CS2022 PROJECT 1- DATAPATH DESIGN

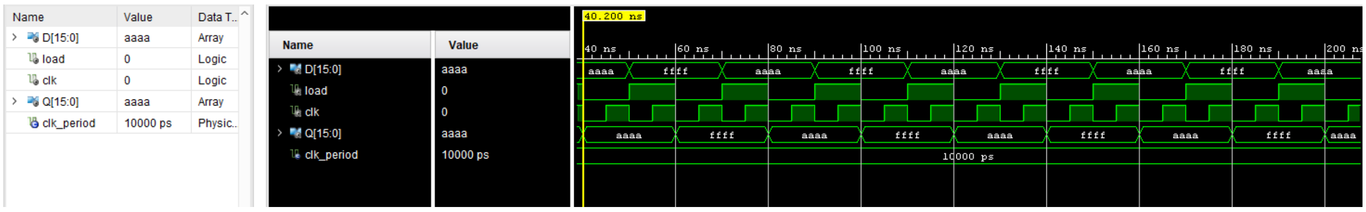
PART A

Aniket Agarwal 17317437

RESULTS OF TEST BENCHES

Register

The register is given a new value after every 5ns. It works as expected for each edge on a given rising clock signal where the load is high.

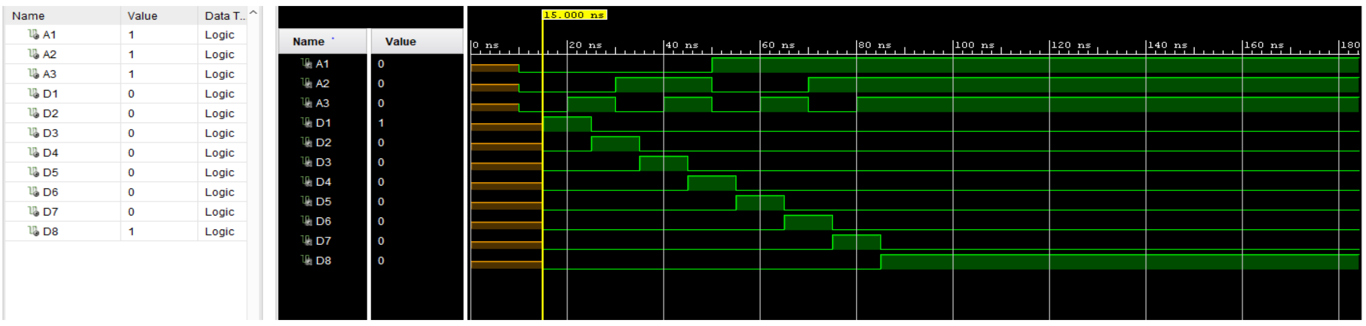


Name	Value	Data T..
D[15:0]	aaaa	Array
load	0	Logic
clk	0	Logic
Q[15:0]	aaaa	Array
clk_period	10000 ps	Physic..

Name	Value
D[15:0]	aaaa
load	0
clk	0
Q[15:0]	aaaa
clk_period	10000 ps

Decoder 3_to_8 16 bit

The test bench cycles through each given combination, causing each given output pin to turn high via the register selection.

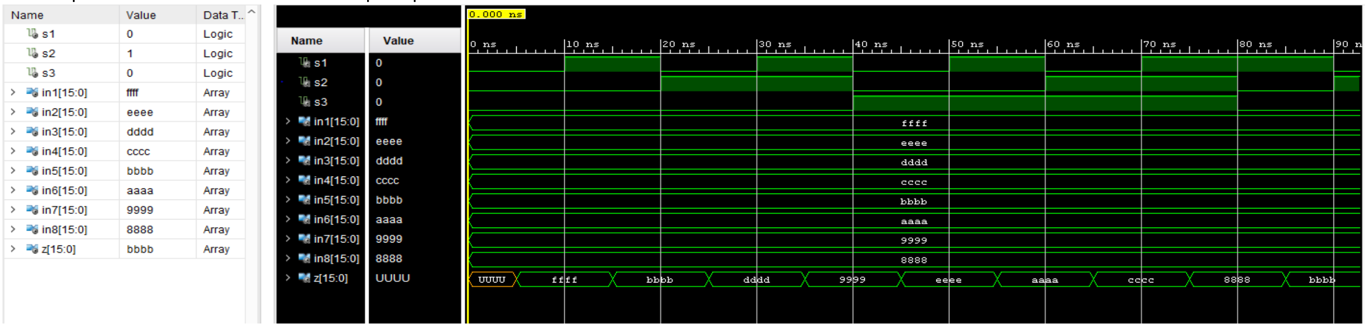


Name	Value	Data T..
A1	1	Logic
A2	1	Logic
A3	1	Logic
D1	0	Logic
D2	0	Logic
D3	0	Logic
D4	0	Logic
D5	0	Logic
D6	0	Logic
D7	0	Logic
D8	1	Logic

Name	Value
A1	0
A2	0
A3	0
D1	1
D2	0
D3	0
D4	0
D5	0
D6	0
D7	0
D8	0

Multiplexer 8_to_1 16 bit

This multiplexer cycles through all given combinations of s1, s2 and s3, thus allowing z to cycle through the inputs in1 to in8 as the output pin.

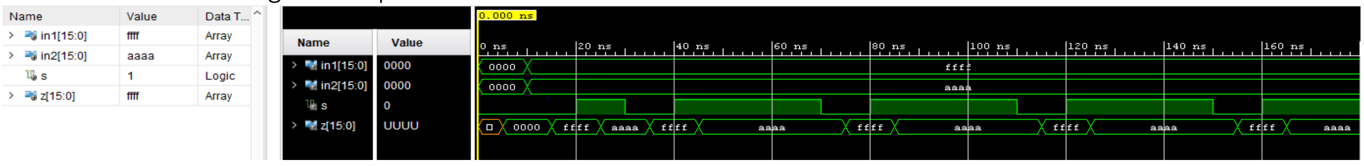


Name	Value	Data T..
s1	0	Logic
s2	1	Logic
s3	0	Logic
in1[15:0]	ffff	Array
in2[15:0]	eeee	Array
in3[15:0]	dddd	Array
in4[15:0]	cccc	Array
in5[15:0]	bbbb	Array
in6[15:0]	aaaa	Array
in7[15:0]	9999	Array
in8[15:0]	8888	Array
z[15:0]	bbbb	Array

Name	Value
s1	0
s2	0
s3	0
in1[15:0]	ffff
in2[15:0]	eeee
in3[15:0]	dddd
in4[15:0]	cccc
in5[15:0]	bbbb
in6[15:0]	aaaa
in7[15:0]	9999
in8[15:0]	8888
z[15:0]	UUUU

Multiplexer 2_to_1 16 bit

The multiplexer cycles through a series of changes via s between 0 and 1. The output of the multiplexer switches between the given outputs for in1 and in2.

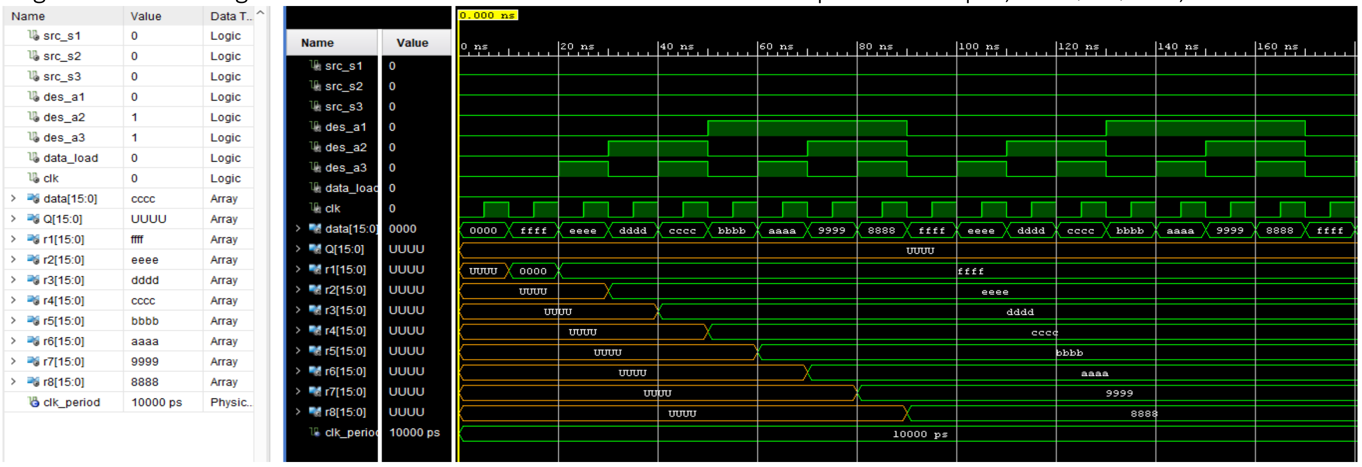


Name	Value	Data T...
in1[15:0]	ffff	Array
in2[15:0]	aaaa	Array
s	1	Logic
z[15:0]	ffff	Array

Name	Value
in1[15:0]	0000
in2[15:0]	0000
s	0
z[15:0]	UUUU

Final VHDL Code

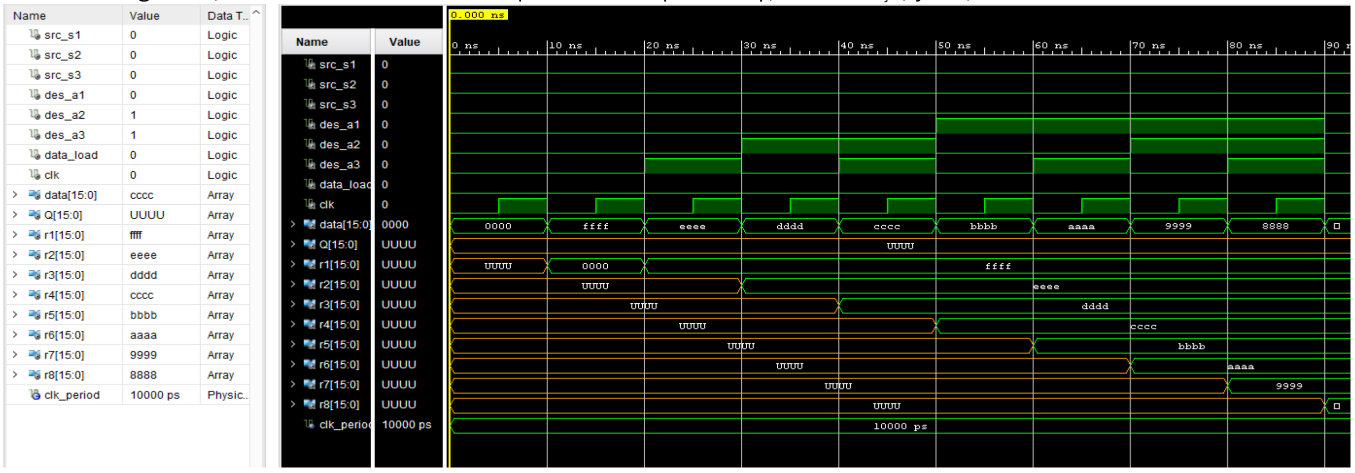
Register simulation given different hexadecimal value for each subsequent data input, i.e. R_i<-X_i i = 1; 8



Name	Value	Data T...
src_s1	0	Logic
src_s2	0	Logic
src_s3	0	Logic
des_a1	0	Logic
des_a2	1	Logic
des_a3	1	Logic
data_load	0	Logic
clk	0	Logic
data[15:0]	cccc	Array
Q[15:0]	UUUU	Array
r1[15:0]	ffff	Array
r2[15:0]	eeee	Array
r3[15:0]	dddd	Array
r4[15:0]	cccc	Array
r5[15:0]	bbbb	Array
r6[15:0]	aaaa	Array
r7[15:0]	9999	Array
r8[15:0]	8888	Array
clk_period	10000 ps	Physic..

Name	Value
src_s1	0
src_s2	0
src_s3	0
des_a1	0
des_a2	0
des_a3	0
data_load	0
clk	0
data[15:0]	0000
Q[15:0]	UUUU
r1[15:0]	UUUU
r2[15:0]	UUUU
r3[15:0]	UUUU
r4[15:0]	UUUU
r5[15:0]	UUUU
r6[15:0]	UUUU
r7[15:0]	UUUU
r8[15:0]	UUUU
clk_period	10000 ps

The register r1 is loaded with values from the input data from the test bench. The value is then transferred from one given register to the incremented register, r2; i.e. from r1 to r2, r2 to r3 ... r7 to r8. As each load is cycled through the data inputs, the transfer of data cycling between each register is passed on until the input is reset to 0x0000 after the first input time, which shows that the values stem from the registers, and not from the data input itself sequentially, i.e. R_i<-R_j i; j = 0; 7



Name	Value	Data T...
src_s1	0	Logic
src_s2	0	Logic
src_s3	0	Logic
des_a1	0	Logic
des_a2	1	Logic
des_a3	1	Logic
data_load	0	Logic
clk	0	Logic
data[15:0]	cccc	Array
Q[15:0]	UUUU	Array
r1[15:0]	ffff	Array
r2[15:0]	eeee	Array
r3[15:0]	dddd	Array
r4[15:0]	cccc	Array
r5[15:0]	bbbb	Array
r6[15:0]	aaaa	Array
r7[15:0]	9999	Array
r8[15:0]	8888	Array
clk_period	10000 ps	Physic..

Name	Value
src_s1	0
src_s2	0
src_s3	0
des_a1	0
des_a2	0
des_a3	0
data_load	0
clk	0
data[15:0]	0000
Q[15:0]	UUUU
r1[15:0]	UUUU
r2[15:0]	UUUU
r3[15:0]	UUUU
r4[15:0]	UUUU
r5[15:0]	UUUU
r6[15:0]	UUUU
r7[15:0]	UUUU
r8[15:0]	UUUU
clk_period	10000 ps

VHDL Components Source Codes

Register

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity reg is
        Port(
                        D : in STD_LOGIC_VECTOR(15 downto 0);
                        load, clk : in STD_LOGIC;
                        Q : out STD_LOGIC_VECTOR(15 downto 0)
                );
end reg;
architecture Behavioral of reg is
begin   process(clk)
        begin
                if(rising_edge(clk)) then
                        if(load = '1') then
                                Q <= D after 5ns;
                        end if;
                end if;
        end process;
end Behavioral;
```

Decoder 3_to_8 16 bit

```
library IEEE;
use ieee.std_logic_1164.all;
entity decoder is
   port(A1, A2, A3: in std_logic;
        D1, D2, D3, D4, D5, D6, D7, D8: out std_logic);
end decoder;
architecture Behavioral of decoder is
begin
D1<=((not A1) and (not A2) and (not A3)) after 5ns ;
D2<=((not A1) and (not A2) and (A3)) after 5ns ;
D3<=((not A1) and (A2) and (not A3)) after 5ns ;
D4<=((not A1) and (A2) and (A3)) after 5ns ;
D5<=((A1) and (not A2) and (not A3)) after 5ns ;
D6<=((A1) and (not A2) and (A3)) after 5ns ;
D7<=((A1) and (A2) and (not A3)) after 5ns ;
D8<=((A1) and (A2) and (A3)) after 5ns ;
end Behavioral;
```

Multiplexer 8_to_1 16 bit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity multiplexer is
   Port ( s1, s2, s3 : in  STD_LOGIC;
          in1, in2, in3, in4, in5, in6, in7, in8 : in  STD_LOGIC_VECTOR (15 downto 0);
          z : out  STD_LOGIC_VECTOR (15 downto 0));
end multiplexer;
architecture Behavioral of multiplexer is
begin
  z <= in1 after 5ns when s1 = '0' and s2 = '0' and s3 = '0' else
       in2 after 5ns when s1 = '0' and s2 = '0' and s3 = '1' else
       in3 after 5ns when s1 = '0' and s2 = '1' and s3 = '0' else
       in4 after 5ns when s1 = '0' and s2 = '1' and s3 = '1' else
       in5 after 5ns when s1 = '1' and s2 = '0' and s3 = '0' else
       in6 after 5ns when s1 = '1' and s2 = '0' and s3 = '1' else
       in7 after 5ns when s1 = '1' and s2 = '1' and s3 = '0' else
       in8 after 5ns when s1 = '1' and s2 = '1' and s3 = '1' else
       x"0000" after 5ns;
end Behavioral;
```

Multiplexer 2_to_1 16 bit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux is
   Port(
                    s : in STD_LOGIC;
                    in1, in2 : in STD_LOGIC_VECTOR(15 downto 0);
                    z : out STD_LOGIC_VECTOR(15 downto 0)
        );
end mux;
architecture Behavioral of mux is
begin
        z <=    in1 after 5ns when s = '0' else
                in2 after 5ns when s = '1' else
                x"0000" after 5ns;
end Behavioral;
```

Final VHDL code

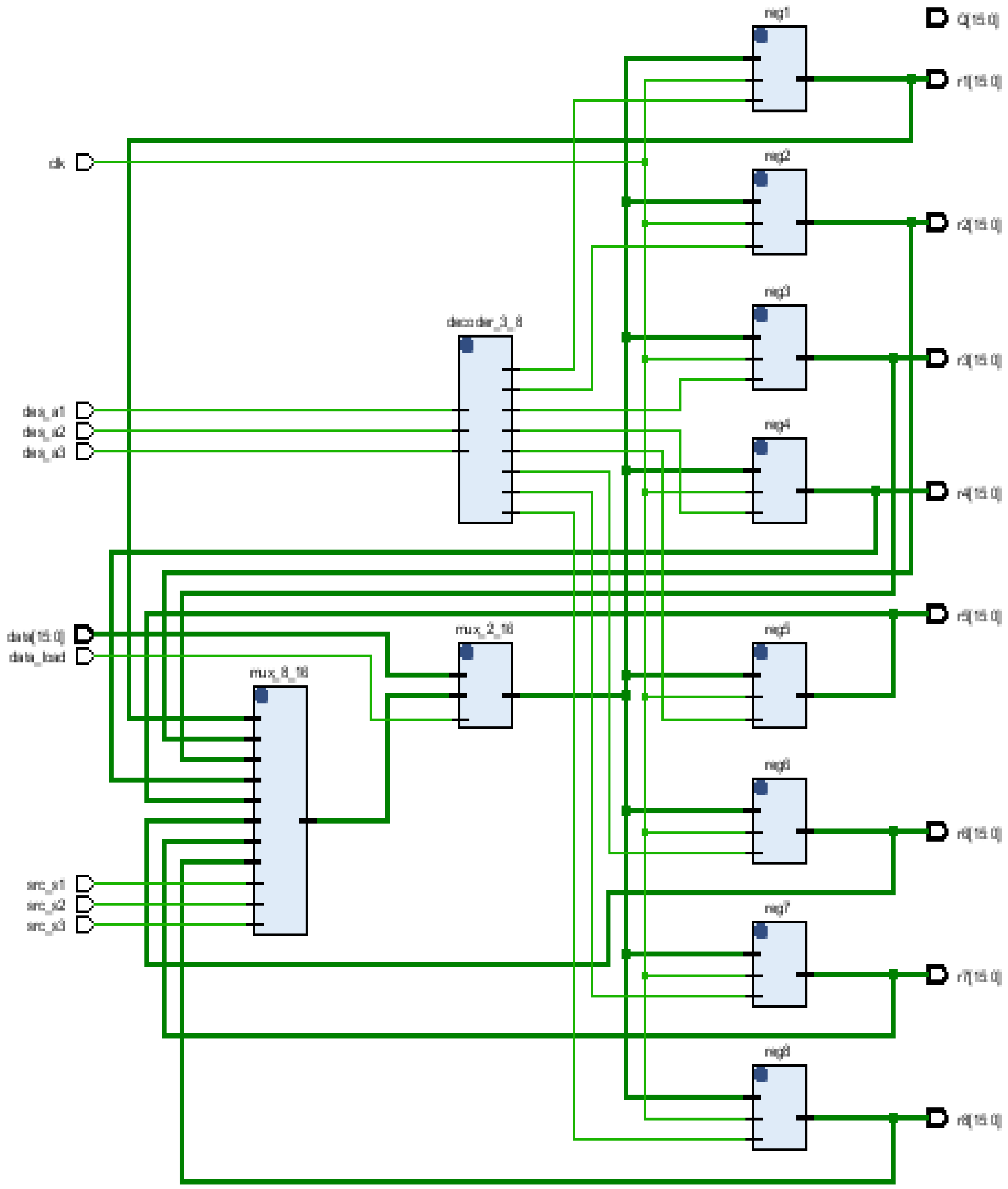
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity register_final is
        Port(
    src_s1, src_s2, src_s3     : in STD_LOGIC;
    des_a1, des_a2, des_a3     : in STD_LOGIC;
    data_load, clk : in STD_LOGIC;
    data          : in STD_LOGIC_VECTOR(15 downto 0);
    Q            : out STD_LOGIC_VECTOR(15 downto 0);
    r1, r2, r3, r4, r5, r6, r7, r8 : out STD_LOGIC_VECTOR(15 downto 0)
    );
end register_final;
architecture Behavioral of register_final is
        Component reg
                Port(
                                load, clk         : in     STD_LOGIC;
                                D                 : in     STD_LOGIC_VECTOR(15 downto 0);
                                Q                 : out STD_LOGIC_VECTOR(15 downto 0)
                        );
        End Component;
Component decoder
                Port(
                                a1, a2, a3 : in STD_LOGIC;
                                D1, D2, D3, D4, D5, D6, D7, D8 : out STD_LOGIC
                        );
        End Component;
        Component mux
                Port(
                                in1, in2 : in STD_LOGIC_VECTOR(15 downto 0);
                                s : in STD_LOGIC;
```

```
                                    z : out STD_LOGIC_VECTOR(15 downto 0)
                              );
            End Component;
            Component multiplexer
                  Port(
                  in1, in2, in3, in4, in5, in6, in7, in8 : in STD_LOGIC_VECTOR(15 downto 0);
                  s1, s2, s3 : STD_LOGIC;
                  z : out STD_LOGIC_VECTOR(15 downto 0)
                   );
            End Component;
signal load_r1, load_r2, load_r3, load_r4, load_r5, load_r6, load_r7, load_r8  : STD_LOGIC;
signal q_r1, q_r2, q_r3, q_r4, q_r5, q_r6, q_r7, q_r8 : STD_LOGIC_VECTOR(15 downto 0);
signal data_src_mux_out, src_r : STD_LOGIC_VECTOR(15 downto 0);
begin
reg1 : reg PORT MAP(
                                    load    =>              load_r1,
                                    clk     =>              clk,
                                      D  =>         data_src_mux_out,
                                     Q      =>              q_r1);
reg2 : reg PORT MAP(
                  load    =>   load_r2,
                  clk    =>   clk,
                  D      =>   data_src_mux_out,
                  Q      =>   q_r2
                );
reg3 : reg PORT MAP(
                  load     =>   load_r3,
                  clk     =>   clk,
                  D       =>   data_src_mux_out,
                  Q       =>   q_r3
                 );
```

```
   reg4 : reg PORT MAP(
                         load    =>   load_r4,
                         clk     =>   clk,
                         D       =>   data_src_mux_out,
                         Q       =>   q_r4
                    );
 reg5 : reg PORT MAP(
                         load    =>   load_r5,
                         clk     =>   clk,
                         D       =>   data_src_mux_out,
                         Q       =>   q_r5
                    );
 reg6 : reg PORT MAP(
                         load    =>   load_r6,
                         clk     =>   clk,
                         D       =>   data_src_mux_out,
                         Q       =>   q_r6
                    );
 reg7 : reg PORT MAP(
                         load     =>  load_r7,
                         clk      =>  clk,
                         D        =>  data_src_mux_out,
                         Q        =>  q_r7     );
 reg8 : reg PORT MAP(
                         load    =>   load_r8,
                         clk     =>   clk,
                         D       =>   data_src_mux_out,
                         Q       =>   q_r8
                    );
 decoder_3_8 : decoder PORT MAP(
                                             a1         =>         des_a1,
                                            a2          =>        des_a2,
                                            a3          =>        des_a3,
                                            D1          =>        load_r1,
                                             D2         =>        load_r2,
                                             D3         =>        load_r3,
                                             D4         =>        load_r4,
                                             D5         =>        load_r5,
                                             D6         =>        load_r6,
                                            D7          =>        load_r7,
                                             D8        =>          load_r8
                                        );
 mux_2_16 : mux PORT MAP(
                                             in1        =>        data,
                                             in2        =>        src_r,
                                              s         =>        data_load,
                                              z         =>        data_src_mux_out
                                      );
 mux_8_16 : multiplexer PORT MAP(
                              in1        =>        q_r1,
                             in2         =>        q_r2,
                              in3        =>        q_r3,
                             in4         =>        q_r4,
                             in5         =>        q_r5,
                              in6        =>        q_r6,
                              in7        =>        q_r7,
                              in8        =>        q_r8,
                             s1          =>        src_s1,
                             s2          =>        src_s2,
                             s3          =>        src_s3,
                              z          =>        src_r
                      );
```

```vhdl
        r1 <= q_r1;

        r2 <= q_r2;

        r3 <= q_r3;

        r4 <= q_r4;

        r5 <= q_r5;

        r6 <= q_r6;

        r7 <= q_r7;

        r8 <= q_r8;

end Behavioral;
```

Schematics for the circuit



Component Test Benches

Register

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity register_final is
        Port(
    src_s1, src_s2, src_s3     : in STD_LOGIC;
    des_a1, des_a2, des_a3     : in STD_LOGIC;
    data_load, clk : in STD_LOGIC;
    data          : in STD_LOGIC_VECTOR(15 downto 0);
    Q            : out STD_LOGIC_VECTOR(15 downto 0);
    r1, r2, r3, r4, r5, r6, r7, r8 : out STD_LOGIC_VECTOR(15 downto 0)
    );
end register_final;
architecture Behavioral of register_final is
        Component reg
                Port(
                                load, clk            : in        STD_LOGIC;
                                D                              : in STD_LOGIC_VECTOR(15 downto 0);
                                Q                              : out STD_LOGIC_VECTOR(15 downto 0)
                        );
        End Component;
        Component decoder
                Port(
                                a1, a2, a3 : in STD_LOGIC;
                                D1, D2, D3, D4, D5, D6, D7, D8 : out STD_LOGIC
                        );
        End Component;
        Component mux
                Port(
                                in1, in2 : in STD_LOGIC_VECTOR(15 downto 0);
```

```
                            s : in STD_LOGIC;
                            z : out STD_LOGIC_VECTOR(15 downto 0)
                       );
        End Component;
        Component multiplexer
                Port(
                in1, in2, in3, in4, in5, in6, in7, in8 : in STD_LOGIC_VECTOR(15 downto 0);
                s1, s2, s3 : STD_LOGIC;
                z : out STD_LOGIC_VECTOR(15 downto 0)
                       );
        End Component;
signal load_r1, load_r2, load_r3, load_r4, load_r5, load_r6, load_r7, load_r8  : STD_LOGIC;
signal q_r1, q_r2, q_r3, q_r4, q_r5, q_r6, q_r7, q_r8 : STD_LOGIC_VECTOR(15 downto 0);
signal data_src_mux_out, src_r : STD_LOGIC_VECTOR(15 downto 0);
begin
        reg1 : reg PORT MAP(
                            load    =>      load_r1,
                            clk     =>      clk,
                              D     =>      data_src_mux_out,
                            Q       =>      q_r1
                     );
        reg2 : reg PORT MAP(
                load    =>    load_r2,
                clk    =>    clk,
                D      =>    data_src_mux_out,
                Q      =>    q_r2                  );
        reg3 : reg PORT MAP(
                load     =>   load_r3,
                clk      =>   clk,
                D        =>   data_src_mux_out,
                Q        =>   q_r3   );
```

```
reg4 : reg PORT MAP(
                load    =>   load_r4,
                clk     =>   clk,
                D       =>   data_src_mux_out,
                Q       =>   q_r4  );
      reg5 : reg PORT MAP(
                load    =>   load_r5,
                clk     =>   clk,
                D       =>   data_src_mux_out,
                Q       =>   q_r5  );
reg6 : reg PORT MAP(
                load    =>   load_r6,
                clk     =>   clk,
                D       =>   data_src_mux_out,
                Q       =>   q_r6  );
reg7 : reg PORT MAP(
                load     =>  load_r7,
                clk      =>  clk,
                D        =>  data_src_mux_out,
                Q        =>  q_r7  );
reg8 : reg PORT MAP(
                load    =>   load_r8,
                clk     =>   clk,
                D       =>   data_src_mux_out,
                Q       =>   q_r8 );
      decoder_3_8 : decoder PORT MAP(
                                a1       =>       des_a1,
                                a2       =>       des_a2,
                                a3       =>       des_a3,
                                D1       =>       load_r1,
                                D2       =>       load_r2,
                                D3        =>        load_r3,
                                D4        =>        load_r4,
                                D5        =>        load_r5,
                                D6        =>        load_r6,
                                D7        =>        load_r7,
                                D8        =>        load_r8 );
      mux_2_16 : mux PORT MAP(
                                in1       =>        data,
                                in2       =>        src_r,
                                s         =>        data_load,
                                z         =>        data_src_mux_out);
      mux_8_16 : multiplexer PORT MAP(
                                 in1       =>        q_r1,
                                 in2       =>        q_r2,
                                 in3       =>        q_r3,
                                 in4       =>        q_r4,
                                 in5       =>        q_r5,
                                 in6       =>        q_r6,
                                 in7       =>        q_r7,
                                 in8       =>        q_r8,
                                 s1        =>        src_s1,
                                 s2        =>        src_s2,
                                 s3        =>        src_s3,
                                 z         =>        src_r    );
r1 <= q_r1;r2 <= q_r2;    r3 <= q_r3;r4 <= q_r4;    r5 <= q_r5;r6 <= q_r6;    r7 <= q_r7;r8 <= q_r8;
end Behavioral;
```

Decoder 3_to_8 16 bit

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;
entity decoder_tb is
end;
architecture behavior of decoder_tb is
component decoder
port(A1, A2, A3: in std_logic;
          D1, D2, D3, D4, D5, D6, D7, D8: out std_logic);
  end component;
signal A1, A2, A3: std_logic;
signal D1, D2, D3, D4, D5, D6, D7, D8: std_logic;
begin
uut: decoder port map ( A1 => A1,A2 => A2, A3 => A3,
                        D1 => D1,D2 => D2,D3 => D3,D4 => D4,D5 => D5,D6 => D6,D7 => D7,D8 => D8
                       );
stimulus: process
begin
         wait for 10ns; A1 <= '0';A2 <= '0';A3 <= '0';
         wait for 10ns; A1 <= '0';A2 <= '0';A3 <= '1';
         wait for 10ns; A1 <= '0';A2 <= '1';A3 <= '0';
         wait for 10ns; A1 <= '0';A2 <= '1';A3 <= '1';
         wait for 10ns; A1 <= '1';A2 <= '0';A3 <= '0';
         wait for 10ns; A1 <= '1';A2 <= '0';A3 <= '1';
         wait for 10ns; A1 <= '1';A2 <= '1';A3 <= '0';
         wait for 10ns; A1 <= '1';A2 <= '1';A3 <= '1';
wait;
end process;
end;
```

Multiplexer 8_to_1 16 bit

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY multiplexer_tb IS
END multiplexer_tb;
ARCHITECTURE behavior OF multiplexer_tb IS
COMPONENT multiplexer
PORT(
        s1: IN  std_logic;
        s2: IN  std_logic;
        s3: IN  std_logic;
        in1 : IN  std_logic_vector(15 downto 0);
        in2 : IN  std_logic_vector(15 downto 0);
        in3 : IN  std_logic_vector(15 downto 0);
        in4 : IN  std_logic_vector(15 downto 0);
        in5 : IN  std_logic_vector(15 downto 0);
        in6 : IN  std_logic_vector(15 downto 0);
        in7 : IN  std_logic_vector(15 downto 0);
        in8 : IN  std_logic_vector(15 downto 0);
        z : OUT  std_logic_vector(15 downto 0)
       );
END COMPONENT;
signal s1 : std_logic := '0';
signal s2 : std_logic := '0';
signal s3 : std_logic := '0';
signal in1 : std_logic_vector(15 downto 0) := (others => '0');
signal in2 : std_logic_vector(15 downto 0) := (others => '0');
signal in3 : std_logic_vector(15 downto 0) := (others => '0');
signal in4 : std_logic_vector(15 downto 0) := (others => '0');
signal in5 : std_logic_vector(15 downto 0) := (others => '0');
signal in6 : std_logic_vector(15 downto 0) := (others => '0');
signal in7 : std_logic_vector(15 downto 0) := (others => '0');

signal in8 : std_logic_vector(15 downto 0) := (others => '0');

signal z : std_logic_vector(15 downto 0);

BEGIN

  uut: multiplexer PORT MAP (

        s1 => s1,s2 => s2,s3 => s3,

        in1 => in1,in2 => in2,in3 => in3,in4 => in4,in5 => in5,in6 => in6,in7 => in7,in8 => in8,

        z => z

      );

stim_proc: process

begin

                  in1 <= x"FFFF";

                  in2 <= x"EEEE";

                  in3 <= x"DDDD";

                  in4 <= x"CCCC";

                  in5 <= x"BBBB";

                  in6 <= x"AAAA";

                  in7 <= x"9999";

                  in8 <= x"8888";

                  wait for 10ns;    s1 <= '1';s2 <= '0';s3 <= '0';

                  wait for 10ns;   s1 <= '0';s2 <= '1';s3 <= '0';

                  wait for 10ns;    s1 <= '1';s2 <= '1';s3 <= '0';

                  wait for 10ns;    s1 <= '0';s2 <= '0';s3 <= '1';

                  wait for 10ns;    s1 <= '1';s2 <= '0';s3 <= '1';

                  wait for 10ns;    s1 <= '0';s2 <= '1';s3 <= '1';

                  wait for 10ns;    s1 <= '1';s2 <= '1';s3 <= '1';

  end process;

END;
```

Multiplexer 2_to_1 16 bit

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

ENTITY mux_tb IS

END mux_tb;

ARCHITECTURE behavior OF mux_tb IS

COMPONENT mux

PORT(

      in1 : IN  std_logic_vector(15 downto 0);

      in2 : IN  std_logic_vector(15 downto 0);

      s : IN  std_logic;

      z : OUT  std_logic_vector(15 downto 0)

     );

  END COMPONENT;

  signal in1 : std_logic_vector(15 downto 0) := (others => '0');

  signal in2 : std_logic_vector(15 downto 0) := (others => '0');

  signal s : std_logic := '0';

  signal z : std_logic_vector(15 downto 0);

BEGIN

  mux_2_16: mux PORT MAP (

       in1 => in1,       in2 => in2,

       s => s,    z => z      );

  stim_proc: process

  begin

                  wait for 10ns; in1 <= x"FFFF"; in2 <= x"AAAA";

                 wait for 10ns;s <= '1';

                 wait for 10ns;s <= '0';

                  wait for 10ns;s <= '1';

            end process;

END;
```

Final VHDL code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY datapath_tb IS
END datapath_tb;
ARCHITECTURE behavior OF datapath_tb IS
COMPONENT register_final
   PORT(
        src_s1 : IN  std_logic;  src_s2 : IN  std_logic; src_s3 : IN  std_logic;
        des_a1 : IN  std_logic; des_a2 : IN  std_logic;des_a3 : IN  std_logic;
        data_load : IN  std_logic;  clk : IN  std_logic;
        data : IN  std_logic_vector(15 downto 0);
        Q : OUT  std_logic_vector(15 downto 0);
        r1 : OUT  std_logic_vector(15 downto 0);
        r2 : OUT  std_logic_vector(15 downto 0);
        r3 : OUT  std_logic_vector(15 downto 0);
        r4 : OUT  std_logic_vector(15 downto 0);
        r5 : OUT  std_logic_vector(15 downto 0);
        r6 : OUT  std_logic_vector(15 downto 0);
        r7 : OUT  std_logic_vector(15 downto 0);
        r8 : OUT  std_logic_vector(15 downto 0)   );
   END COMPONENT;
  signal src_s1 : std_logic := '0';
  signal src_s2 : std_logic := '0';
  signal src_s3 : std_logic := '0';
  signal des_a1 : std_logic := '0';
  signal des_a2 : std_logic := '0';
  signal des_a3 : std_logic := '0';
  signal data_load : std_logic := '0';
  signal clk : std_logic := '0';
  signal data : std_logic_vector(15 downto 0) := (others => '0');
  signal Q : std_logic_vector(15 downto 0);
  signal r1 : std_logic_vector(15 downto 0);
  signal r2 : std_logic_vector(15 downto 0);
  signal r3 : std_logic_vector(15 downto 0);
  signal r4 : std_logic_vector(15 downto 0);
  signal r5 : std_logic_vector(15 downto 0);
  signal r6 : std_logic_vector(15 downto 0);
  signal r7 : std_logic_vector(15 downto 0);
  signal r8 : std_logic_vector(15 downto 0);
BEGIN
  uut: register_final PORT MAP (
       src_s1 => src_s1, src_s2 => src_s2,  src_s3 => src_s3,
       des_a1 => des_a1, des_a2 => des_a2, des_a3 => des_a3,
       data_load => data_load, clk => clk,   data => data,
       Q => Q, r1 => r1, r2 => r2,  r3 => r3,  r4 => r4, r5 => r5, r6 => r6, r7 => r7, r8 => r8);
  clk_process :process
  begin
                clk <= '0';wait for clk_period/2; clk <= '1';wait for clk_period/2;
  end process;
  stim_proc: process
  begin
                wait for 10ns;    des_a1 <= '0';    des_a2 <= '0';    des_a3 <= '0'; data <= x"FFFF";
                wait for 10ns;    des_a1 <= '0';    des_a2 <= '0';    des_a3 <= '1';    data <= x"EEEE";
                wait for 10ns;    des_a1 <= '0';    des_a2 <= '1';    des_a3 <= '0';    data <= x"DDDD";
                wait for 10ns;    des_a1 <= '0';    des_a2 <= '1';    des_a3 <= '1';    data <= x"CCCC";
                wait for 10ns;    des_a1 <= '1';    des_a2 <= '0';    des_a3 <= '0';    data <= x"BBBB";
                wait for 10ns;    des_a1 <= '1';    des_a2 <= '0';    des_a3 <= '1';    data <= x"AAAA";
                wait for 10ns;    des_a1 <= '1';    des_a2 <= '1';    des_a3 <= '0';    data <= x"9999";
                wait for 10ns;    des_a1 <= '1';des_a2 <= '1';des_a3 <= '1';data <= x"8888";
  end process;
END;
```