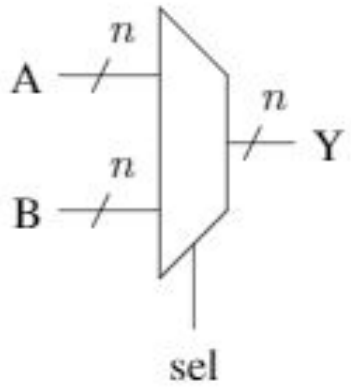
Lab 3 Report

Generic Multiplexors

CENG 342 Digital Systems



Samuel Donovan

Samantha Pfeiffer

2021-02-04

OVERVIEW

The purpose of this lab is to become familiar with variable components. This was done by using generics to design and simulate an *n*-bit 2-to-1 multiplexor and an *n*-bit 8-to-1 multiplexor.

--- N-BIT X-TO-1 MULTIPLEXOR ---

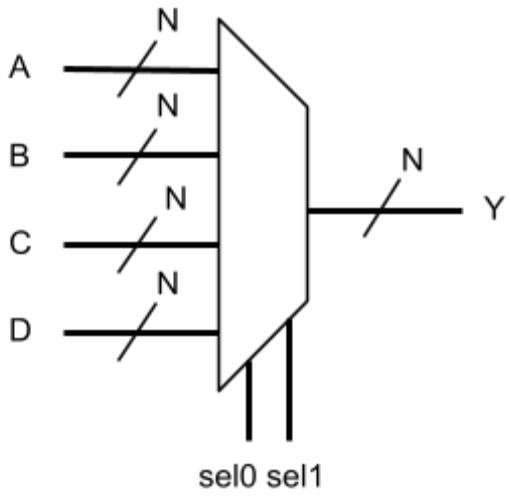
Multiplexors have a single output, **Y**, and *X* inputs, along with log_2X selector inputs. Depending on the value of the selector, a unique input is shorted to **Y**. **Table 1** shows the truth table for a 4-to-1 multiplexor. The number *N* determines the bus size of the inputs and **Y**. In a 4-to-1 mux, there are 2 selector signals, as $log_24=2$.

Table 1 4-to-1 Multiplexor

Inputs
A
B
C
D

sel_1	sel_0	Y
0	0	A
0	1	B
1	0	C
1	1	D

Figure 1 N-Bit 4-to-1 Multiplexer



DESIGN

All designs were made for a Nexys A7-100T (xc7a100tcsg324-1) FPGA board in VHDL using Vivado 2020.2.

First and foremost, for both multiplexors, *N* was set as a generic data type, meaning that it's value is passed in like a function argument when the component is instantiated. This is enabled through the use of the keyword `generic`. See the 2-to-1 mux code in **Appendix A** for an example.

--- N-BIT 2-TO-1 MULTIPLEXOR ---

The design of *N*-bit 2-to-1 multiplexor was simple. The inputs include two *N*-bit standard logic vectors, *A* and *B*, and a 1-bit standard logic value, *SEL*. The output, *Y*, is also an *N*-bit standard logic vector as well. In the code, a `when else` statement was used to set the value of *Y*.

To test the design, 4 entities with various *N* values were created. Each entity was named according to its bit length and tested in a separate process. The input values were set according to **Table 2**. Results from this simulation can be seen in **Figure 2** and the testbench code in **Appendix B**. All outputs correlated with what was expected.

Table 3 8x1 Mux Simulation Inputs

Input	1-bit	2-bit	4-bit	8-bit
A	0	0	0	00
B	1	3	f	ff

--- N-BIT 8-TO-1 MULTIPLEXOR ---

The design of an *N*-bit 8-to-1 multiplexor was more complex. The inputs include eight *N*-bit standard logic vectors, *A* to *H*, and a 3-bit standard logic vector, *SEL*. The output, *Y*, is a *N*-bit standard logic vector as well. In the architecture `when` statements were used inside a `with select` statement to set the value of *Y* with respect to the value of *SEL*. This is similar to a `switch` statement in C.

Testing for this design was similar to the process above, however only one process was used, this is further discussed in the Issues Encountered section.

To simplify our readings when simulating the on the testbench, inputs *A*-*H* were each set to the values as seen in **Table 3**. All outputs correlated with the expected tied input's value, as can be seen in **Figure 3**. Note, the input values were omitted from **Figure 3** for brevity. The testbench code exists in **Appendix D**.

Table 3 8x1 Mux Simulation Inputs

Input	1-bit	2-bit	4-bit	8-bit
A	1	1	1	01
B	0	2	2	02
C	1	3	3	03
D	0	0	4	04
E	1	1	5	05
F	0	2	6	06
G	1	3	7	07
H	0	0	0	00

--- CONSTRAINTS ---

To test the *N*-bit 2-to-1 multiplexor on our FPGA, the default value of N in the entity was set to 4. We set the constraints file up to have all 4 bits of both bus *A* and *B* hooked to switches, with the output bus *Y* connected to four LEDs. *SEL* was also hooked to a switch. The final constraint file can be seen in **Appendix E**.

SIMULATION RESULTS

Figure 2 1-bit, 2-bit, 4-bit and 8-bit 2x1 Multiplexor Testbench Simulation Result

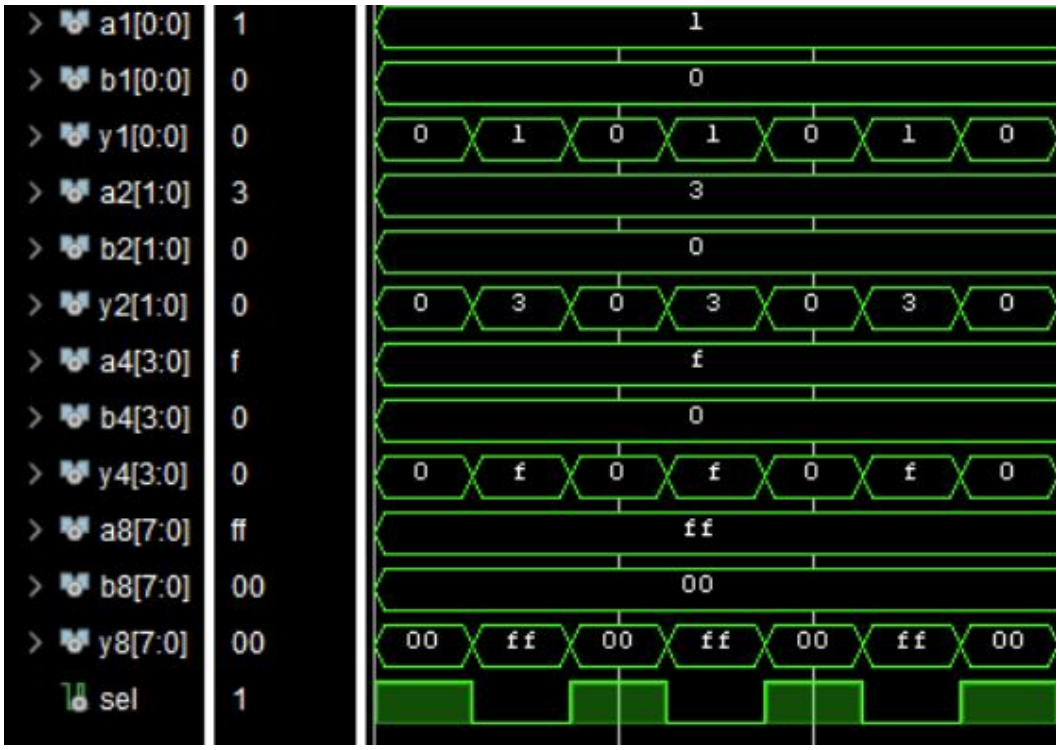


Figure 3 1-bit, 2-bit, 4-bit and 8-bit 8x1 Multiplexor Testbench Simulation Result

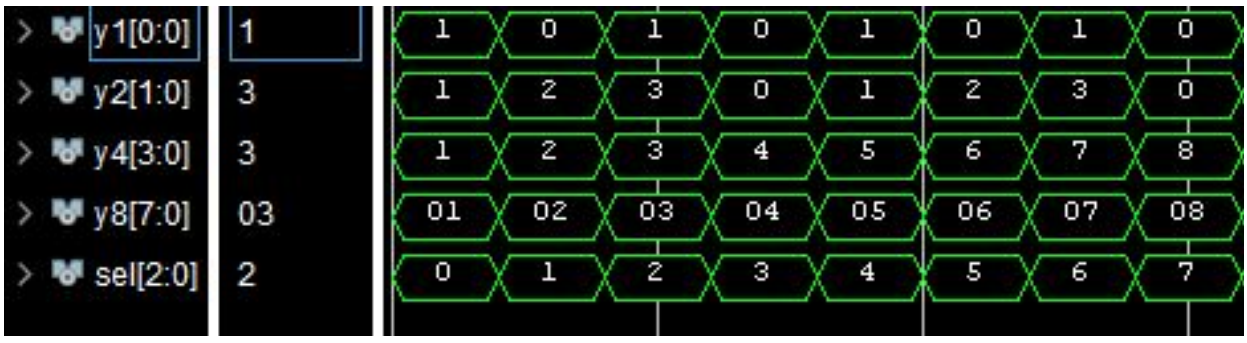
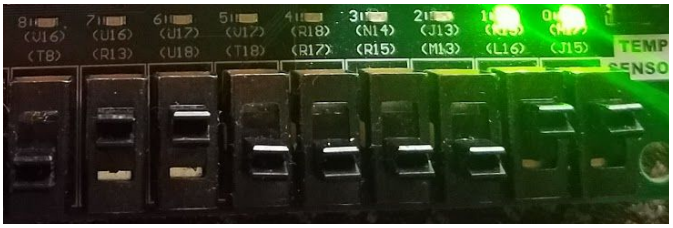


Figure 4 Picture of Programmed Board with switches SEL, B, A from left to right

SEL = ‘1’



SEL = ‘0’

ISSUES ENCOUNTERED

When coding the 2-to-1 multiplexor, creating a process for each entity became tedious as there was repeated code when changing the select bit. This issue was resolved in the 8-to-1 testbench when it was realized that a single process can be used to test each entity if the *SEL* value is shared across them.

Another issue was an inconsistency in the write-up. In part, the write-up states that a default value for *N* should be 1, while in another, it states to synthesize a **4**-bit 2-to-1 mux. In order to synthesize a 4-bit 2-to-1 multiplexor on the FPGA, the value of N needed to be changed to 4. Without this change, only one output bit was displayed.

CONCLUSIONS

We found this tool to be very useful and easy to implement when redesigning our code from Lab 2. Genericizing variables allows us to easily change our system from being 8-bit to 32-bit, without much overhead.

While it was relatively simple applying this concept to a mux's bus size, we imagine creating logic that works with a variable data size maybe difficult, if not impossible in some situations.

APPENDIX A: N-bit 2x1 Multiplexor VHDL Code (gen_mux_2x1.vhdl)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity gen_2x1_mux is
  generic(N: integer:=4);
  Port (
  A, B: in std_logic_vector(N-1 downto 0);
  SEL: in std_logic;
  Y: out std_logic_vector(N-1 downto 0)
  );
end gen_2x1_mux;

architecture Behavioral of gen_2x1_mux is

begin
    Y <= A when SEL = '0' else
         B;

end Behavioral;
```

APPENDIX B: N-bit 2x1 Multiplexor Testbench VHDL Code (gen_mux_2x1_testbench.vhdl)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity gen_2x1_mux_testbench is
end gen_2x1_mux_testbench;

architecture Behavioral of gen_2x1_mux_testbench is
signal a1, b1, y1: std_logic_vector(0 downto 0);
signal a2, b2, y2: std_logic_vector(1 downto 0);
signal a4, b4, y4: std_logic_vector(3 downto 0);
signal a8, b8, y8: std_logic_vector(7 downto 0);
signal sel : std_logic;
begin

oneBitMux: entity work.gen_2x1_mux (behavioral)
    generic map(N=>1)
    port map(A=>a1, B=>b1, Y=>y1, SEL=>sel);

 twoBitMux: entity work.gen_2x1_mux (behavioral)
    generic map(N=>2)
    port map(A=>a2, B=>b2, Y=>y2, SEL=>sel);

 fourBitMux: entity work.gen_2x1_mux (behavioral)
    generic map(N=>4)
    port map(A=>a4, B=>b4, Y=>y4, SEL=>sel);

 eightBitMux: entity work.gen_2x1_mux (behavioral)
    generic map(N=>8)
    port map(A=>a8, B=>b8, Y=>y8, SEL=>sel);

test1: process
begin

sel <= '1';
a1 <= "1";
```

```
b1 <= "0";
wait for 20ns;

sel <= '0';
wait for 20ns;

end process;

test2: process
begin

sel <= '1';
a2 <= "11";
b2 <= "00";
wait for 20ns;

sel <= '0';
wait for 20ns;

end process;

test4: process
begin

sel <= '1';
a4 <= "1111";
b4 <= "0000";
wait for 20ns;

sel <= '0';
wait for 20ns;

end process;

test8: process
begin

sel <= '1';
a8 <= "11111111";
b8 <= "00000000";
wait for 20ns;

sel <= '0';
```

```
wait for 20ns;

end process;

end Behavioral;
```

APPENDIX C: N-bit 8x1 Multiplexor VHDL Code (gen_mux_8x1.vhdl)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity gen_8x1_mux is
  generic(N: integer:=1);
  Port (
  A,B,C,D,E,F,G,H: in std_logic_vector(N-1 downto 0);
  SEL: in std_logic_vector(2 downto 0);
  Y: out std_logic_vector(N-1 downto 0)
  );
end gen_8x1_mux;

architecture Behavioral of gen_8x1_mux is

begin
with SEL select
    Y <= A when "000",
         B when "001",
         C when "010",
         D when "011",
         E when "100",
         F when "101",
         G when "110",
         H when others;

end Behavioral;
```

APPENDIX D: N-bit 8x1 Multiplexor Testbench VHDL Code (gen_mux_8x1_testbench.vhdl)

```
 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity gen_8x1_mux_testbench is
end gen_8x1_mux_testbench;

architecture Behavioral of gen_8x1_mux_testbench is
signal a1,b1,c1,d1,e1,f1,g1,h1,y1: std_logic_vector(0 downto
0);
signal a2,b2,c2,d2,e2,f2,g2,h2,y2: std_logic_vector(1 downto
0);
signal a4,b4,c4,d4,e4,f4,g4,h4,y4: std_logic_vector(3 downto
0);
signal a8,b8,c8,d8,e8,f8,g8,h8,y8: std_logic_vector(7 downto
0);
signal sel : std_logic_vector(2 downto 0);
begin

oneBitMux: entity work.gen_8x1_mux (behavioral)
    generic map(N=>1)
    port
map(A=>a1,B=>b1,C=>c1,D=>d1,E=>e1,F=>f1,G=>g1,H=>h1,Y=>y1,
SEL=>sel);

 twoBitMux: entity work.gen_8x1_mux (behavioral)
    generic map(N=>2)
    port
map(A=>a2,B=>b2,C=>c2,D=>d2,E=>e2,F=>f2,G=>g2,H=>h2,Y=>y2,
SEL=>sel);


 fourBitMux: entity work.gen_8x1_mux (behavioral)
    generic map(N=>4)
    port
map(A=>a4,B=>b4,C=>c4,D=>d4,E=>e4,F=>f4,G=>g4,H=>h4,Y=>y4,
```

```
SEL=>sel);


 eightBitMux: entity work.gen_8x1_mux (behavioral)
    generic map(N=>8)
    port
map(A=>a8,B=>b8,C=>c8,D=>d8,E=>e8,F=>f8,G=>g8,H=>h8,Y=>y8,
SEL=>sel);

test: process
variable i: integer;
begin
a1 <= "1";
b1 <= "0";
c1 <= "1";
d1 <= "0";
e1 <= "1";
f1 <= "0";
g1 <= "1";
h1 <= "0";

a2 <= "01";
b2 <= "10";
c2 <= "11";
d2 <= "00";
e2 <= "01";
f2 <= "10";
g2 <= "11";
h2 <= "00";

a4 <= "0001";
b4 <= "0010";
c4 <= "0011";
d4 <= "0100";
e4 <= "0101";
f4 <= "0110";
g4 <= "0111";
h4 <= "1000";

a8 <= "00000001";
b8 <= "00000010";
c8 <= "00000011";
d8 <= "00000100";
```

```
e8 <= "00000101";
f8 <= "00000110";
g8 <= "00000111";
h8 <= "00001000";

sel <= "000";
for i in 0 to 8 loop
    sel <= std_logic_vector(TO_UNSIGNED(i,3));
    wait for 20ns;
end loop;

end process;

end Behavioral;
```

APPENDIX E: Constraints (constraints.xdc)

```
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 }
[get_ports { A[0] }];
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 }
[get_ports { A[1] }];
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 }
[get_ports { A[2] }];
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 }
[get_ports { A[3] }];
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 }
[get_ports { B[0] }];
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 }
[get_ports { B[1] }];
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 }
[get_ports { B[2] }];
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 }
[get_ports { B[3] }];
set_property -dict { PACKAGE_PIN T8   IOSTANDARD LVCMOS33 }
[get_ports { SEL }];


set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 }
[get_ports { Y[0] }];
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 }
[get_ports { Y[1] }];
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 }
[get_ports { Y[2] }];
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 }
[get_ports { Y[3] }];
```