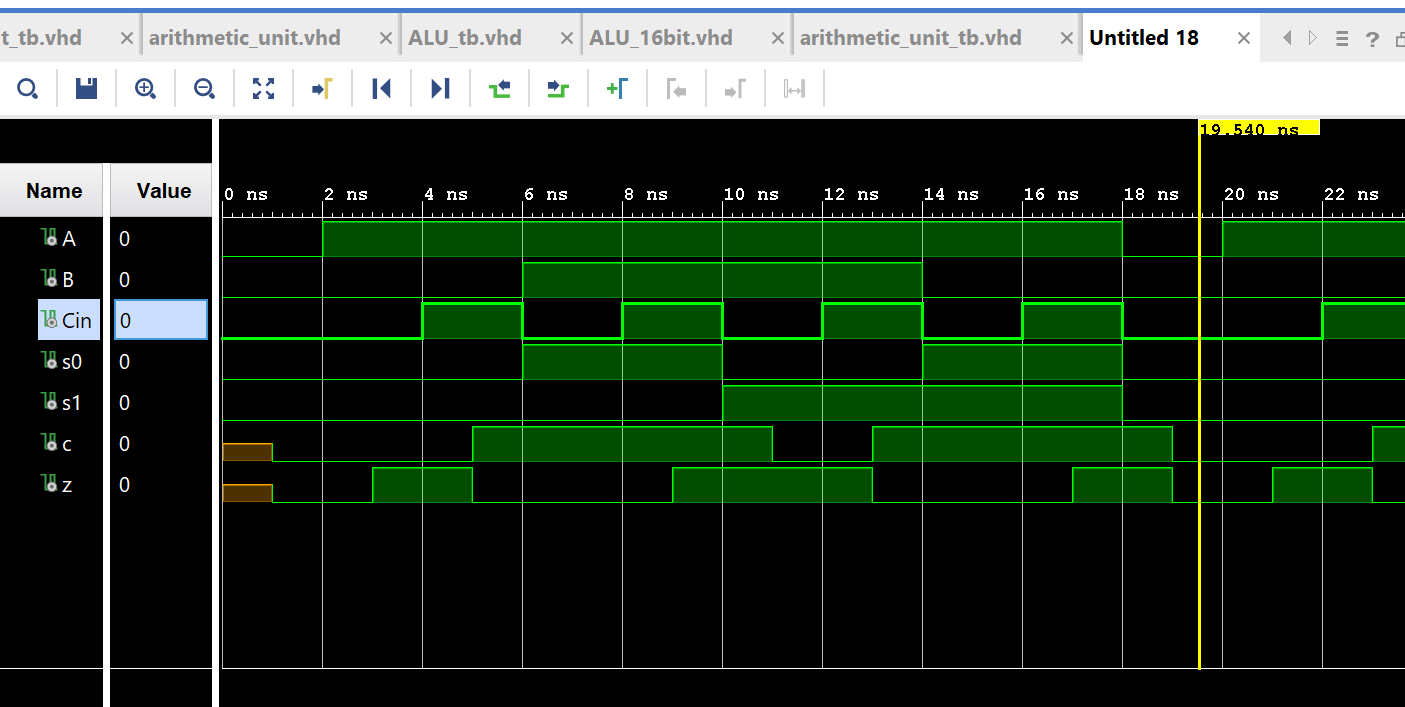
**Computer Architecture – Project 3 – Simulation Discussion – Killian Ronan - 18328687**

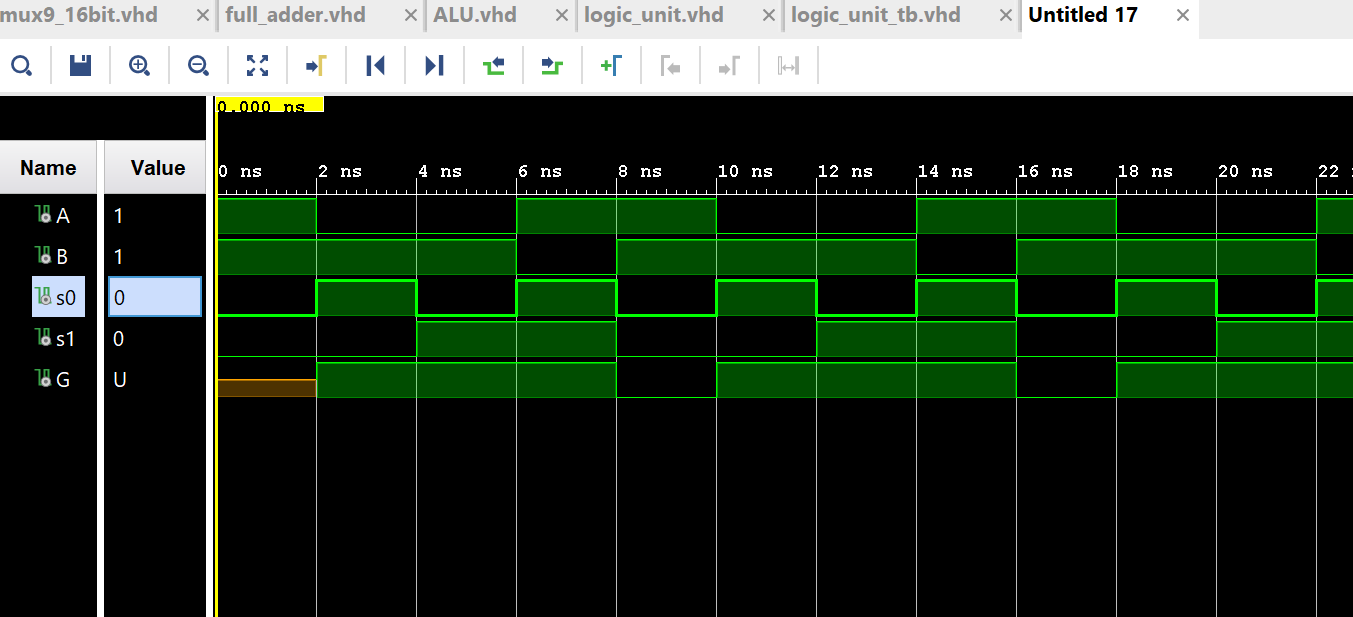
**Arithmetic Unit:**

Bitwise arithmetic operations are performed and tested. The output is represented by the z signal. A is loaded initially, (F= A) and incremented (F= A +1). All operations are tested and output correctly as can be seen below.

****

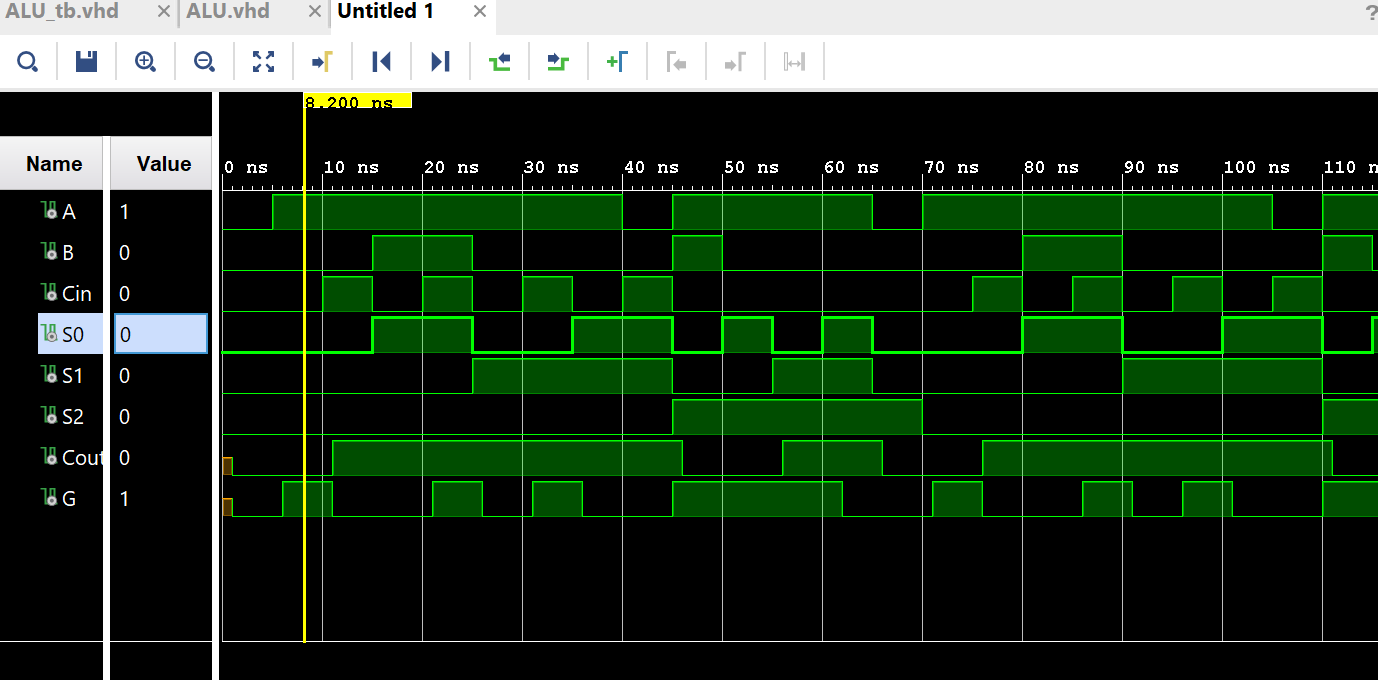
**Logic Unit:**

Bitwise logic operations are performed and tested. The output is represented by the z signal. The first operation is an AND operation of A and B. Next an OR operation is performed and so on until all operations were tested and outputted correctly as can be seen below.



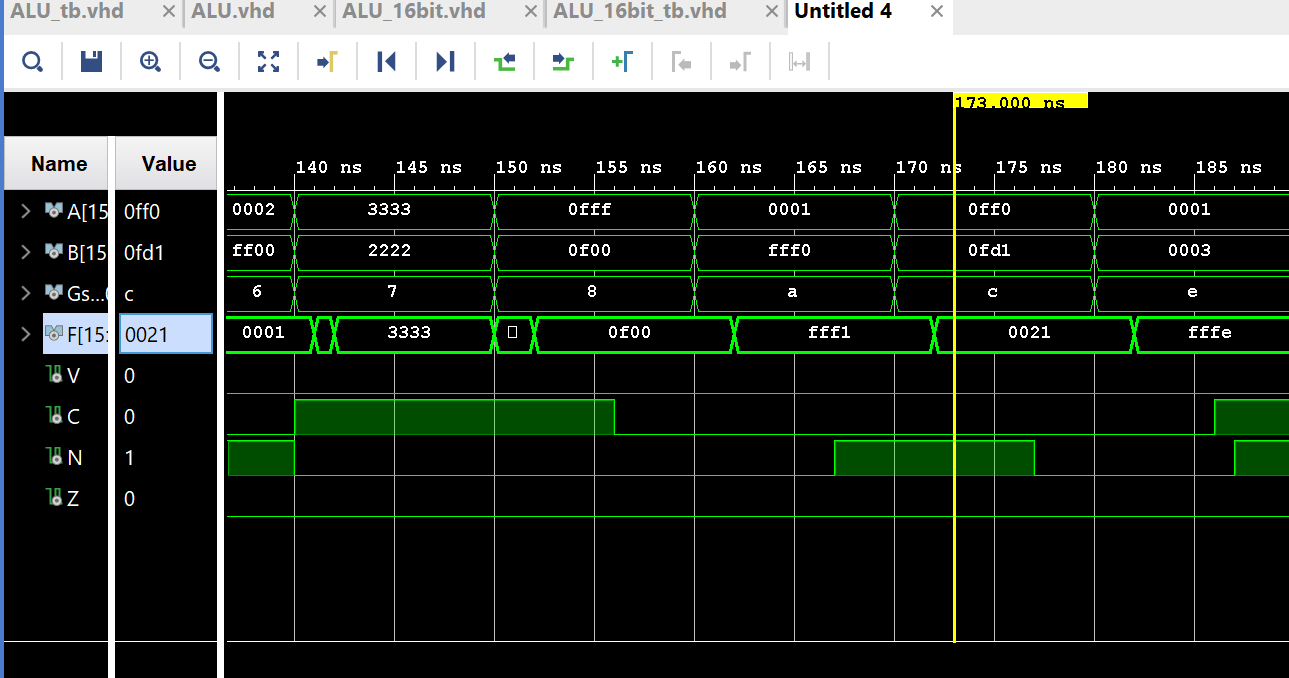
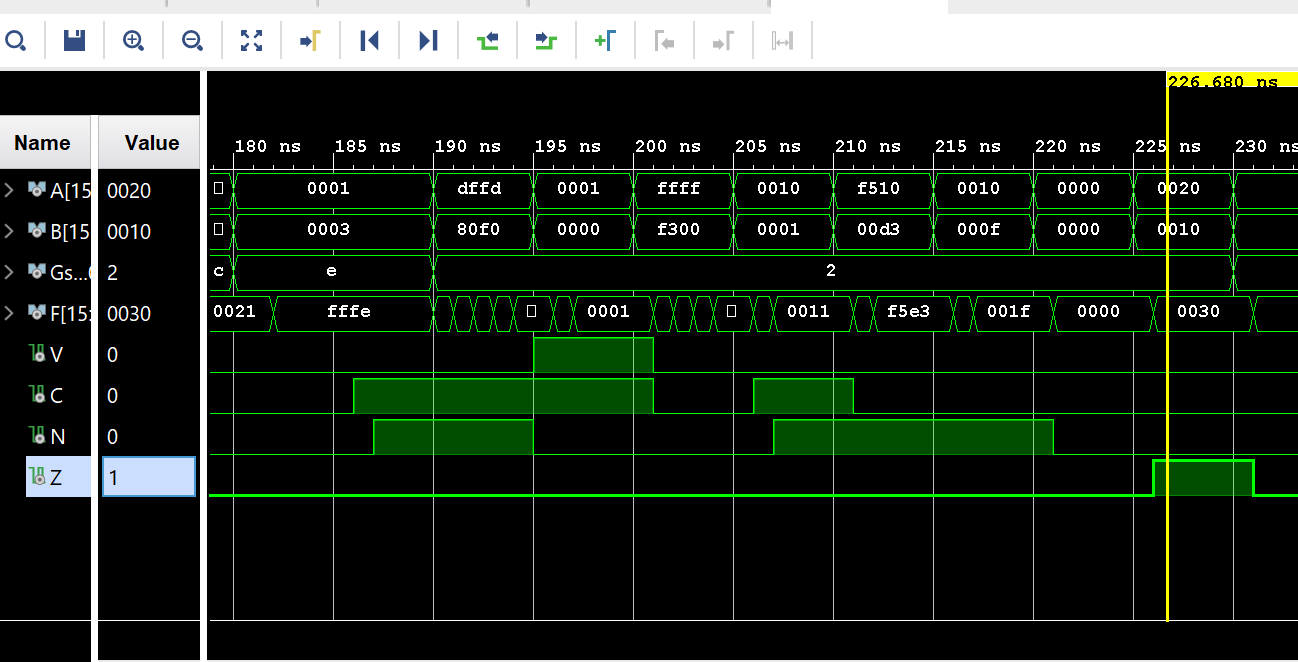
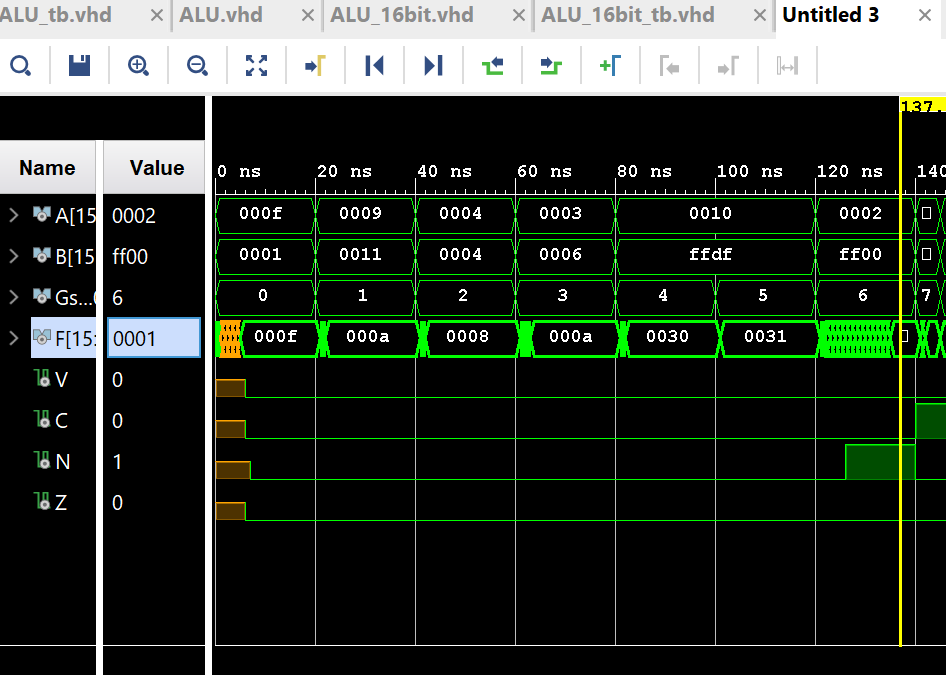
**ALU1bit:**

Tests basic arithmetic operations using the arithmetic unit, logic unit and a mux. Each arithmetic and logic operations are checked and outputted correctly such as loading, addition, incrementing , AND and OR operations. The correct results can be seen in the snip.

****

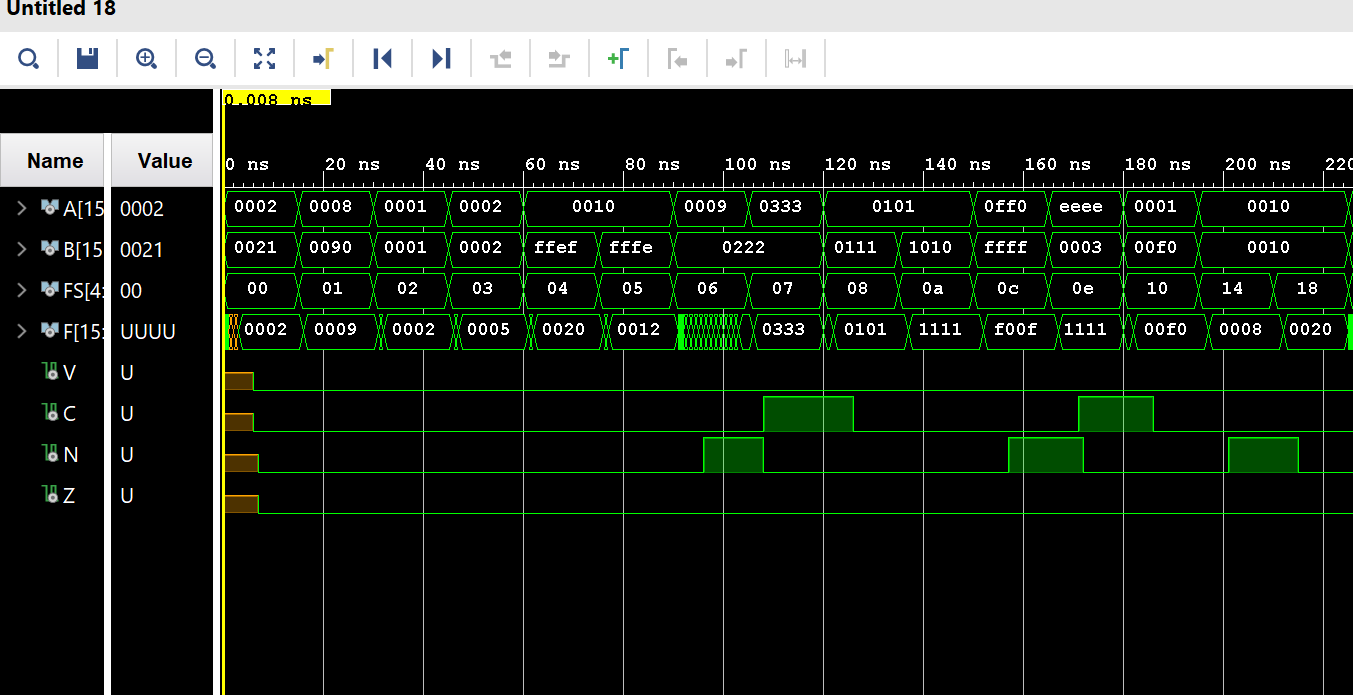
**ALU16 bit:**

Logic and Arithmetic tests were performed and can be seen below. The first loads in the input (F= A). The same operations as above are tested using 16 bit values. These operations are selected using the Gselect signal. The correct results are outputted to the F signal. Flag tests are also carried out.

****

**Functional Unit:**

All arithmetic, logic and shift tests are performed here. In the diagram below Arithmetic operations are performed until the input A changes to 0333. This is when logic operations are performed. Finally shift operations are tested and output correctly once A is assigned a value of 0001. The result is outputted to the F signal.

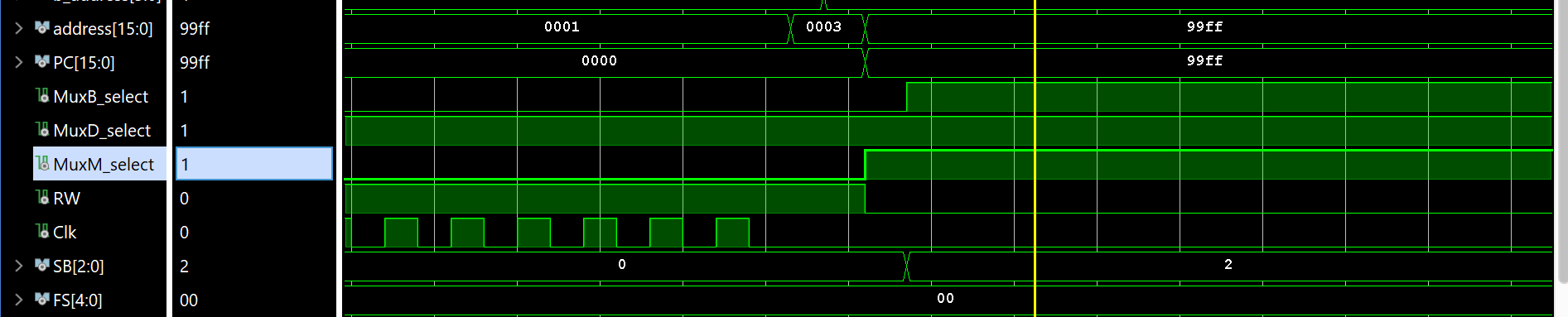
****

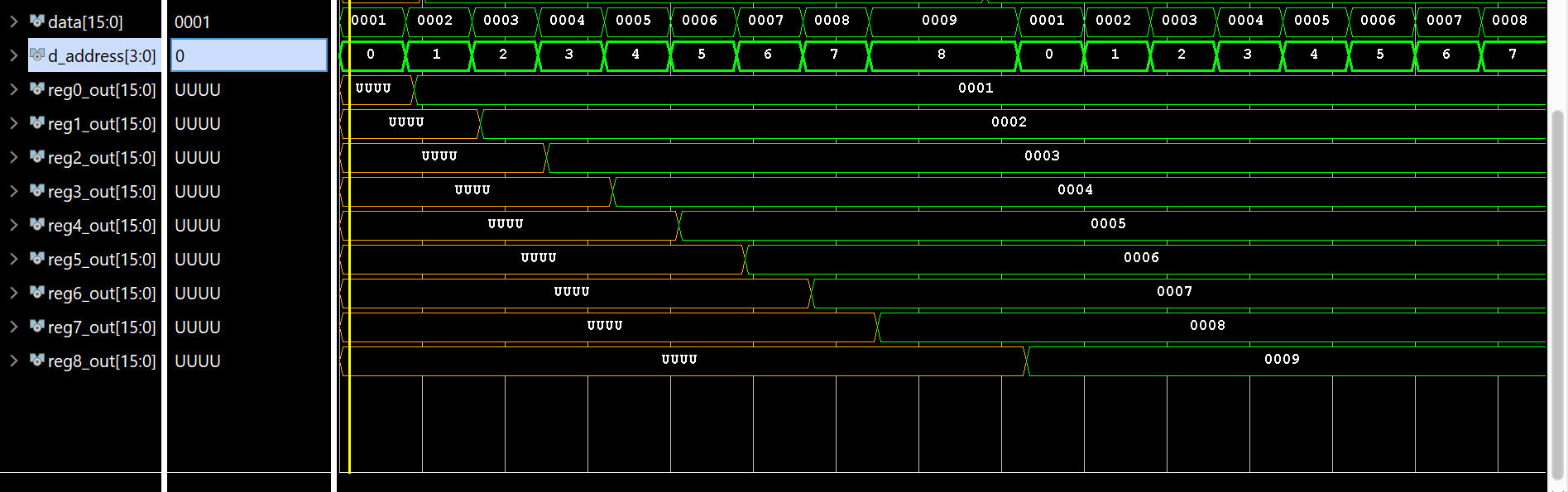
**Datapath:**

Registers are loaded with unique values one by one. The selected register to be written to is specified with d\_address and the unique input(data) is assigned to each of them.

Bus tests can be seen in the VHDL testbench code as register 2 is selected by bus A and register 4 is selected by bus B. The values from these registers are then correctly loaded.

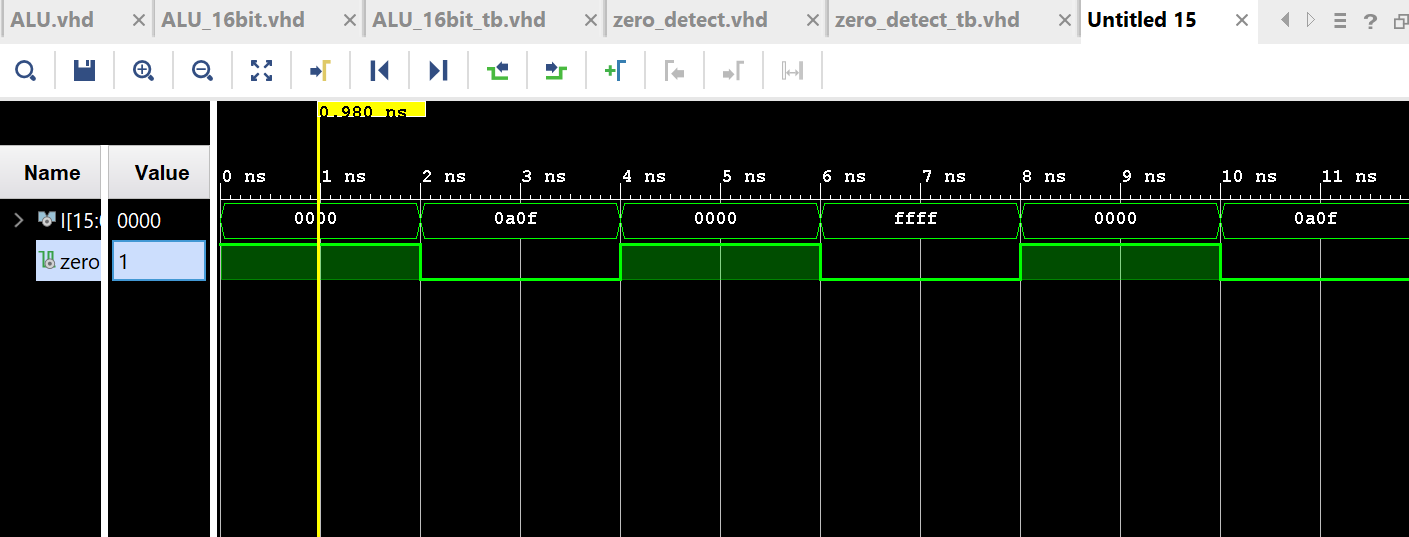
PC load is tested when the value 99FF is loaded from the PC signal. This can be seen in the address signal.

****

****

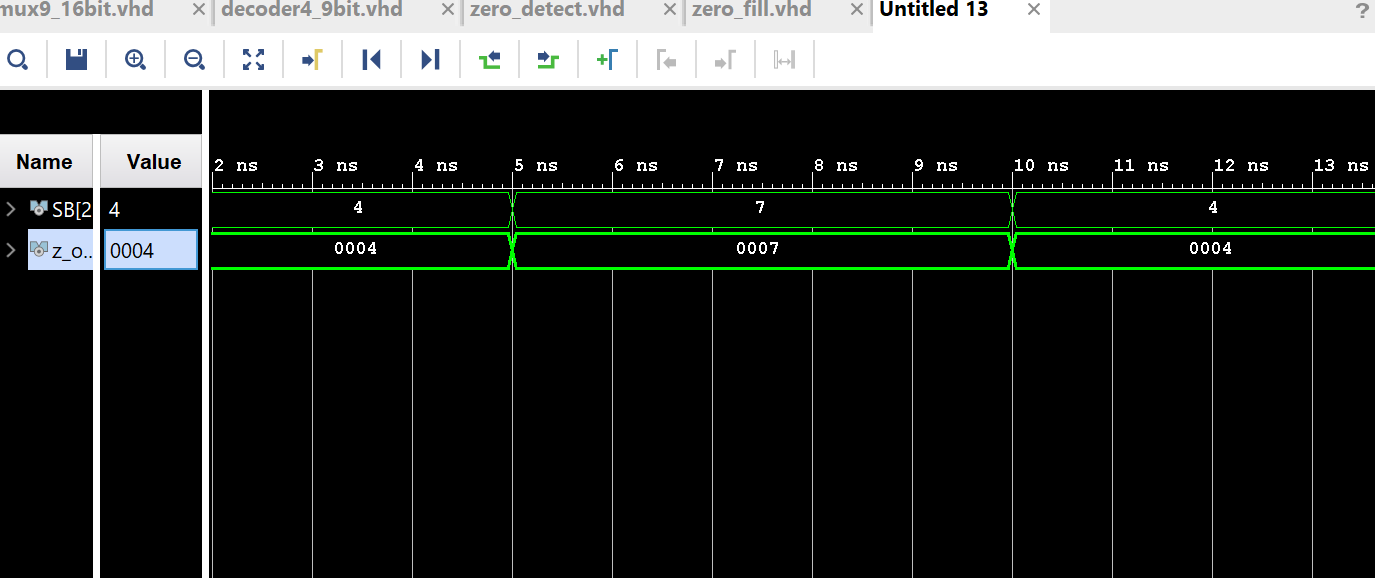
**zeroDetect:**

This component checks to see if the Z flag should be set if there is an input of value 0.



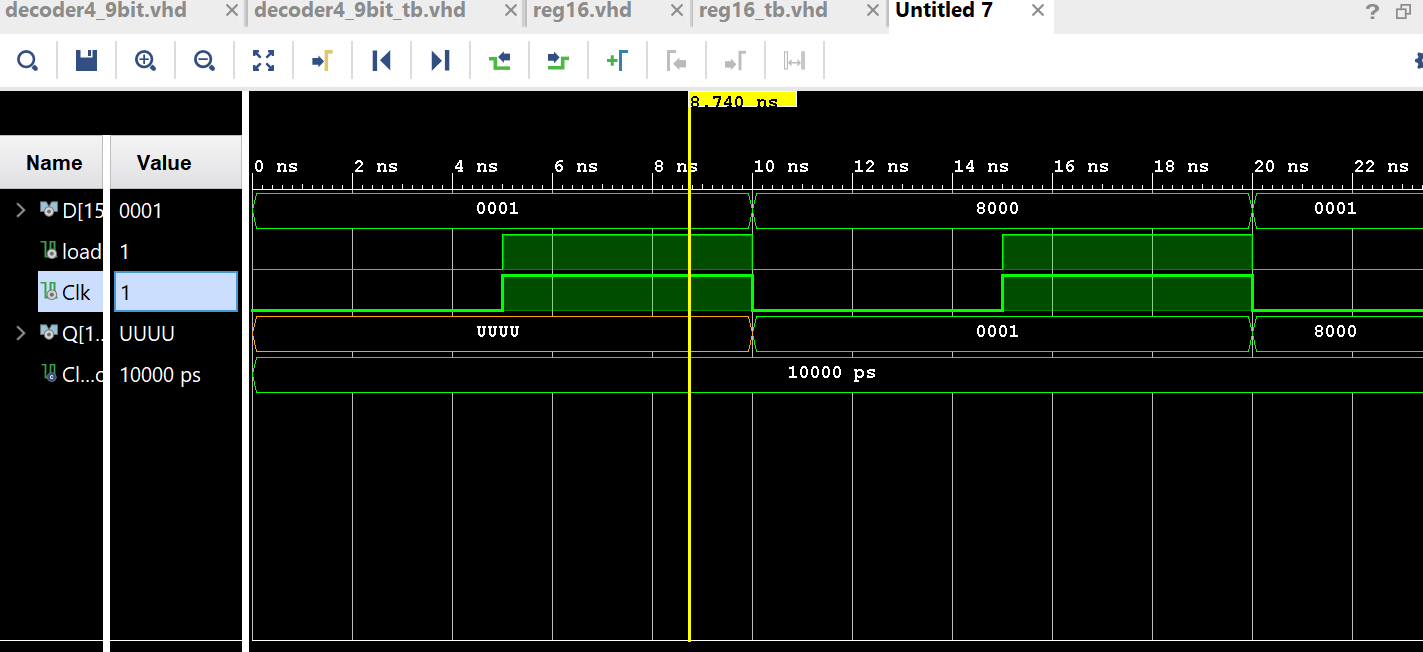
**zeroFill:**

This component returns a 16 bit vector based on a 3 bit input. It fills the remaining bits with 0’s.

****

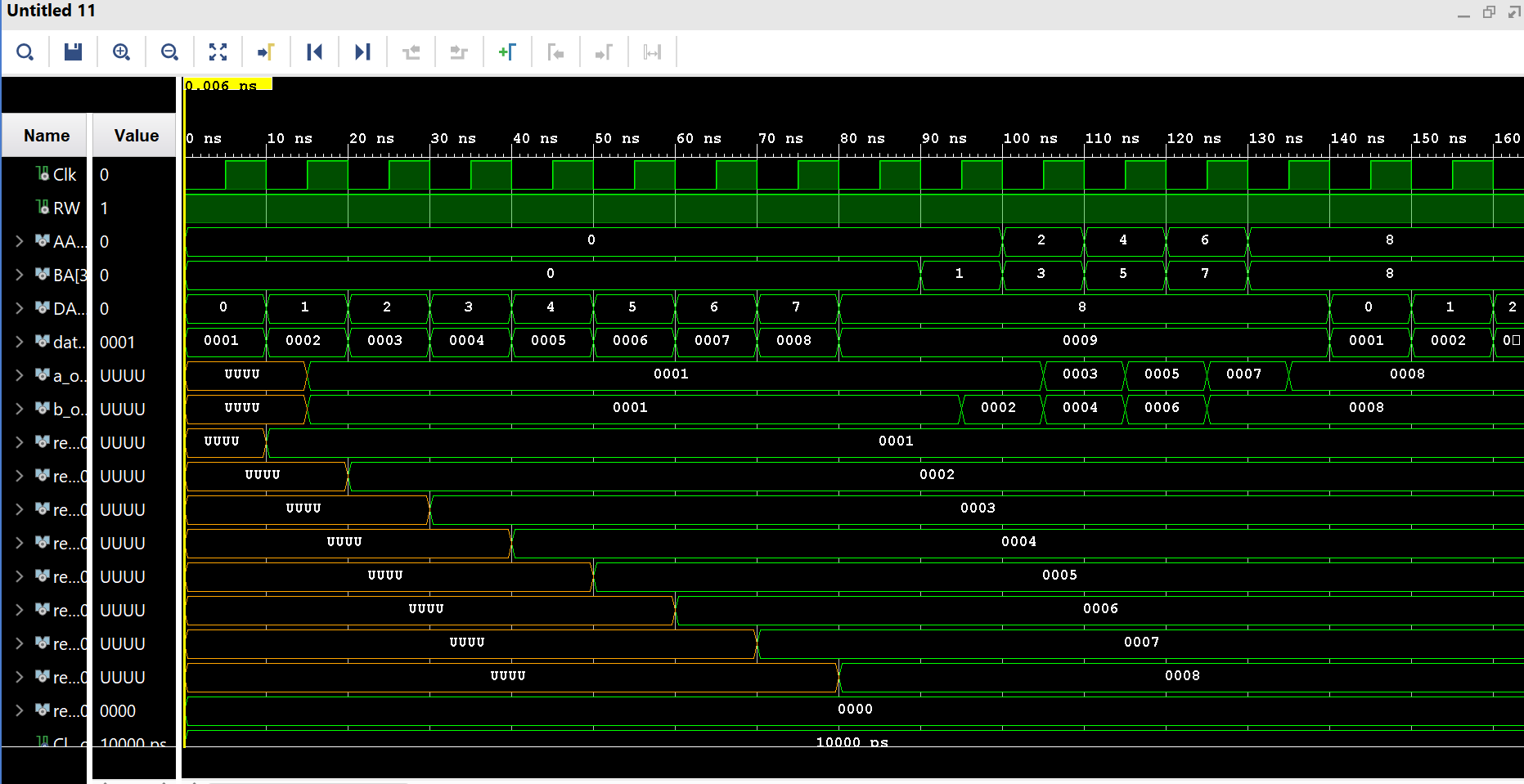
**Reg16:**

Register is loaded with new data based on whether the load and Clk variables are set. If they are, the output Q is set to the input specified(D).

****

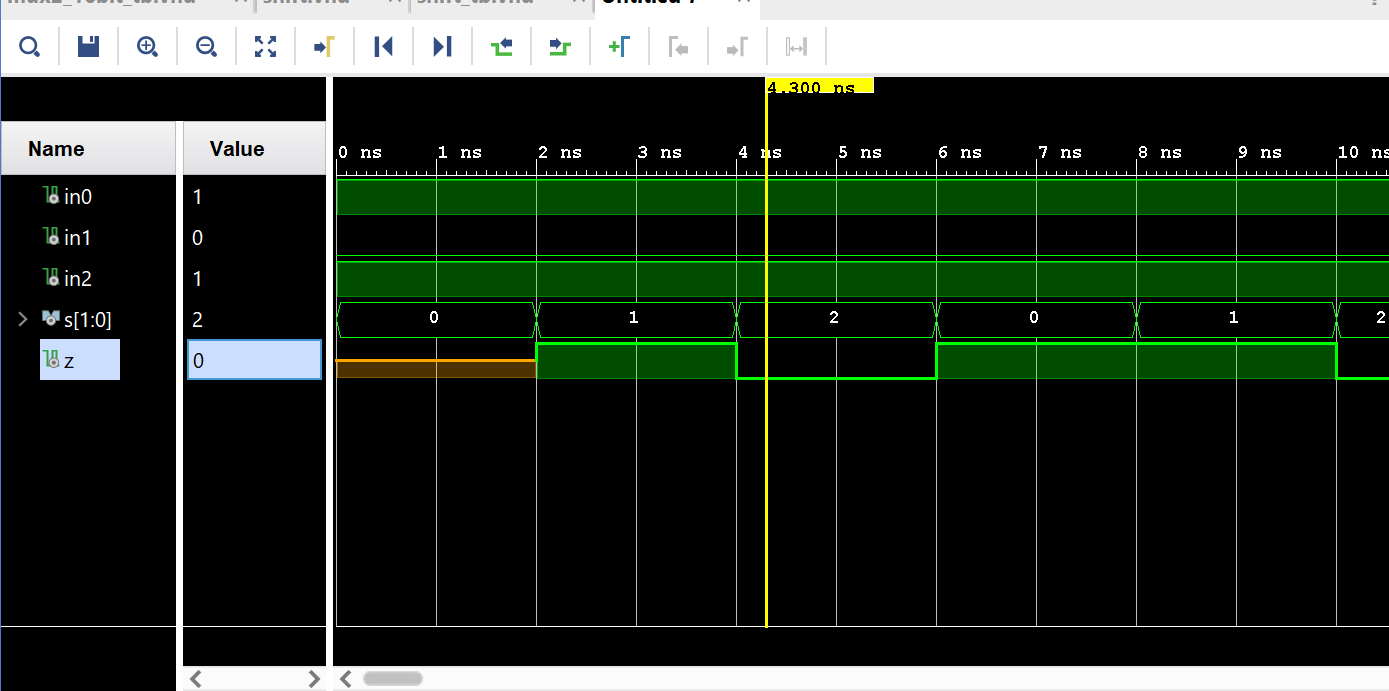
**Regfile:**

Registers are loaded with different values when the clock is set to 1. This waterfall style is implemented using cycles and can be seen in the snip. DA specifies the register currently being written to, data represents the input data being written to each register. Registers are selected using the AA and BA variables and the correct readings can be seen in the a\_out and b\_out signals.

****

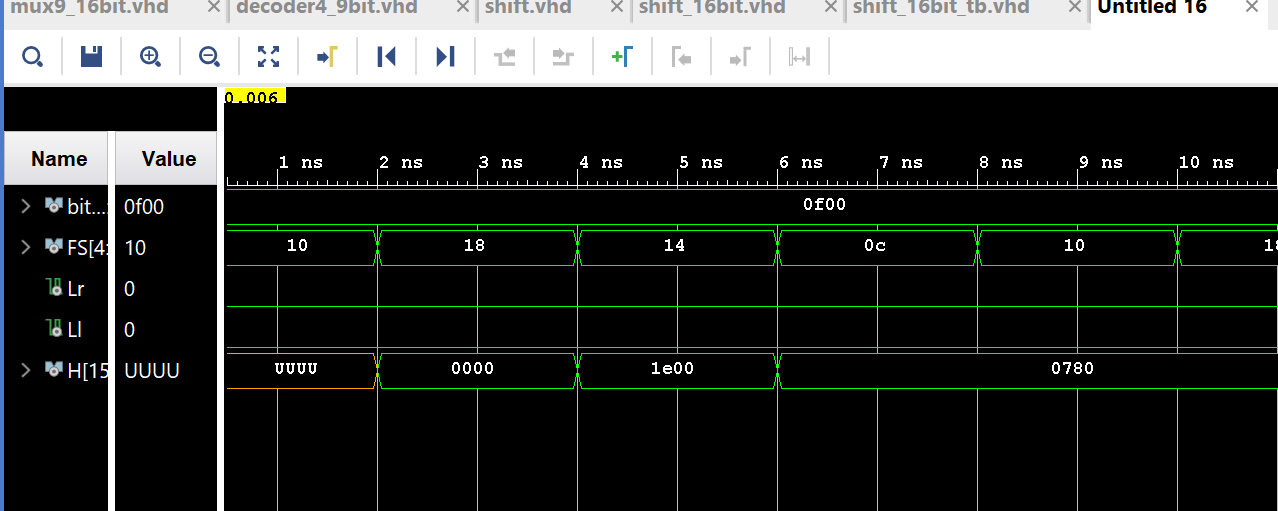
**Shifter:**

Shift operations on a 1 bit input provided based on the select input.

****

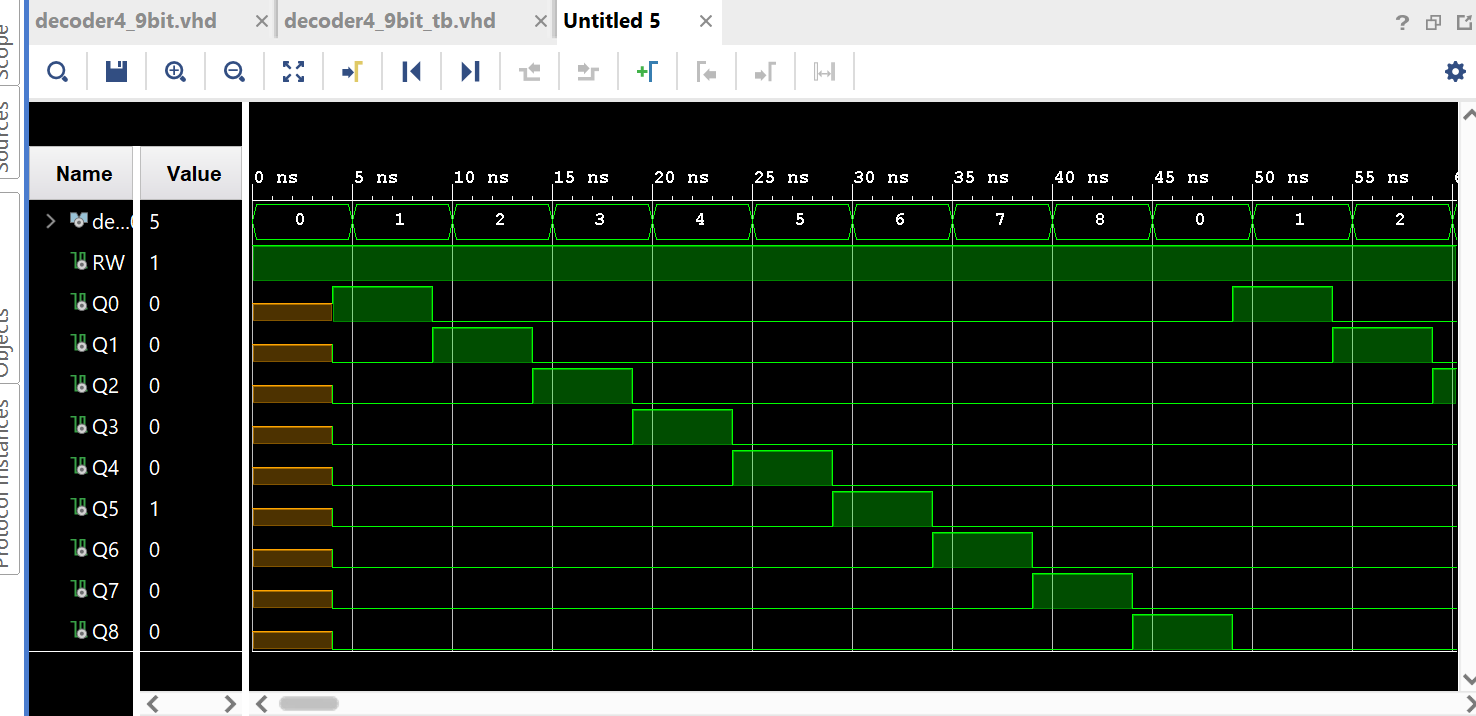
**Shifter16bit:**

Shift operations on a 16 bit input provided based on opcode input(FS). 1e00 is shifted right once LSR to form 0780.

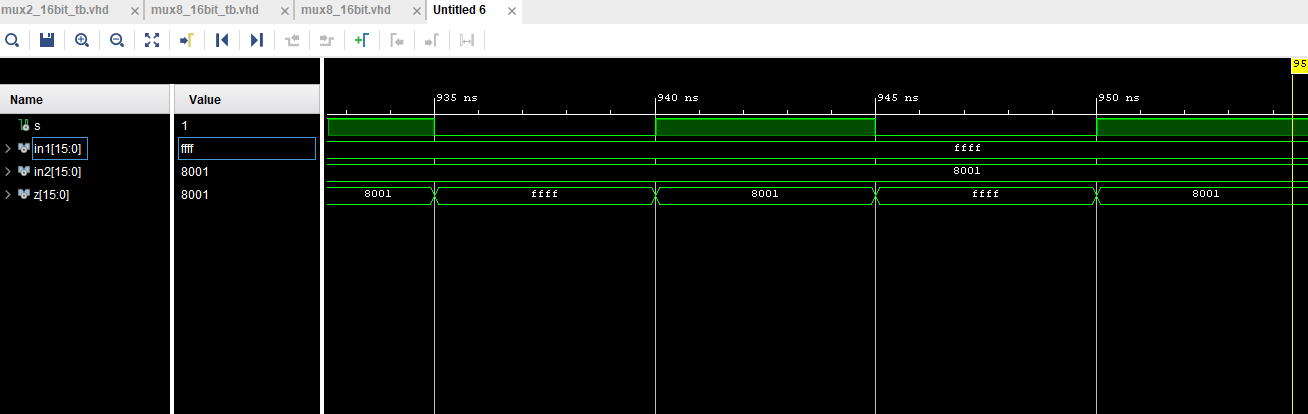
****

**Decoder3\_9bit:**

The decoder output depends on the des signal. As can be seen in the snip below when it is set to 0 it will output to Q0. When it is 1 it will output to Q1 etc. The output is also only set when the RW signal is set to 1 as can be seen below.

****

**Multiplexers:**

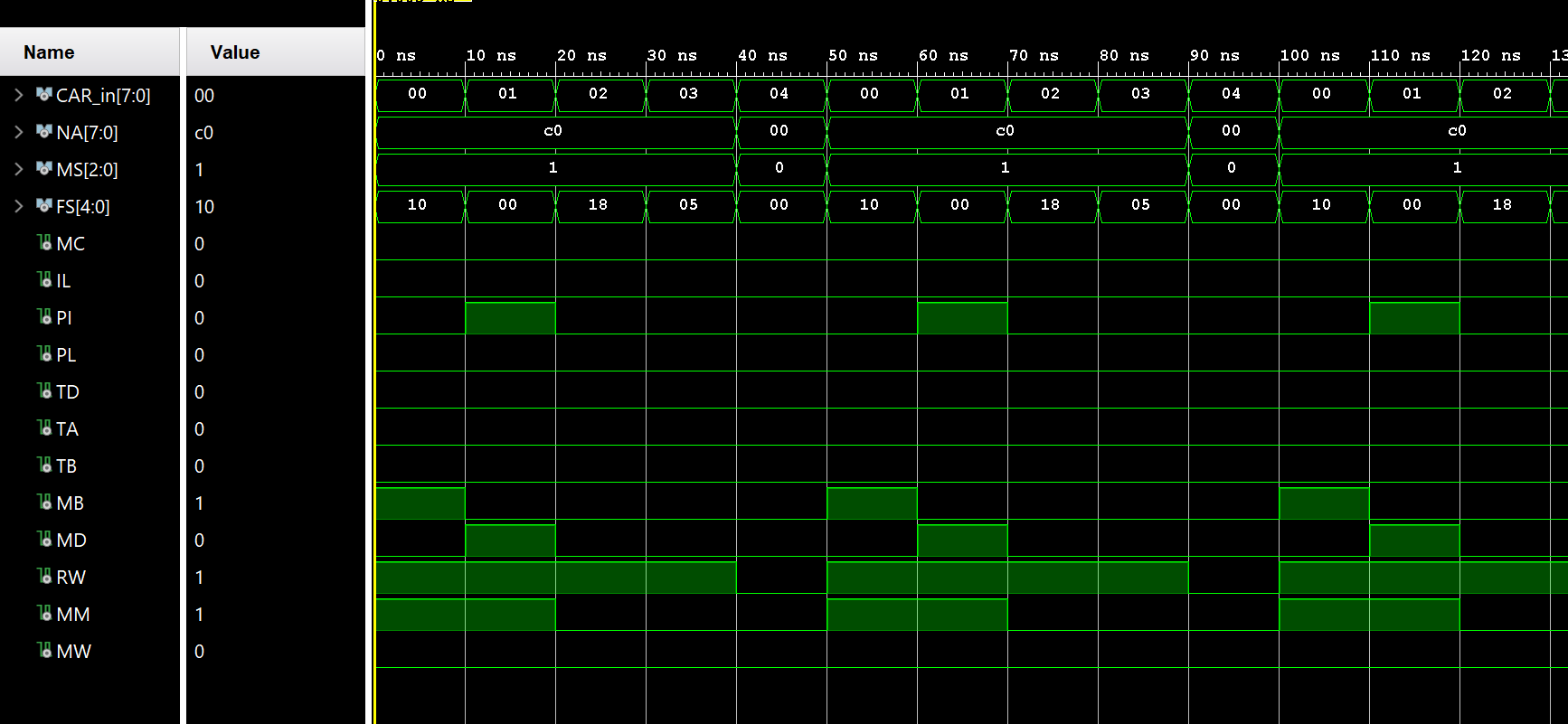
All multiplexers I designed and tested work based on the same principle. An output is chosen from the multiple inputs using a select variable. The Z output line is set to the input chosen with Select(S).

**Microprogrammed Controller:**

The Clk variable controls cycles and is set and unset to fetch and process requests. This process begins by setting reset and Clk which initialises the system to point to the memory location determined in the CAR component at 0xE0. The system retrieves the instruction at the CAR specified memory address. This instruction sets the Next Address flag to the next address which is 0xE0 + 0x1 = 0xE1. This tells the system to load the next address from the IR. We pass the address 0x0000 for which an Opcode is formatted in the Instruction Register which sets FS to ‘10000’ which can be seen in the simulation results.

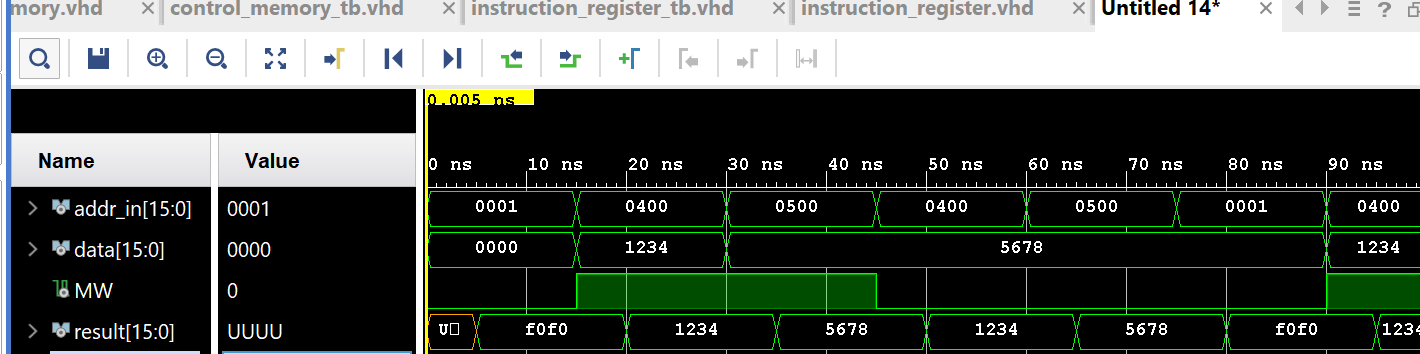
**Control memory:**

This testbench points to different memory locations starting at 0x0000 and incrementing. The values stored at each address are read in and formatted(decoded) by correctly setting the output fields. This is done by splitting the address into different chunks of bits, representing different output variables.



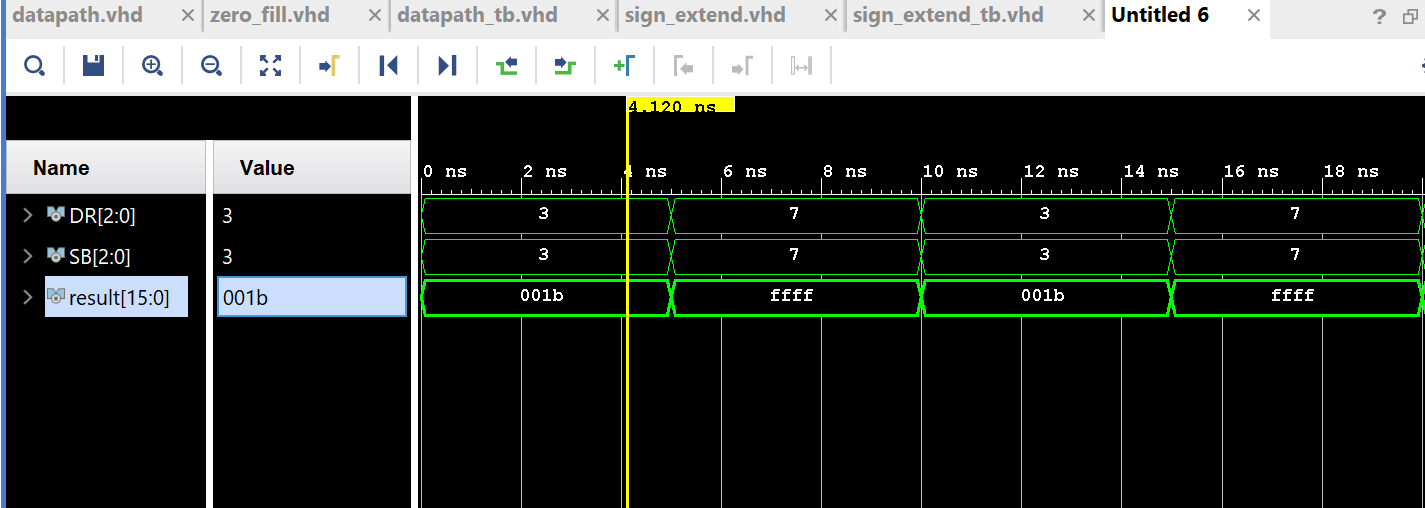
**Memory\_M:**

The value at address 0x0001 is loaded initially which is 0xF0F0. If the MW flag is set then the input value will be written in to memory. The value 0x1234 is written to memory address 0x4000 as can be seen in the snip. Another value is written to 0x5000. The MW flag is the set to ‘0’ mean no writing to memory can occur. When the input data changes, it is not written to memory. The value is just read from memory instead as can be seen with the previously written 0x1234 appearing in the output.



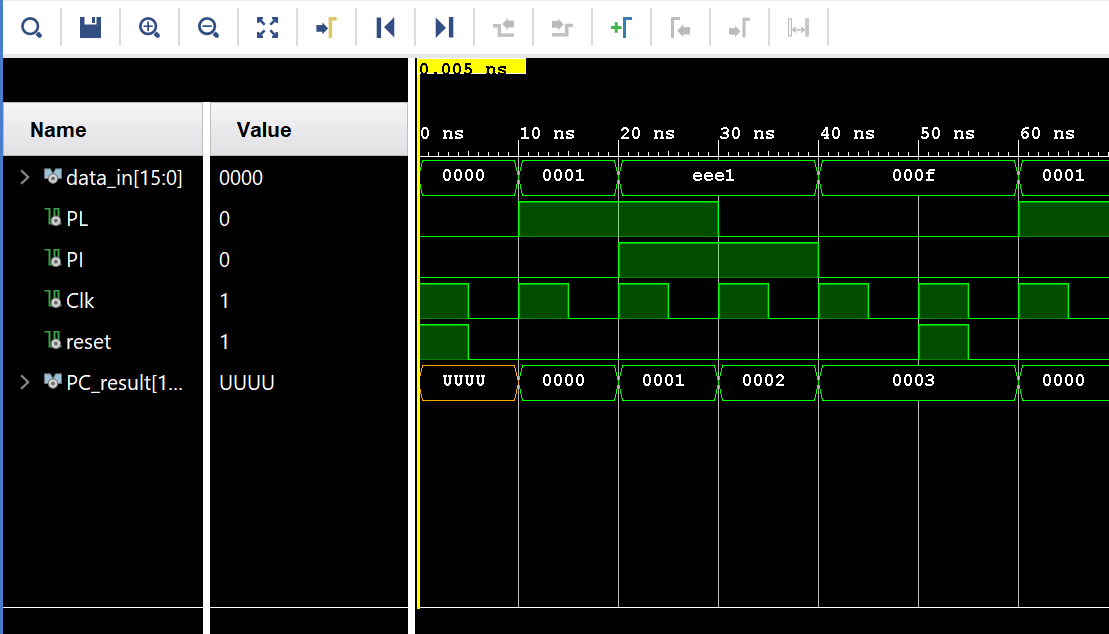
**Sign Extend:**

The sign extend component takes two 3 bit inputs of SB and DR and extends them based on the signed bit in DR. The resulting output will be filled with 1’s if the DR bit is signed and 0’s if not.



**Program Counter:**

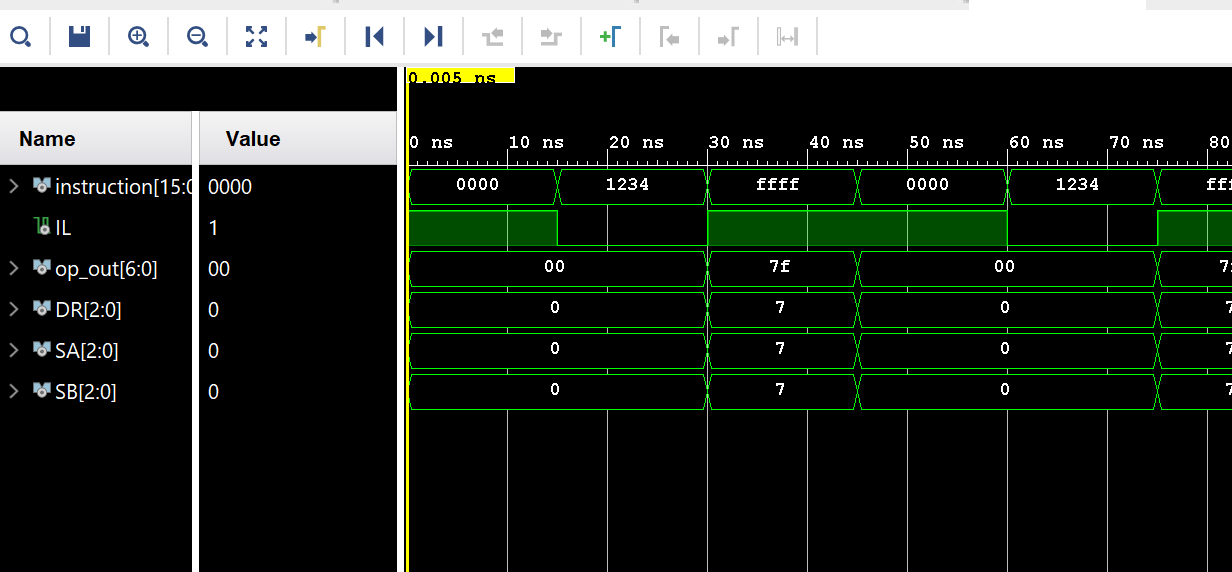
The PC is initialized by setting the reset and Clk variables to 1. This will reset the PC to 0x0000. If we set PI and not PL the PC result will increment. If we set PL and not PI the PC value is set to the input data provided. Setting both PI and PL will perform a branch operation adding the values 0 and 1 to output 1 for example in the snip provided.



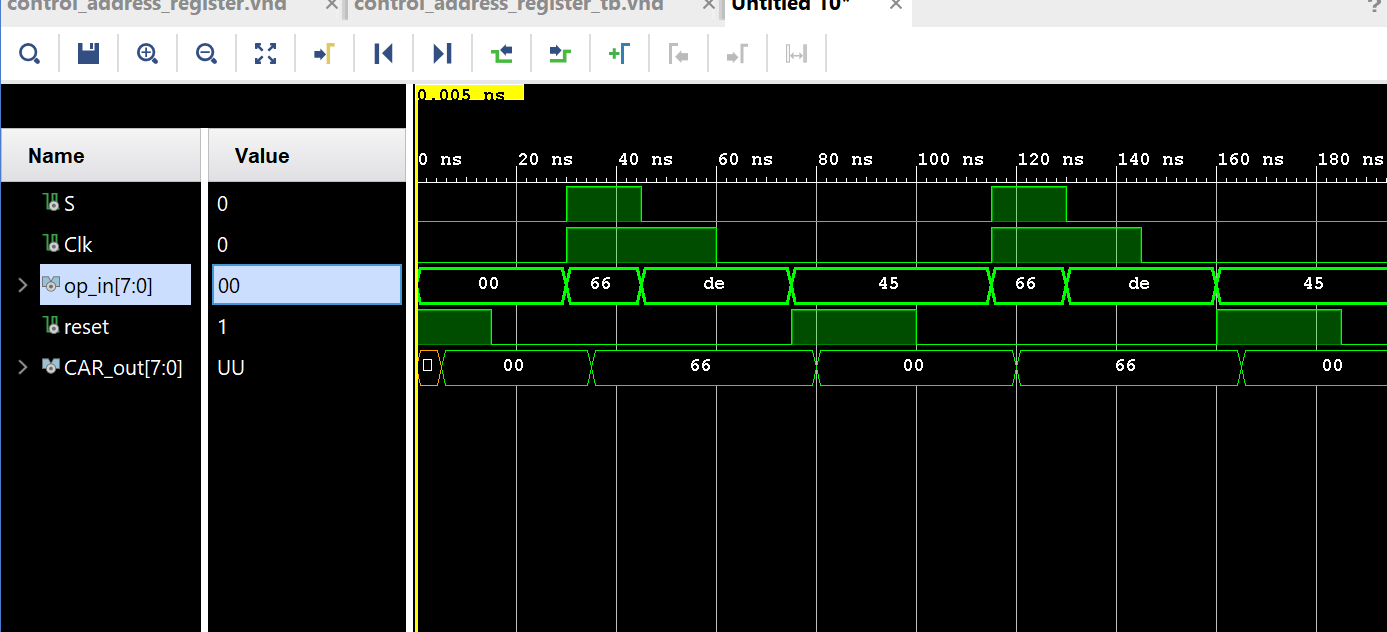
**Instruction Register:**

This component takes a 16 bit address as input and formats it correctly to form an opcode and set the DR, SA and SB variables. When IL is set to 0 it should not change the output as can be seen in the snip provided. I first pass 0 in my test bench, which changes each output respectively to 0. I then change the input but set the IL variable to 0 which means the output does not change. I then set the IL variable and change the input to FFFF. This is then formatted correctly as can be seen in the snip.

The IR should split the microprogram into the 3 bit SB,SA and DR registers and a 7 bit opcode.



**Control Address Register:**

The CAR is initialized by setting reset to 1. This points the CAR to the initial memory address. The output should only change when the MUXS input is set to 1. This can be seen as it changes the address stored to 66 when S is set and does not change the output to de when S is not set. Reset is tested again at the end of this testbench.

**CPU:**

Sadly, I could not get the full functionality of the project completed as I struggled to get this far alone. The high-level CPU was where I ran into problems and I feel if I had more time I would have been able to get a lot more functionality worked into it.

On initialization the reset did not set the system up properly. Only the first register was loaded meaning there must have been a mishap in my control\_memory. The NA signal may not have fed back the correct address.

