LAB REPORT - EXPERIMENT 3

COEN 316 – Computer Architecture – Winter 2025

Section WC - X

Lab performed on : March 5th 2025

Report submitted on : March 19th 2025

Lab performed and report written by : Simon Guindon – 40058301

TA Instructor:

Umair Hussain

Introduction and Objective

This experiment's objective is to introduce the experimenter to designing a next address unit. We will design the 32 bit version of it, as well as a sized down version so that the code would be physically testable on a Nexys A7 board.

Theory and Results

The usefulness of a next-address unit is its versatility to properly respond to a variety of opcodes. Depending on what is requested by the input, the system will properly respond to update the PC appropriately. For example, we can enter a 32 bits instruction to jump to a target address or to add/substract an offset to the PC when branching.

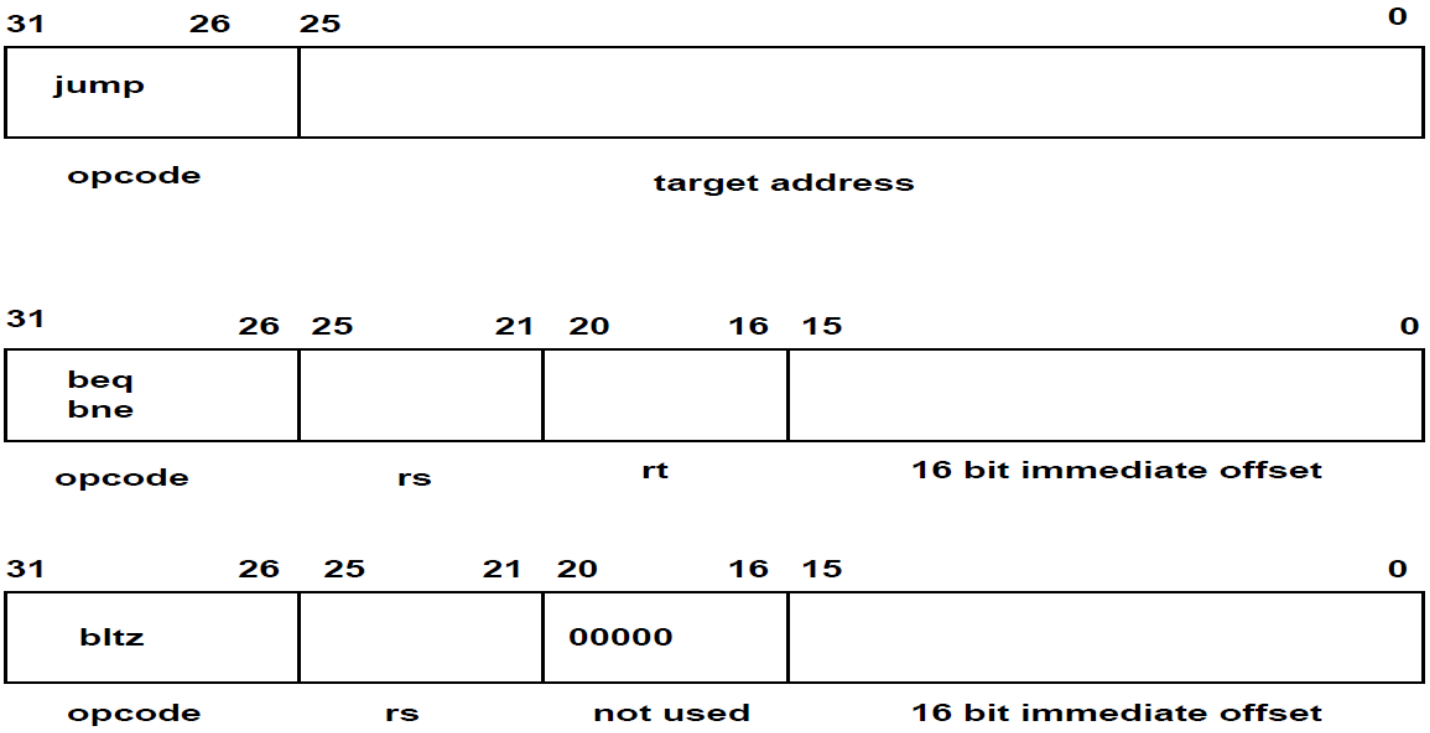


Figure 1: jump and branch instruction formats.

Results

Section 1.1 – Testing ModelSim's output for next_address.vhd

For this section, we need to test the following combinations:

PC_sel = 00 & branch_type = 00 (adds + 1 to PC)

PC_sel = 00 & branch_type = 01 & rs == rt (adds offset + 1 to PC)

PC_sel = 00 & branch_type = 01 & rs =/= rt (adds + 1 to PC)

PC_sel = 00 & branch_type = 10 & rs == rt (adds + 1 to PC)

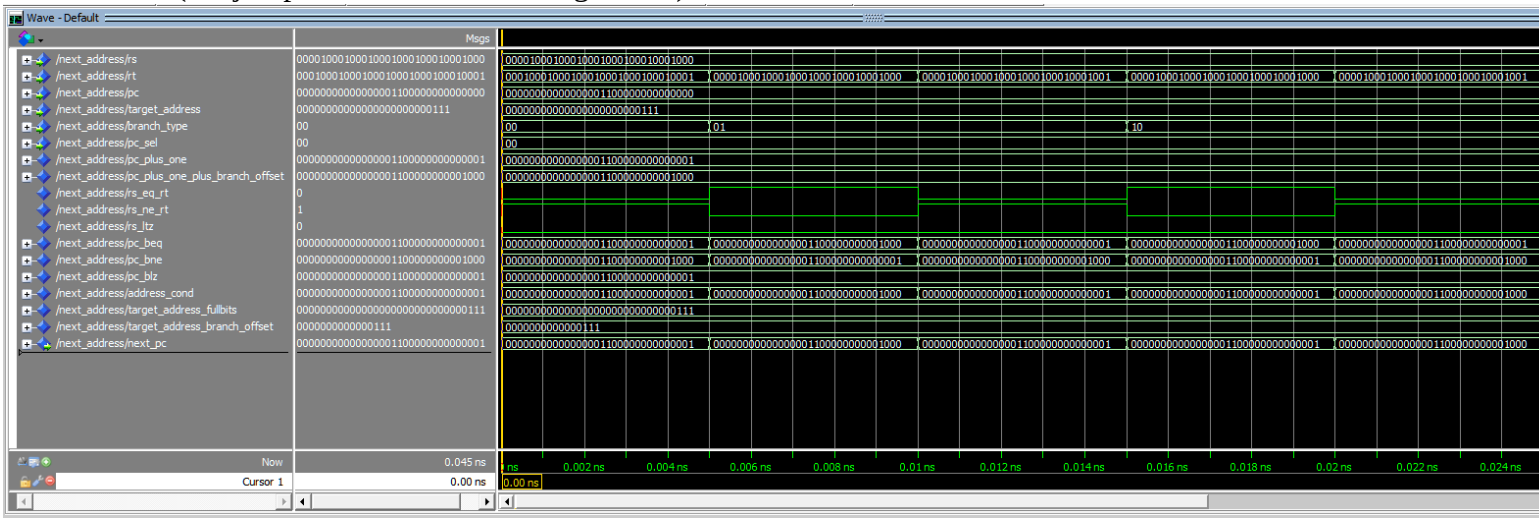
PC_sel = 00 & branch_type = 10 & rs =/= rt (adds offset + 1 to PC)

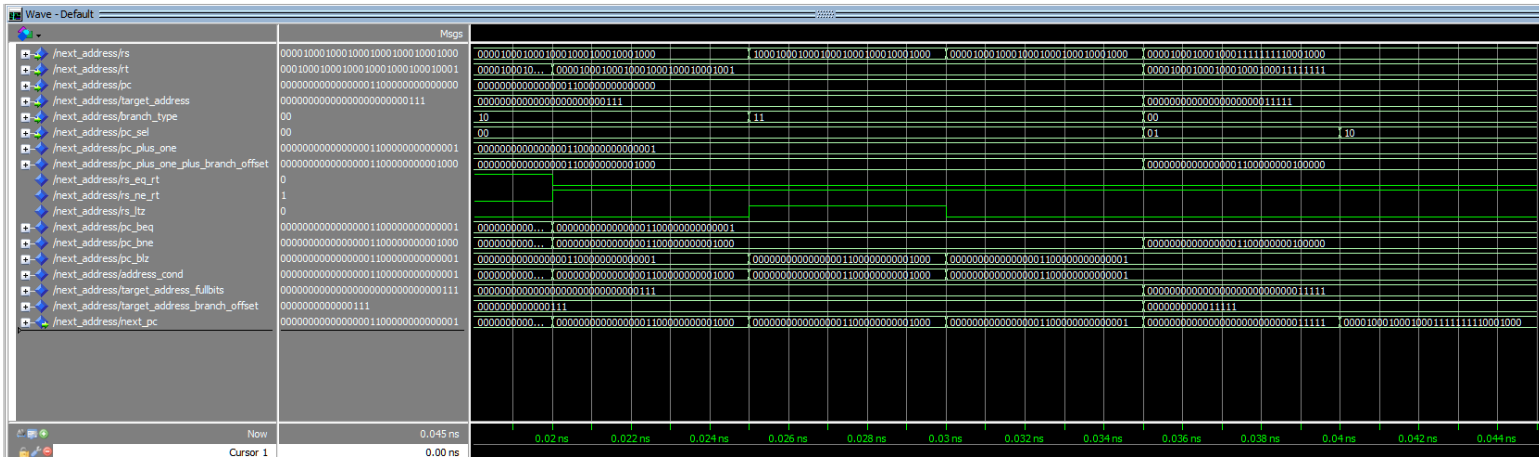
PC_sel = 00 & branch_type = 11 & rs < 0 (adds offset + 1 to PC)

PC_sel = 00 & branch_type = 11 & rs > 0 (adds + 1 to PC)

PC_sel = 01 (PC jumps to immediate value)

PC_sel = 10 (PC jumps to value stored in register rs)





We obtain the following:

PC_sel = 00 & branch_type = 00 (adds + 1 to PC)

Expected: 00000000000000001100000000000001 since pc is 00000000000000001100000000000000

Obtained: 00000000000000001100000000000001

PC_sel = 00 & branch_type = 01 & rs == rt (adds offset + 1 to PC)

Expected: 00000000000000001100000000001000 since offset is 111 and we add 1

Obtained: 00000000000000001100000000001000

PC_sel = 00 & branch_type = 01 & rs =/= rt (adds + 1 to PC)

Expected: 00000000000000001100000000000001
Obtained: 00000000000000001100000000000001
PC_sel = 00 & branch_type = 10 & rs == rt (adds + 1 to PC)

Expected: 00000000000000001100000000000001
Obtained: 00000000000000001100000000000001
PC_sel = 00 & branch_type = 10 & rs =/= rt (adds offset + 1 to PC)

Expected: 00000000000000001100000000001000
Obtained: 00000000000000001100000000001000
PC_sel = 00 & branch_type = 11 & rs < 0 (adds offset + 1 to PC)

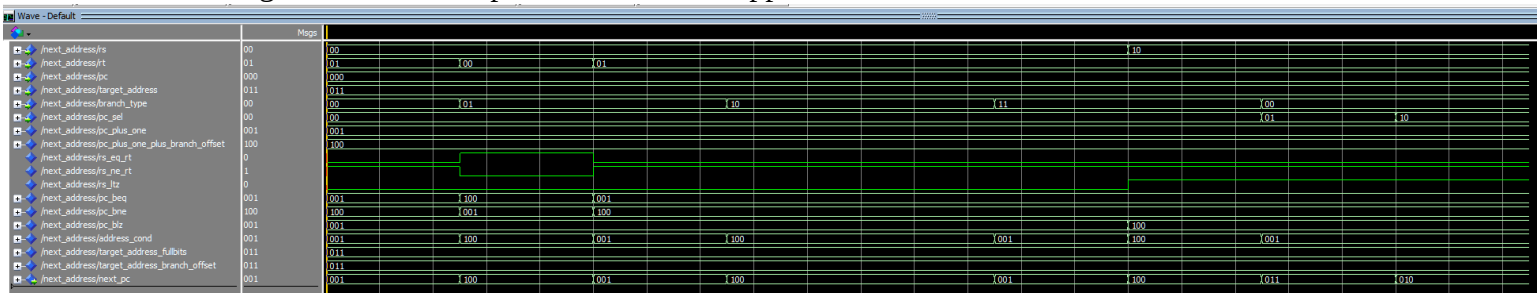
Expected: 00000000000000001100000000001000
Obtained: 00000000000000001100000000001000
PC_sel = 00 & branch_type = 11 & rs > 0 (adds + 1 to PC)

Expected: 00000000000000001100000000000001
Obtained: 00000000000000001100000000000001
PC_sel = 01 (PC jumps to immediate value)

Expected: 00000000000000000000000000011111
Obtained: 00000000000000000000000000011111
PC_sel = 10 (PC jumps to value stored in register rs)

Expected: 00001000100010001111111110001000 (value in rs)
Obtained: 00001000100010001111111110001000

Section 1.2 – Testing ModelSim's output for nextadd_wrapper.vhd



Similarly to above, we can see that the code works as intended, because the expected value matches what was obtained (thus reflects the results seen in the non-wrapper version). To avoid redundancy, the wrapper's results will not be explicitly listed below.

Section 2.1 – VHDL source code for next_address.vhd

next_address.vhd | next_address_wrapper.vhd

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity next_address is
    port(
        rs : in std_logic_vector(1 downto 0);
        rt : in std_logic_vector(1 downto 0);
        pc : in std_logic_vector(2 downto 0);
        target_address : in std_logic_vector(2 downto 0);
        branch_type : in std_logic_vector(1 downto 0);
        pc_sel : in std_logic_vector(1 downto 0);
        next_pc : out std_logic_vector(2 downto 0)
    );
end next_address;

architecture arch_next_address of next_address is

    signal pc_plus_one : std_logic_vector(2 downto 0);
    signal pc_plus_one_plus_branch_offset : std_logic_vector(2 downto 0);
    signal rs_eq_rt : std_logic;
    signal rs_ne_rt : std_logic;
    signal rs_ltz : std_logic;
    signal pc_beq : std_logic_vector(2 downto 0);
    signal pc_bne : std_logic_vector(2 downto 0);
    signal pc_blz : std_logic_vector(2 downto 0);
    signal address_cond : std_logic_vector(2 downto 0);
    signal target_address_fullbits : std_logic_vector(2 downto 0);
    signal target_address_branch_offset : std_logic_vector(2 downto 0);

begin
    pc_plus_one <= std_logic_vector(unsigned(pc) + 1);
    target_address_branch_offset <= target_address(2 downto 0);
    pc_plus_one_plus_branch_offset <= std_logic_vector(unsigned(pc) + 1 + unsigned(target_address_branch_offset));
    target_address_fullbits <=  target_address;

    rs_eq_rt <= '1' when rs = rt else '0';
    rs_ne_rt <= '1' when rs /= rt else '0';
    rs_ltz <= '1' when rs(1) = '1' else '0';
```

First, I declare internal signals to help with the wiring. I assign pc_plus_one to pc plus adding 1, take the first 16 bits of target_address to obtain the offset, add the offset and 1 to the pc for pc plus one plus offset and add 6 zeroes to target address to obtain a 32 bits value.

next_address.vhd | next_address_wrapper.vhd

```
    rs_ltz <= '1' when rs(1) = '1' else '0';

    process(pc_plus_one, pc_plus_one_plus_branch_offset, rs_eq_rt)
    begin
        if rs_eq_rt = '1' then
            pc_beq <= pc_plus_one_plus_branch_offset;
        else
            pc_beq <= pc_plus_one;
        end if;
    end process;
    process(pc_plus_one, pc_plus_one_plus_branch_offset, rs_ne_rt)
    begin
        if rs_ne_rt = '1' then
            pc_bne <= pc_plus_one_plus_branch_offset;
        else
            pc_bne <= pc_plus_one;
        end if;
    end process;
    process(pc_plus_one, pc_plus_one_plus_branch_offset, rs_ltz)
    begin
        if rs_ltz = '1' then
            pc_blz <= pc_plus_one_plus_branch_offset;
        else
            pc_blz <= pc_plus_one;
        end if;
    end process;
    process(pc_plus_one, pc_beq, pc_bne, pc_blz, branch_type)
    begin
        case branch_type is
            when "00" => address_cond <= pc_plus_one;
            when "01" => address_cond <= pc_beq;
            when "10" => address_cond <= pc_bne;
            when others => address_cond <= pc_blz;
        end case;
    end process;
    process(address_cond, target_address_fullbits, rs, pc_sel)
    begin
        case pc_sel is
            when "00" => next_pc <= address_cond;
            when "01" => next_pc <= target_address_fullbits;
            when others => next_pc <= "0" & rs;
        end case;
    end process;
end arch_next_address;

```

Section 2.2 – VHDL source code for next_address_wrapper.vhd

next_address.vhd | next_address_wrapper.vhd

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity next_address is
    port(
        rs : in std_logic_vector(1 downto 0);
        rt : in std_logic_vector(1 downto 0);
        pc : in std_logic_vector(2 downto 0);
        target_address : in std_logic_vector(2 downto 0);
        branch_type : in std_logic_vector(1 downto 0);
        pc_sel : in std_logic_vector(1 downto 0);
        next_pc : out std_logic_vector(2 downto 0)
    );
end next_address;

architecture arch_next_address of next_address is

    signal pc_plus_one : std_logic_vector(2 downto 0);
    signal pc_plus_one_plus_branch_offset : std_logic_vector(2 downto 0);
    signal rs_eq_rt : std_logic;
    signal rs_ne_rt : std_logic;
    signal rs_ltz : std_logic;
    signal pc_beq : std_logic_vector(2 downto 0);
    signal pc_bne : std_logic_vector(2 downto 0);
    signal pc_blz : std_logic_vector(2 downto 0);
    signal address_cond : std_logic_vector(2 downto 0);
    signal target_address_fullbits : std_logic_vector(2 downto 0);
    signal target_address_branch_offset : std_logic_vector(15 downto 0);

begin
    pc_plus_one <= std_logic_vector(unsigned(pc) + 1);
    target_address_branch_offset <= target_address(2 downto 0);
    pc_plus_one_plus_branch_offset <= std_logic_vector(unsigned(pc) + 1 + unsigned(target_address_branch_offset));
    target_address_fullbits <=  target_address;

    rs_eq_rt <= '1' when rs = rt else '0';
    rs_ne_rt <= '1' when rs /= rt else '0';
    rs_ltz <= '1' when rs(31) = '1' else '0';

    process(pc_plus_one, pc_plus_one_plus_branch_offset, rs_eq_rt)
    begin
        if rs_eq_rt = '1' then
            pc_beq <= pc_plus_one_plus_branch_offset;
```

next_address.vhd | next_address_wrapper.vhd

```
    rs_ltz <= '1' when rs(31) = '1' else '0';

    process(pc_plus_one, pc_plus_one_plus_branch_offset, rs_eq_rt)
    begin
        if rs_eq_rt = '1' then
            pc_beq <= pc_plus_one_plus_branch_offset;
        else
            pc_beq <= pc_plus_one;
        end if;
    end process;
    process(pc_plus_one, pc_plus_one_plus_branch_offset, rs_ne_rt)
    begin
        if rs_ne_rt = '1' then
            pc_bne <= pc_plus_one_plus_branch_offset;
        else
            pc_bne <= pc_plus_one;
        end if;
    end process;
    process(pc_plus_one, pc_plus_one_plus_branch_offset, rs_ltz)
    begin
        if rs_ltz = '1' then
            pc_blz <= pc_plus_one_plus_branch_offset;
        else
            pc_blz <= pc_plus_one;
        end if;
    end process;
    process(pc_plus_one, pc_beq, pc_bne, pc_blz, branch_type)
    begin
        case branch_type is
            when "00" => address_cond <= pc_plus_one;
            when "01" => address_cond <= pc_beq;
            when "10" => address_cond <= pc_bne;
            when others => address_cond <= pc_blz;
        end case;
    end process;
    process(address_cond, target_address_fullbits, rs, pc_sel)
    begin
        case pc_sel is
            when "00" => next_pc <= address_cond;
            when "01" => next_pc <= target_address_fullbits;
            when others => next_pc <= rs;
        end case;
    end process;
end arch_next_address;

```

Section 3.1 – Do file for next_address.do

```
add wave rs
add wave rt
add wave pc
add wave target_address
add wave branch_type
add wave pc_sel

add wave pc_plus_one
add wave pc_plus_one_plus_branch_offset

add wave rs_eq_rt
add wave rs_ne_rt
add wave rs_ltz

add wave pc_beq
add wave pc_bne
add wave pc_blz

add wave address_cond
add wave target_address_fullbits
add wave target_address_branch_offset

add wave next_pc

force rs 00001000100010001000100010001000
force rt 00010001000100010001000100010001
force pc 00000000000000001100000000000000
force target_address 00000000000000000000000111
force branch_type 00
force pc_sel 00
run 5

force rs 00001000100010001000100010001000
force rt 00001000100010001000100010001000
force pc 00000000000000001100000000000000
force target_address 00000000000000000000000111
force branch_type 01
force pc_sel 00
run 5

force rs 00001000100010001000100010001000
force rt 00001000100010001000100010001001
force pc 00000000000000001100000000000000
force target_address 00000000000000000000000111
force branch_type 01
force pc_sel 00
run 5

force rs 00001000100010001000100010001000
force rt 00001000100010001000100010001000
force pc 00000000000000001100000000000000
force target_address 00000000000000000000000111
force branch_type 10
force pc_sel 00
run 5
```

```
48
49  force rs 00001000100010001000100010001000
50  force rt 00001000100010001000100010001000
51  force pc 00000000000000001100000000000000
52  force target_address 00000000000000000000000111
53  force branch_type 10
54  force pc_sel 00
55  run 5
56
57  force rs 00001000100010001000100010001000
58  force rt 00001000100010001000100010001001
59  force pc 00000000000000001100000000000000
60  force target_address 00000000000000000000000111
61  force branch_type 10
62  force pc_sel 00
63  run 5
64
65  force rs 10001000100010001000100010001000
66  force rt 00001000100010001000100010001001
67  force pc 00000000000000001100000000000000
68  force target_address 00000000000000000000000111
69  force branch_type 11
70  force pc_sel 00
71  run 5
72
73  force rs 00001000100010001000100010001000
74  force rt 00001000100010001000100010001001
75  force pc 00000000000000001100000000000000
76  force target_address 00000000000000000000000111
77  force branch_type 11
78  force pc_sel 00
79  run 5
80
81  force rs 00001000100010001111111110001000
82  force rt 00001000100010001000100011111111
83  force pc 00000000000000001100000000000000
84  force target_address 00000000000000000000011111
85  force branch_type 00
86  force pc_sel 01
87  run 5
88
89  force rs 00001000100010001111111110001000
90  force rt 00001000100010001000100011111111
91  force pc 00000000000000001100000000000000
92  force target_address 00000000000000000000011111
93  force branch_type 00
94  force pc_sel 10
95  run 5
```

Section 3.2 – Do file for next_address_wrapper.do

```
add wave rs
add wave rt
add wave pc
add wave target_address
add wave branch_type
add wave pc_sel
add wave pc_plus_one
add wave pc_plus_one_plus_branch_offset
add wave rs_eq_rt
add wave rs_ne_rt
add wave rs_ltz
add wave pc_beq
add wave pc_bne
add wave pc_blz
add wave address_cond
add wave target_address_fullbits
add wave target_address_branch_offset
add wave next_pc

force rs 00
force rt 01
force pc 000
force target_address 011
force branch_type 00
force pc_sel 00
run 5

force rs 00
force rt 00
force pc 000
force target_address 011
force branch_type 01
force pc_sel 00
run 5

force rs 00
force rt 01
force pc 000
force target_address 011
force branch_type 01
force pc_sel 00
run 5

force rs 00
force rt 01
force pc 000
force target_address 011
force branch_type 10
force pc_sel 00
run 5

```

```
51
52  force rs 00
53  force rt 01
54  force pc 000
55  force target_address 011
56  force branch_type 10
57  force pc_sel 00
58  run 5
59
60  force rs 00
61  force rt 01
62  force pc 000
63  force target_address 011
64  force branch_type 11
65  force pc_sel 00
66  run 5
67
68  force rs 10
69  force rt 01
70  force pc 000
71  force target_address 011
72  force branch_type 11
73  force pc_sel 00
74  run 5
75
76  force rs 10
77  force rt 01
78  force pc 000
79  force target_address 011
80  force branch_type 00
81  force pc_sel 01
82  run 5
83
84  force rs 10
85  force rt 01
86  force pc 000
87  force target_address 011
88  force branch_type 00
89  force pc_sel 10
90  run 5
```

Section 3.3 – XDC file for next_address_wrapper.vhd (next_address_wrapper.xdc)

```
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [ get_ports { rs[0] } ];
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [ get_ports { rs[1] } ];
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [ get_ports { rt[0] } ];
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [ get_ports { rt[1] } ];

set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [ get_ports { pc[0] } ];
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [ get_ports { pc[1] } ];
set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [ get_ports { pc[2] } ];

set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [ get_ports { target_address[0] } ];
set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS33  } [ get_ports { target_address[1] } ];
set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS33  } [ get_ports { target_address[2] } ];

set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [ get_ports { branch_type[0] } ];
set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [ get_ports { branch_type[1] } ];
set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33  } [ get_ports { pc_sel[0] } ];
set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33  } [ get_ports { pc_sel[1] } ];

set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [ get_ports { next_pc[0] } ];
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [ get_ports { next_pc[1] } ];
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [ get_ports { next_pc[2] } ];

```

Conclusion

In conclusion we had our first experience building two versions of a next address logic unit, one with a full 32 bits input and a wrapper that can be presented on a Nexys A7 board. The results we obtained were in agreement with the theoretical output for the next address logic unit. Future labs will incorporate this logic unit into more complex systems.

;