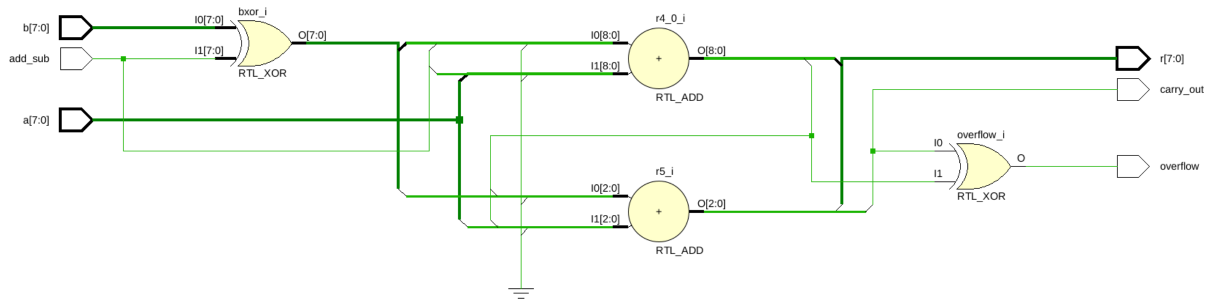
Lab 3: Generic Adder/Subtractor and Generic Barrel Shifter

-Overview-

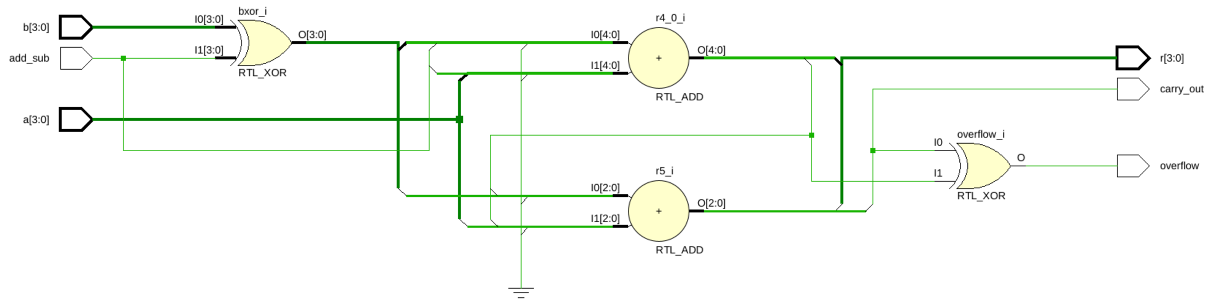
In this lab, we're learning to use generics and the generate statement in VHDL to build flexible circuits. We'll first modify a 6-bit two's complement adder/subtractor to make it generic and test both 4-bit and 8-bit versions through simulation. Then, we'll design a generic multi-function barrel shifter that can perform different types of shifts, adjusting for various bit sizes. After verifying everything in simulation, we'll synthesize the shifter and test it on a Nexys A7 FPGA board.

-Design-

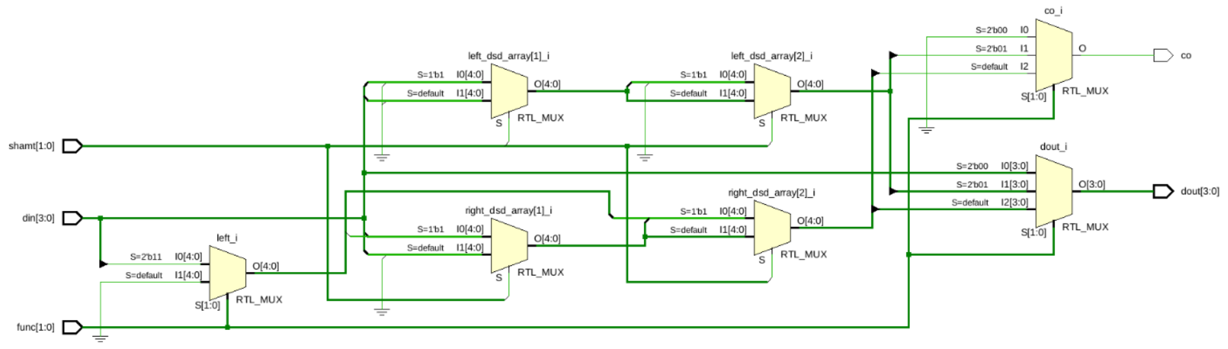
8 bit adder/subtractor design



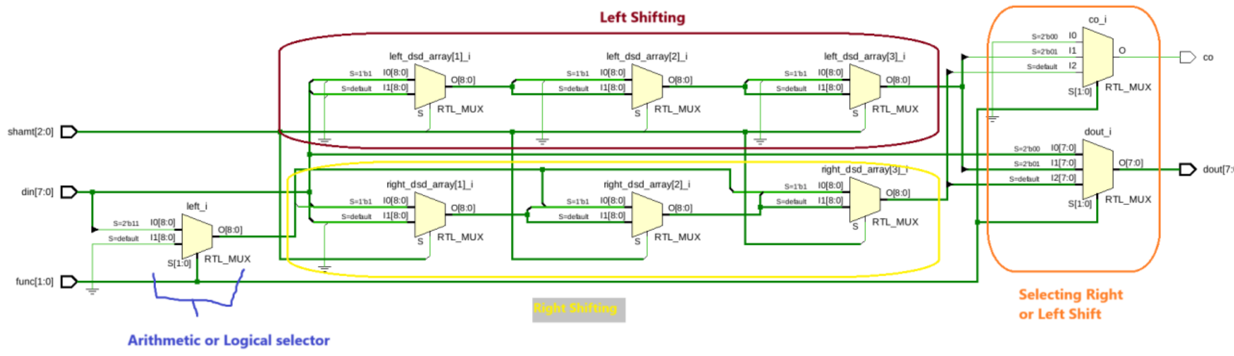
4 bit adder/subtractor design



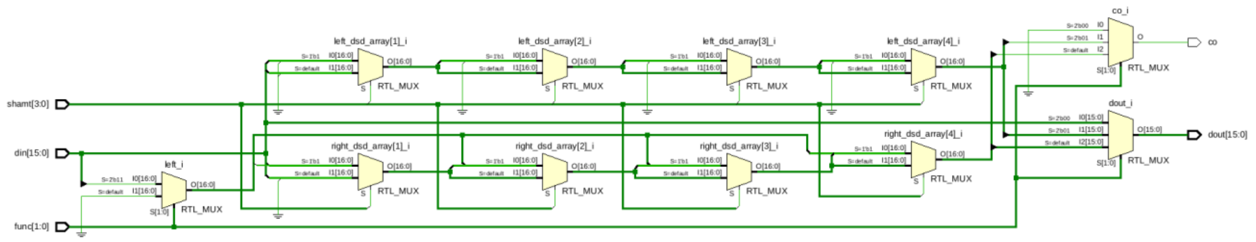
4 bit input (2 shift bits) barrel shifter design



8 bit input (3 shift bits) barrel shifter design

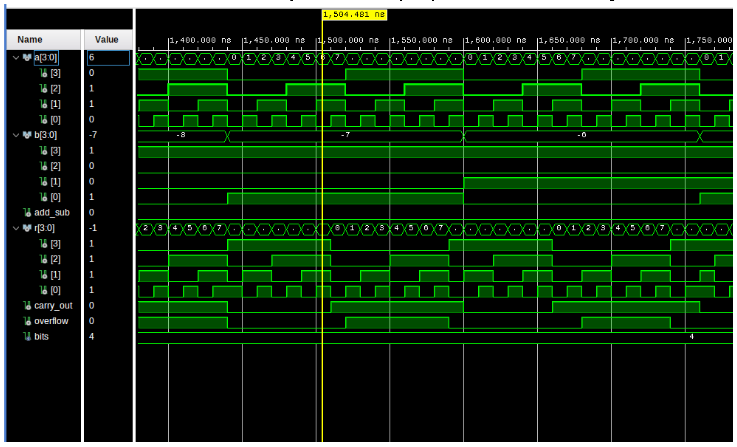


16 bit input (4 shift bits) barrel shifter design



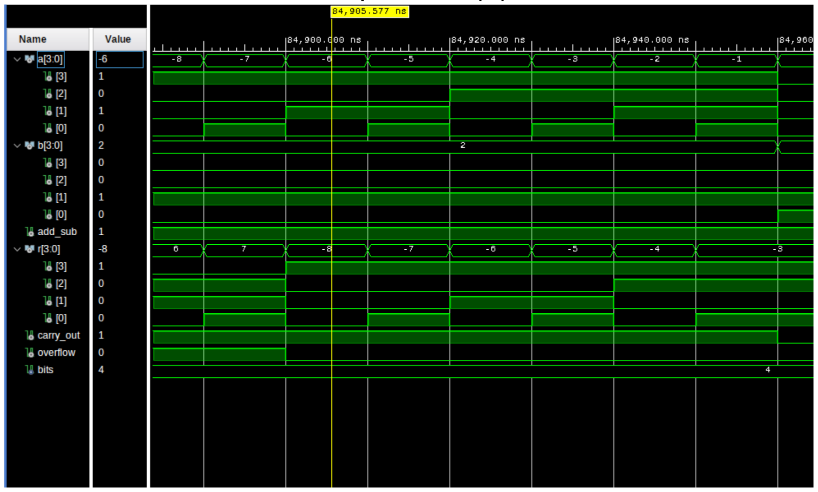
-Simulation-
-Adder/Subtractor-

4 bit adder example: 6 + (-7) = -1: no carry out



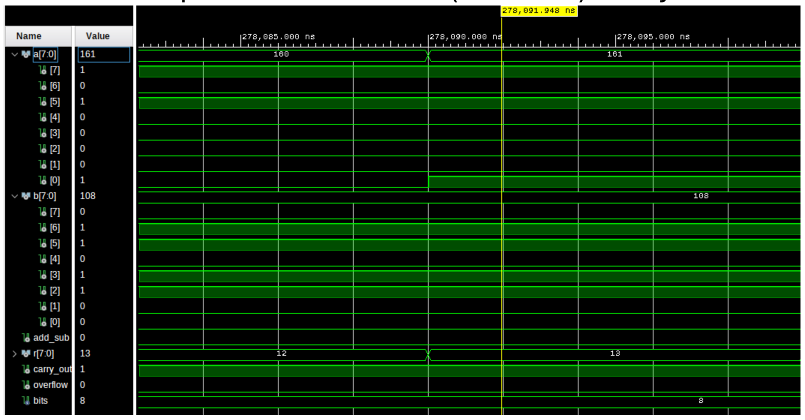
```
Name        Value
a[3:0]      6
  [3]       0
  [2]       1
  [1]       1
  [0]       0
b[3:0]      -7
  [3]       1
  [2]       0
  [1]       0
  [0]       1
add_sub     0
r[3:0]      -1
  [3]       1
  [2]       1
  [1]       1
  [0]       1
carry_out   0
overflow    0
bits        4
```

4 bit subtractor example: -6 – (2) = -8: no overflow



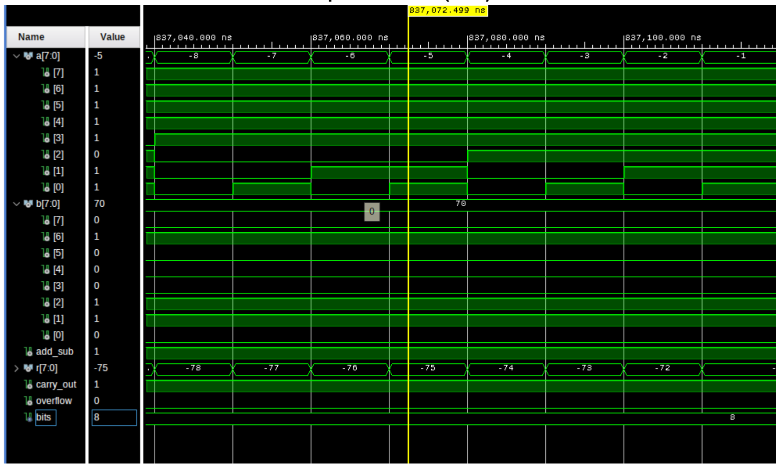
```
Name        Value
a[3:0]      -6
  [3]       1
  [2]       0
  [1]       1
  [0]       0
b[3:0]      2
  [3]       0
  [2]       0
  [1]       1
  [0]       0
add_sub     1
r[3:0]      -8
  [3]       1
  [2]       0
  [1]       0
  [0]       0
carry_out   1
overflow    0
bits        4
```

8 bit adder example: 161 + 108 = 13 (269 -256): carry out asserted



```
Name        Value
a[7:0]      161
  [7]       1
  [6]       0
  [5]       1
  [4]       0
  [3]       0
  [2]       0
  [1]       0
  [0]       1
b[7:0]      108
  [7]       0
  [6]       1
  [5]       1
  [4]       0
  [3]       1
  [2]       1
  [1]       0
  [0]       0
add_sub     0
r[7:0]      13
carry_out   1
overflow    0
bits        8
```

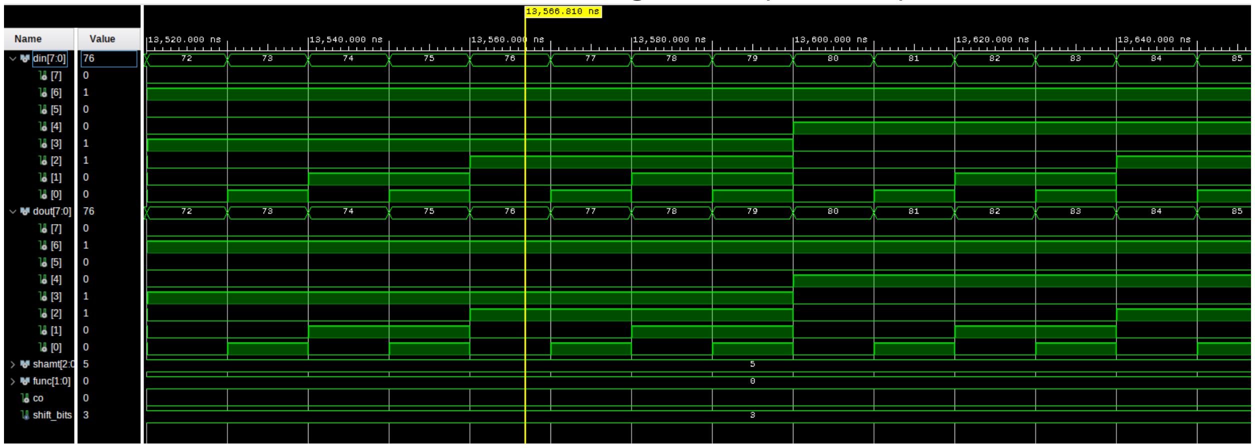
8 bit subtractor example: -5 – (70) = -75: no overflow



Name	Value
a[7:0]	-5
[7]	1
[6]	1
[5]	1
[4]	1
[3]	1
[2]	0
[1]	1
[0]	1
b[7:0]	70
[7]	0
[6]	1
[5]	0
[4]	0
[3]	0
[2]	1
[1]	1
[0]	0
add_sub	1
r[7:0]	-75
carry_out	1
overflow	0
bits	8

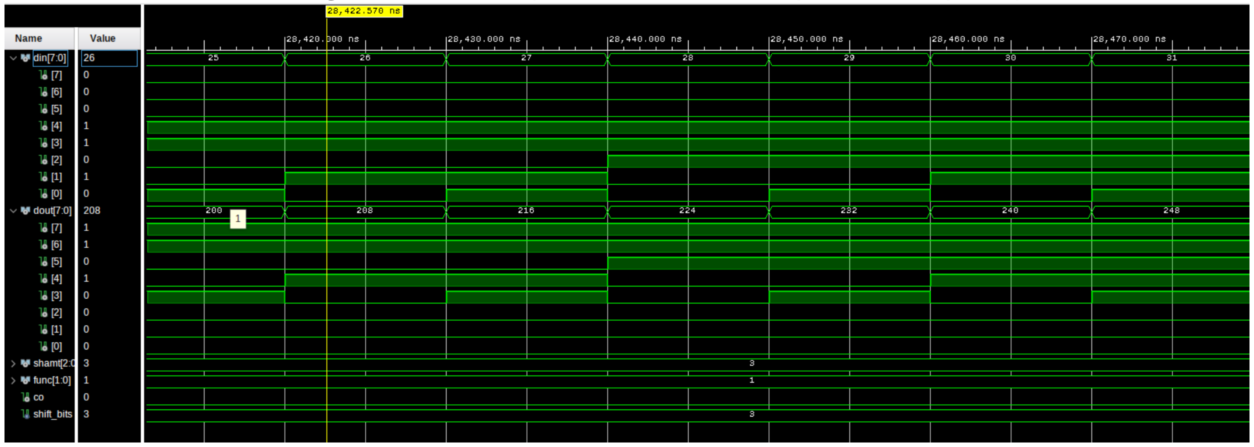
-Barrel Shifter-

Func = “00” so no change from input to output



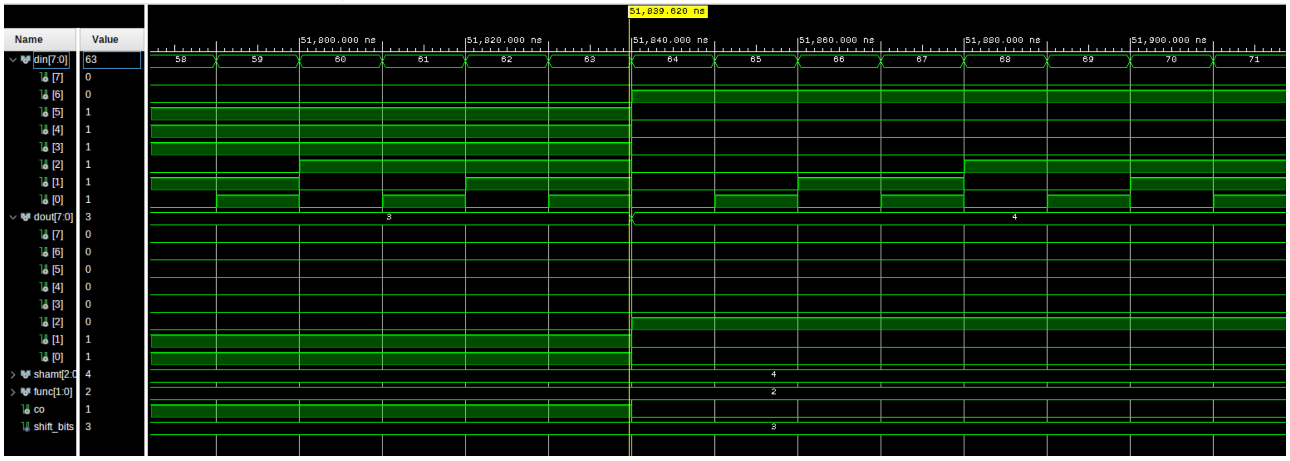
Name	Value
din[7:0]	76
[7]	0
[6]	1
[5]	0
[4]	0
[3]	1
[2]	1
[1]	0
[0]	0
dout[7:0]	76
[7]	0
[6]	1
[5]	0
[4]	0
[3]	1
[2]	1
[1]	0
[0]	0
shamt[2:0	5
func[1:0]	0
co	0
shift_bits	3

Func = “01”: logical shift left by 3 bits which multiplies the input by 2**3=8

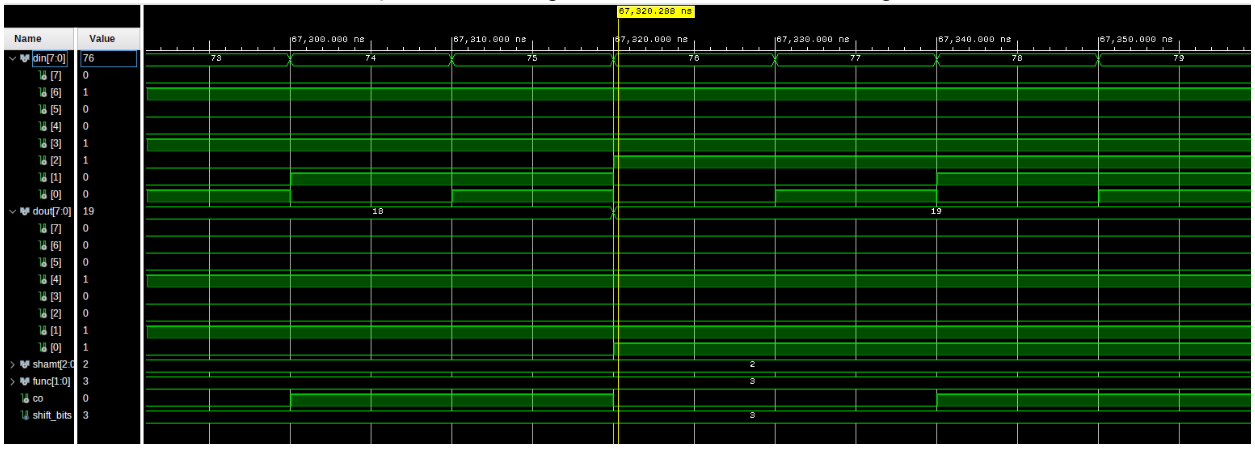


Name	Value
din[7:0]	26
[7]	0
[6]	0
[5]	0
[4]	1
[3]	1
[2]	0
[1]	1
[0]	0
dout[7:0]	208
[7]	1
[6]	1
[5]	0
[4]	1
[3]	0
[2]	0
[1]	0
[0]	0
shamt[2:0	3
func[1:0]	1
co	0
shift_bits	3

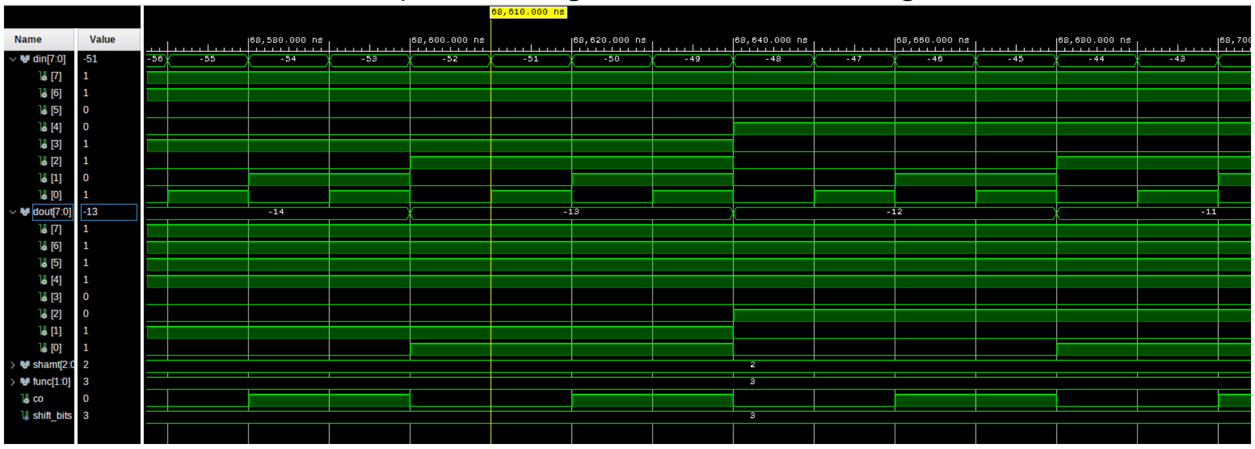
Func = “10”: logical shift left by 4 bits which divides the input by 2**4=16



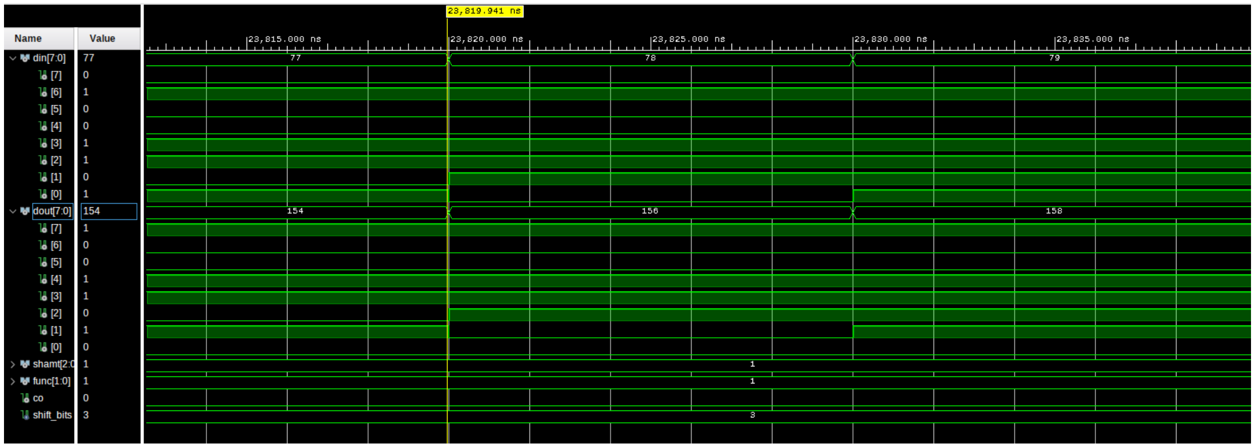
Func = “11”: arithmetic shift left by 2 bits which divides the input by 2**2=4
Example of 0 being filled in as that is the sign bit



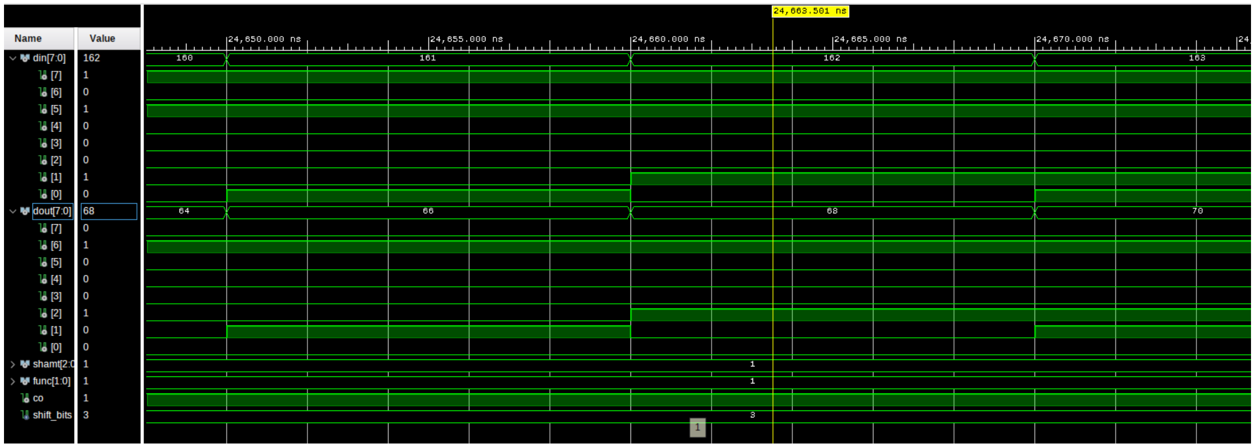
Func = “11”: arithmetic shift left by 2 bits which divides the input by 2**2=4
Example of 1 being filled in as that is the sign



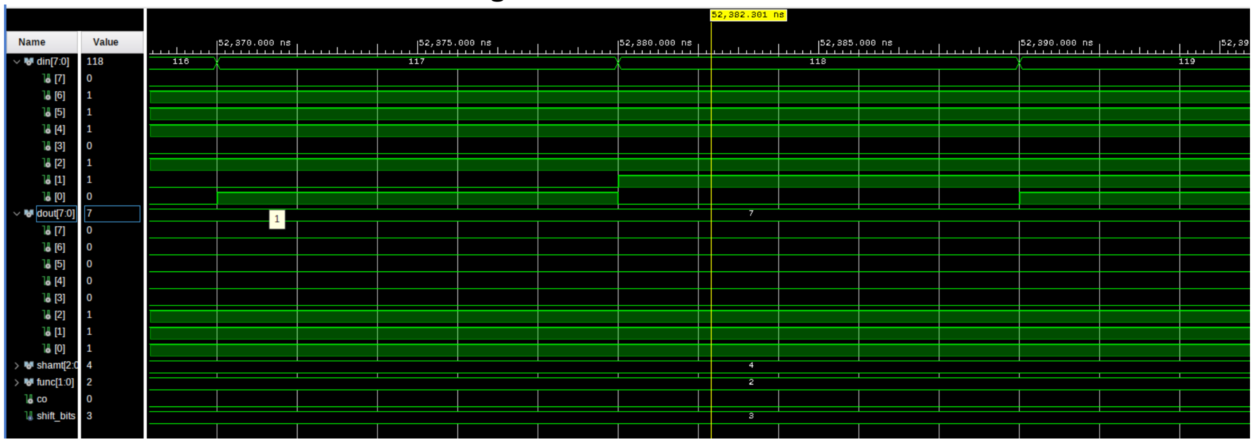
Left shift with a co = ‘0’



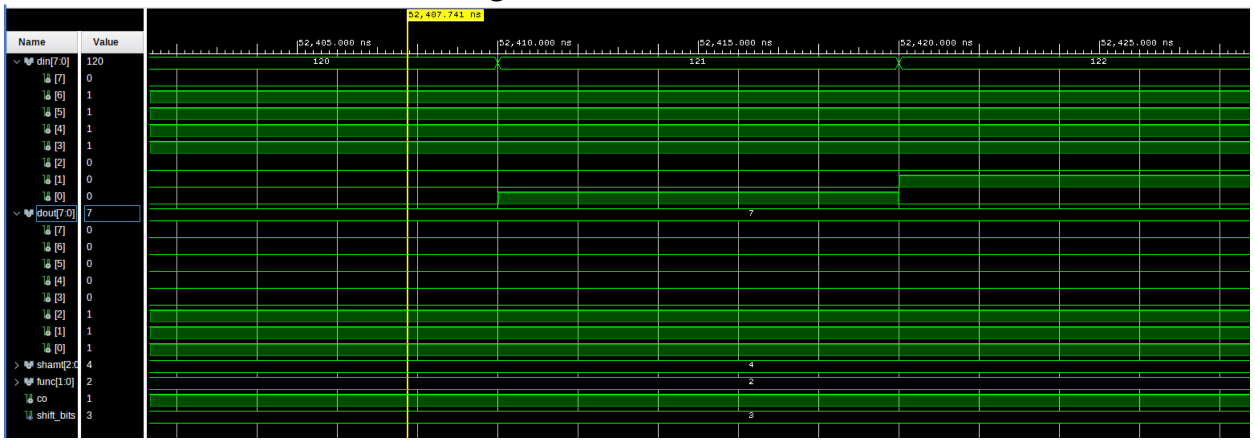
Left shift with a co = ‘1’



Right shift with a co = ‘0’

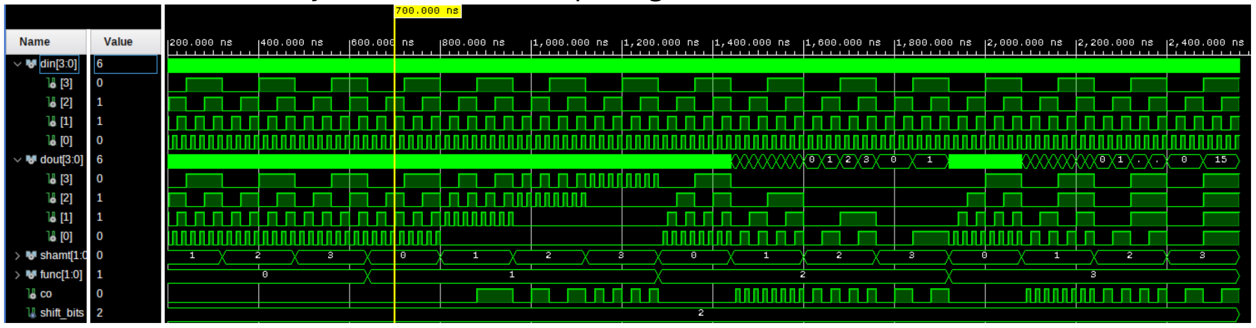


Right shift with a co = ‘1’

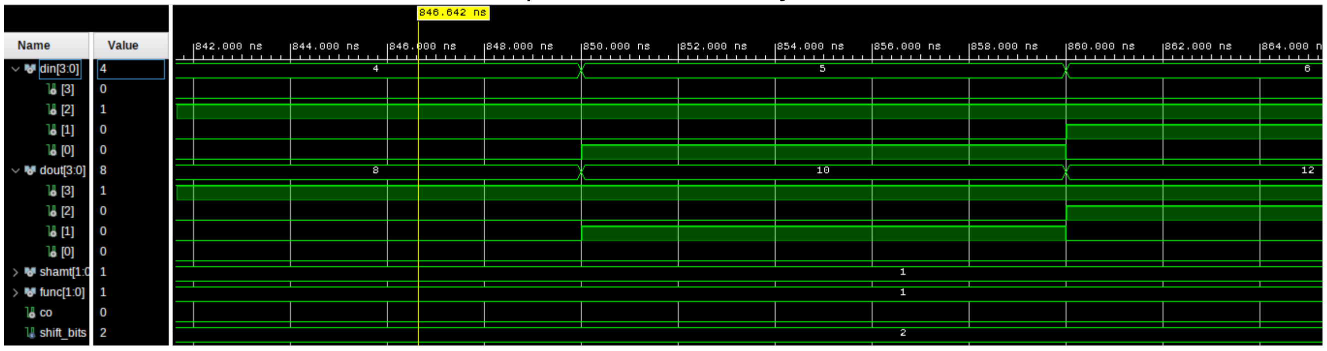


4 bit shifter overview

**Notice how you can see the steps of green as the shift amount increases

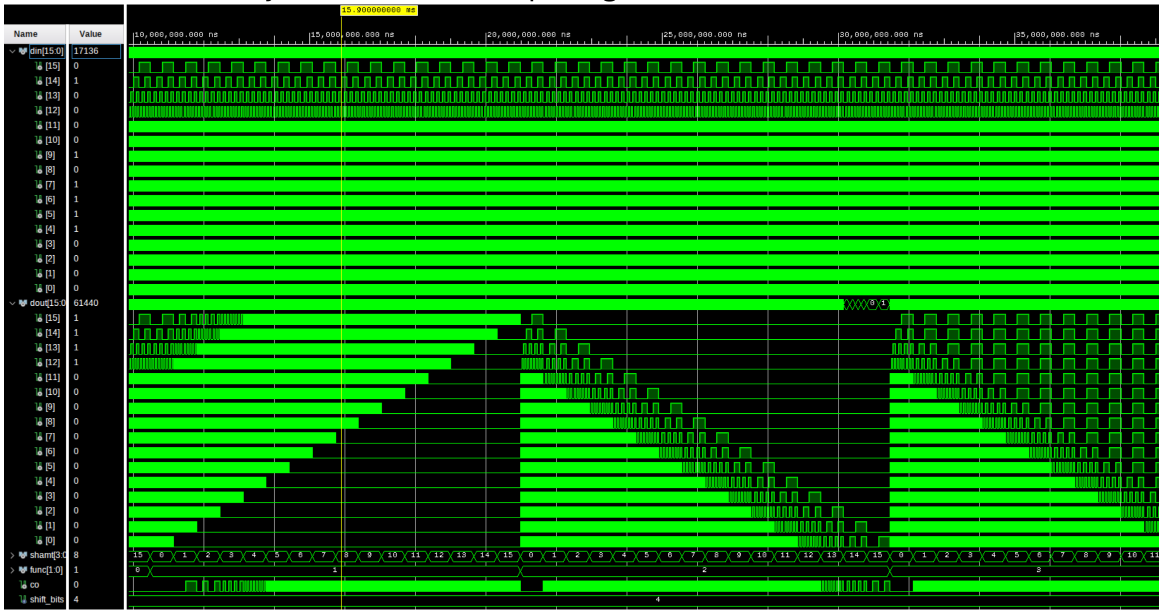


4 bit shifter example: 0100 shifted by 1 left = 1000



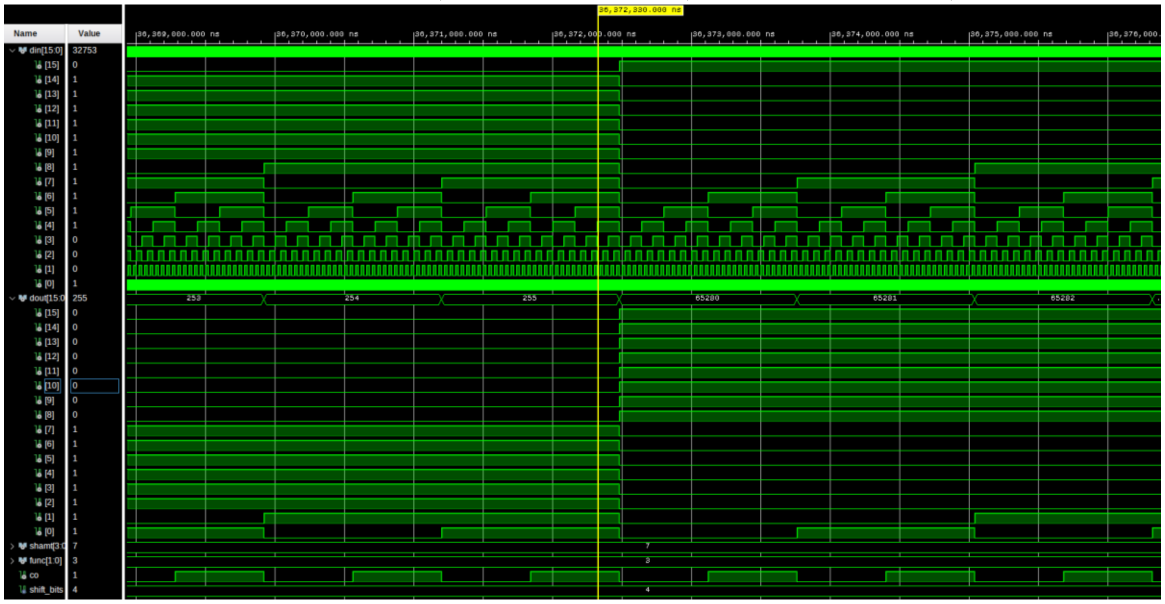
16 bit shifter overview

**Notice how you can see the steps of green as the shift amount increases



16 bit shifter example: arithmetic shift right 0111111111110001 shifted right 7 is 0000000011111111

**Notice that when the msb is 0, zeros are filled in, but when msb is 1, 1s are filled in.



-Implemented Design Verification-

8 rightmost switches are the input number (din)

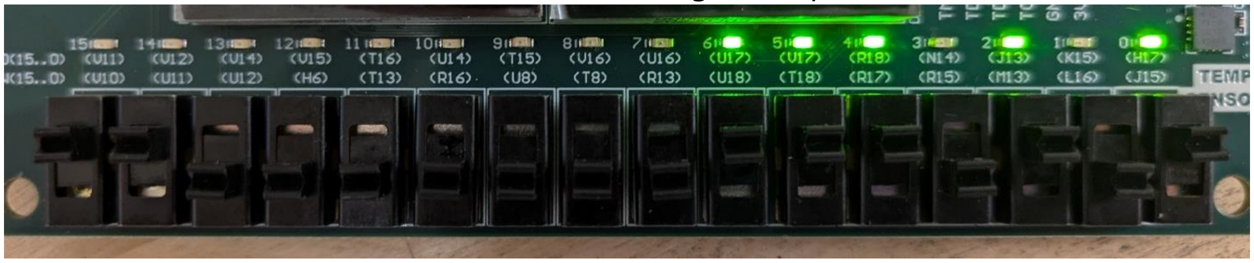
3 leftmost switches are the shift amount (shamt)

Switches 11 and 10 are the function (func)

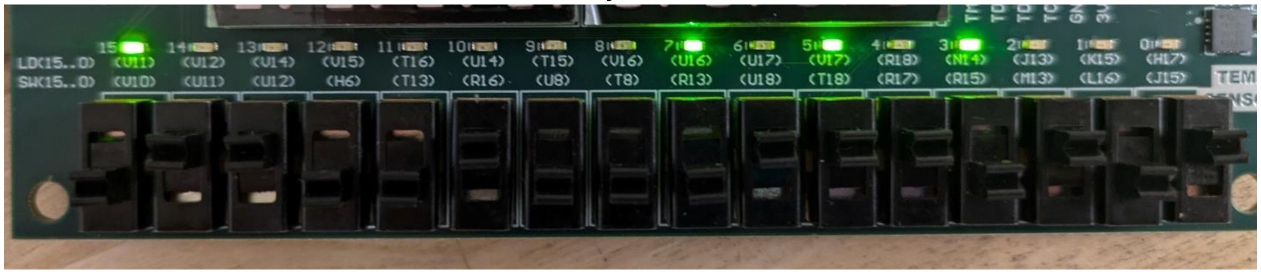
8 rightmost LEDs are the output number (dout)

Leftmost LED is the carry out bit (co)

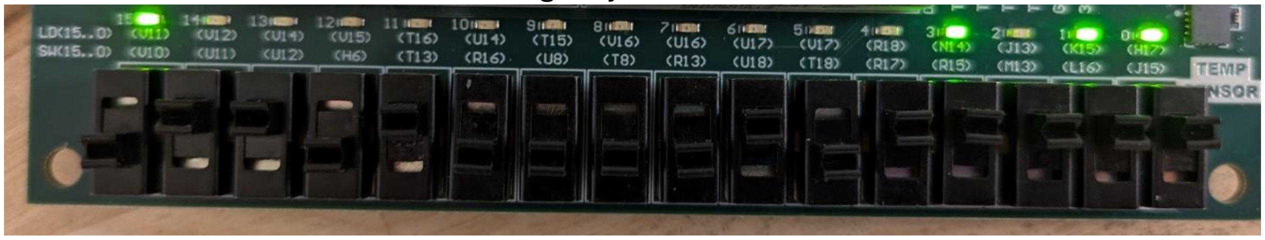
Func = “00”: No change to output



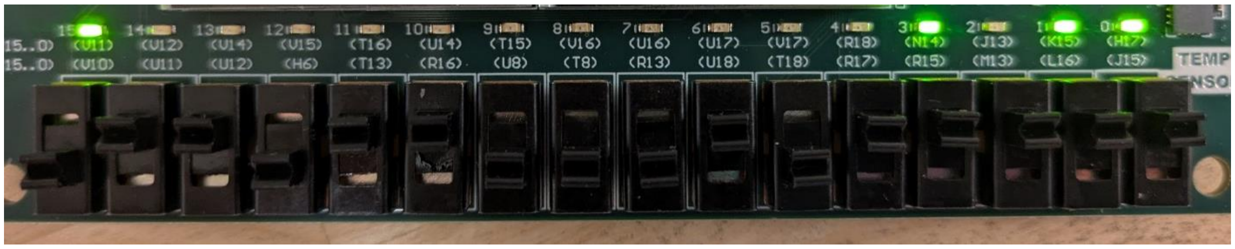
Func = “01”: shift left by 3: 01110101→10101000



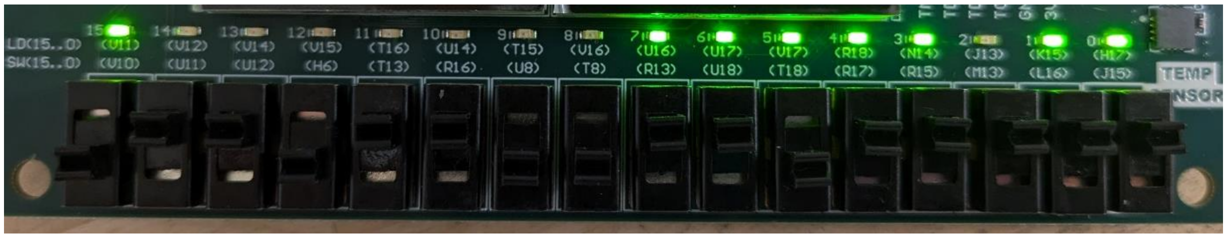
Func = “10”: shift right by 3: 01011111 → 00001011



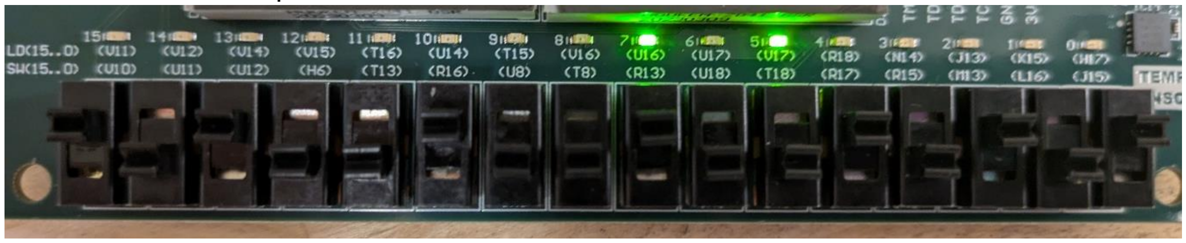
Func = “11”: arithmetic shift right by 3: 01011111 → 00001011
**0 are filled in as msb = 0



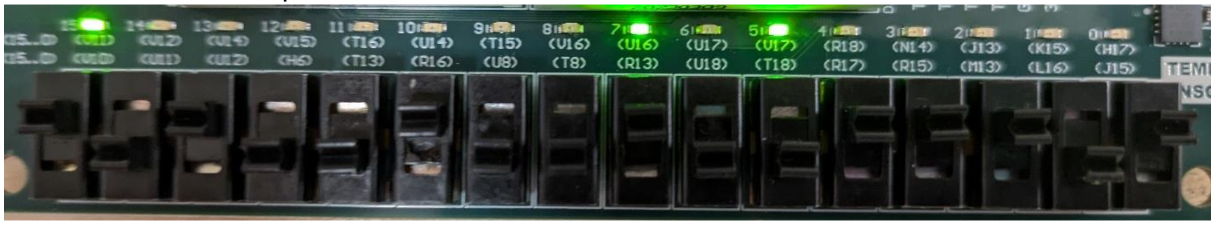
Func = “11”: arithmetic shift right by 3: 11011111 → 11111011
**1 are filled in as msb = 1



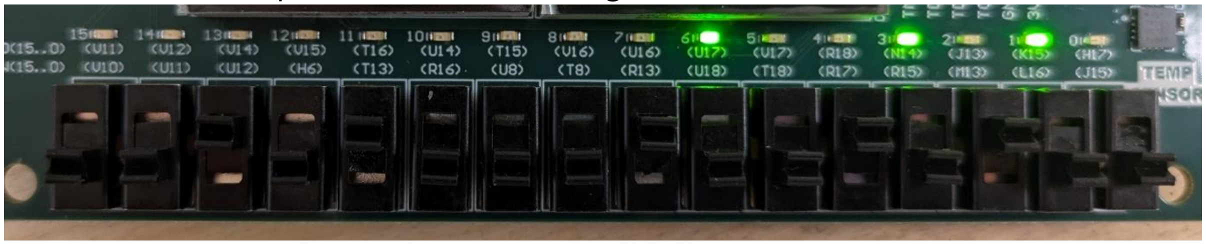
Example of co = 0 after shift left 5: 10010101 → 10100000



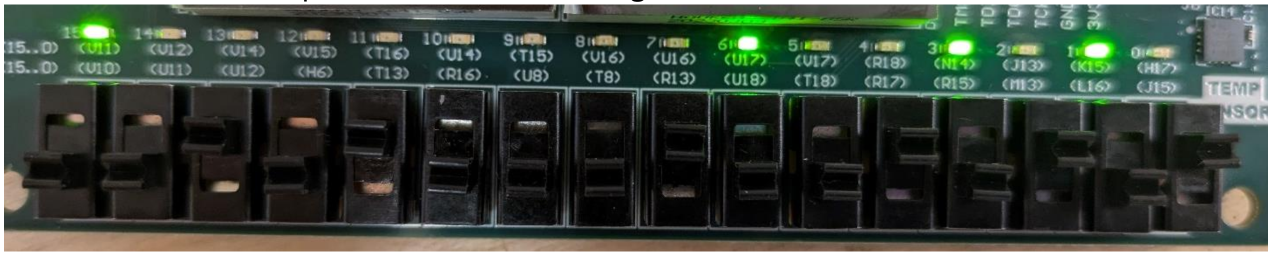
Example of co = 1 after shift left 5: 100111101 → 10100000



Example of co = 0 after shift right 1: 10010100 → 01001010



Example of co = 1 after shift right 1: 10010101 → 01001010



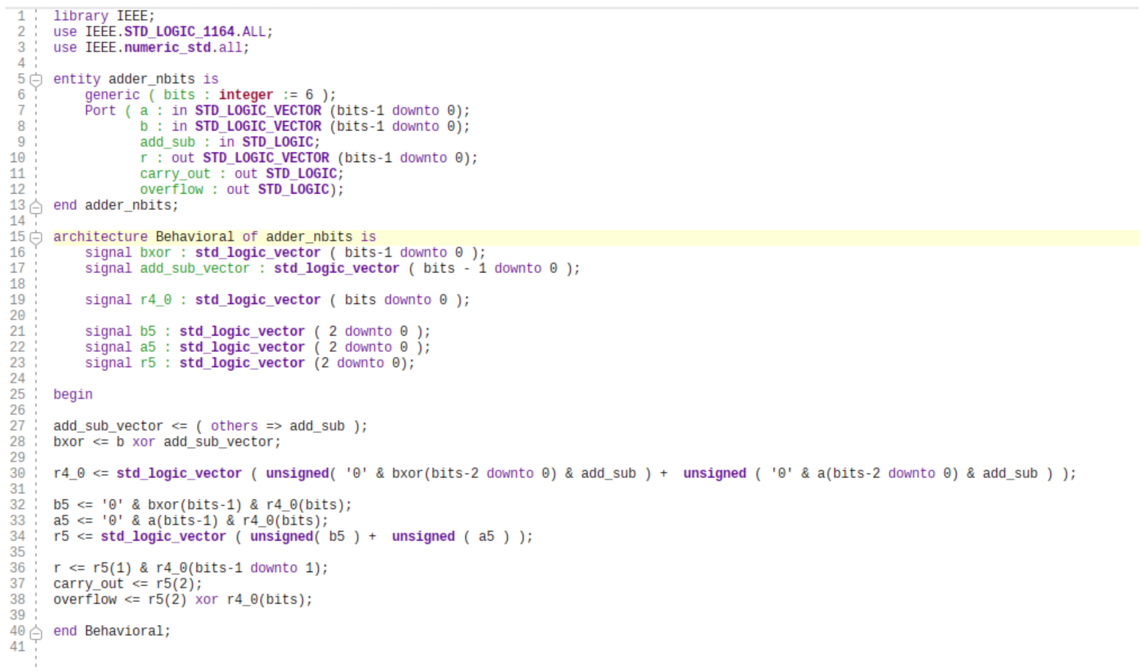
-Conclusion-

In conclusion, this lab was a great learning experience, especially when it came to mastering the generics and generate statements in VHDL. Initially, I struggled with some of the VHDL syntax, but after working through those challenges, I successfully created the adder/subtractor and multi-function barrel shifter. Simulations confirmed that both designs worked as expected, and I was able to implement the shifters without any major issues. This lab helped solidify my understanding of VHDL, and I now feel more confident tackling complex designs in the future.

-Appendix-

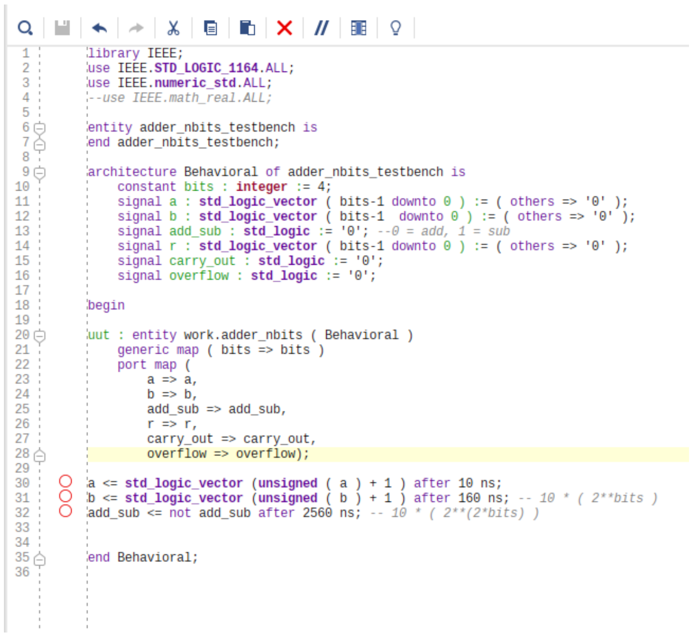
-adder/subtractor-

Main VHDL



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity adder_nbits is
    generic ( bits : integer := 6 );
    Port ( a : in STD_LOGIC_VECTOR (bits-1 downto 0);
           b : in STD_LOGIC_VECTOR (bits-1 downto 0);
           add_sub : in STD_LOGIC;
           r : out STD_LOGIC_VECTOR (bits-1 downto 0);
           carry_out : out STD_LOGIC;
           overflow : out STD_LOGIC);
end adder_nbits;

architecture Behavioral of adder_nbits is
    signal bxor : std_logic_vector ( bits-1 downto 0 );
    signal add_sub_vector : std_logic_vector ( bits - 1 downto 0 );

    signal r4_0 : std_logic_vector ( bits downto 0 );

    signal b5 : std_logic_vector ( 2 downto 0 );
    signal a5 : std_logic_vector ( 2 downto 0 );
    signal r5 : std_logic_vector (2 downto 0);

begin

add_sub_vector <= ( others => add_sub );
bxor <= b xor add_sub_vector;

r4_0 <= std_logic_vector ( unsigned( '0' & bxor(bits-2 downto 0) & add_sub ) +  unsigned ( '0' & a(bits-2 downto 0) & add_sub ) );

b5 <= '0' & bxor(bits-1) & r4_0(bits);
a5 <= '0' & a(bits-1) & r4_0(bits);
r5 <= std_logic_vector ( unsigned( b5 ) +  unsigned ( a5 ) );

r <= r5(1) & r4_0(bits-1 downto 1);
carry_out <= r5(2);
overflow <= r5(2) xor r4_0(bits);

end Behavioral;
```

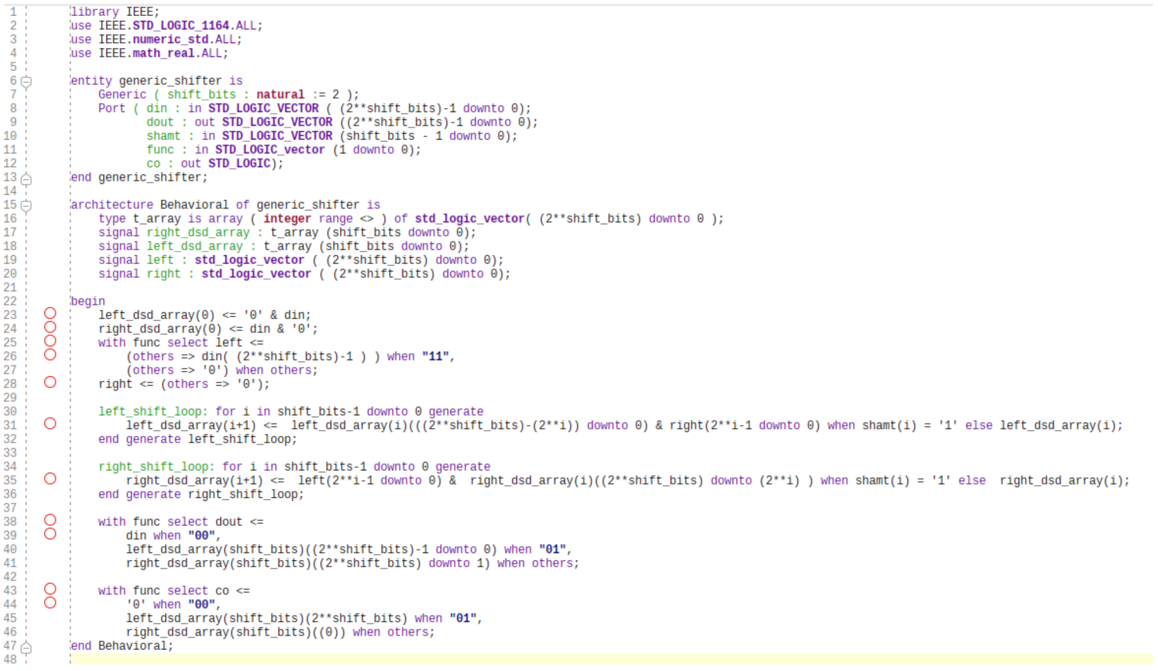
Testbench VHDL



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
--use IEEE.math_real.ALL;

entity adder_nbits_testbench is
end adder_nbits_testbench;

architecture Behavioral of adder_nbits_testbench is
    constant bits : integer := 4;
    signal a : std_logic_vector ( bits-1 downto 0 ) := ( others => '0' );
    signal b : std_logic_vector ( bits-1  downto 0 ) := ( others => '0' );
    signal add_sub : std_logic := '0'; --0 = add, 1 = sub
    signal r : std_logic_vector ( bits-1 downto 0 ) := ( others => '0' );
    signal carry_out : std_logic := '0';
    signal overflow : std_logic := '0';

begin

uut : entity work.adder_nbits ( Behavioral )
    generic map ( bits => bits )
    port map (
        a => a,
        b => b,
        add_sub => add_sub,
        r => r,
        carry_out => carry_out,
        overflow => overflow);

a <= std_logic_vector (unsigned ( a ) + 1 ) after 10 ns;
b <= std_logic_vector (unsigned ( b ) + 1 ) after 160 ns; -- 10 * ( 2**bits )
add_sub <= not add_sub after 2560 ns; -- 10 * ( 2**(2*bits) )


end Behavioral;
```

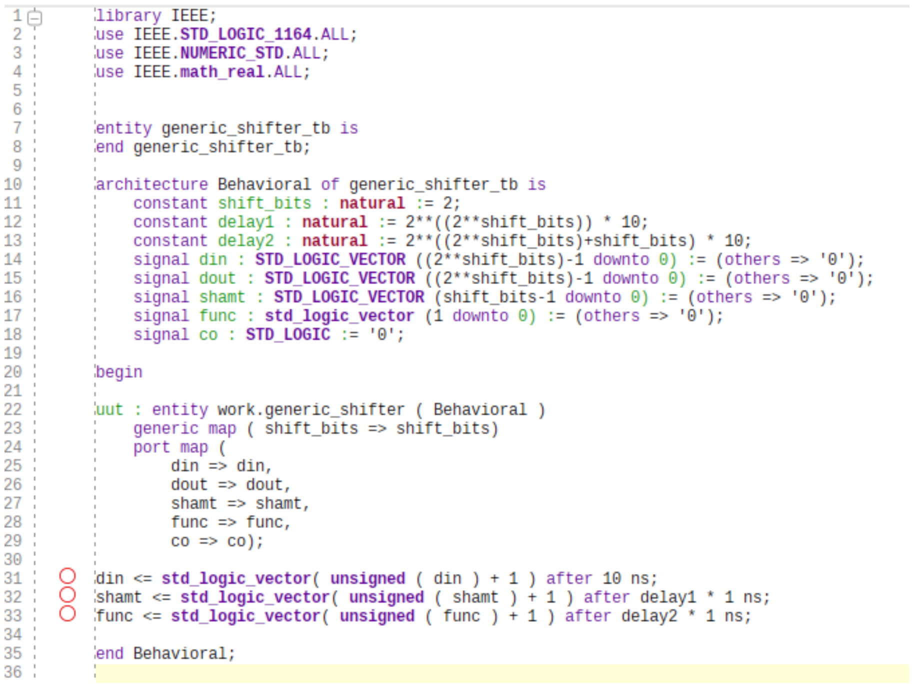
-Barrel Shifter-

Main VHDL



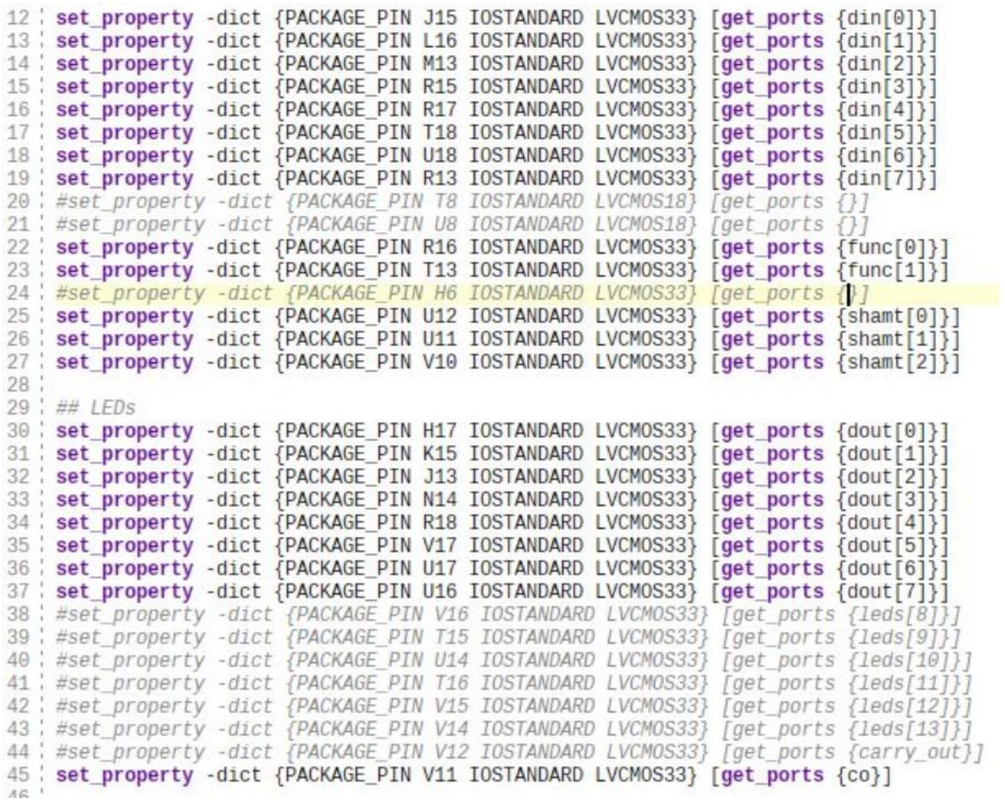
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use IEEE.math_real.ALL;

entity generic_shifter is
    Generic ( shift_bits : natural := 2 );
    Port ( din : in STD_LOGIC_VECTOR ( (2**shift_bits)-1 downto 0);
           dout : out STD_LOGIC_VECTOR ((2**shift_bits)-1 downto 0);
           shamt : in STD_LOGIC_VECTOR (shift_bits - 1 downto 0);
           func : in STD_LOGIC_vector (1 downto 0);
           co : out STD_LOGIC);
end generic_shifter;

architecture Behavioral of generic_shifter is
    type t_array is array ( integer range <> ) of std_logic_vector( (2**shift_bits) downto 0 );
    signal right_dsd_array : t_array (shift_bits downto 0);
    signal left_dsd_array : t_array (shift_bits downto 0);
    signal left : std_logic_vector ( (2**shift_bits) downto 0);
    signal right : std_logic_vector ( (2**shift_bits) downto 0);

begin
    left_dsd_array(0) <= '0' & din;
    right_dsd_array(0) <= din & '0';
    with func select left <=
        (others => din( (2**shift_bits)-1 ) ) when "11",
        (others => '0') when others;
    right <= (others => '0');

    left_shift_loop: for i in shift_bits-1 downto 0 generate
        left_dsd_array(i+1) <=  left_dsd_array(i)(((2**shift_bits)-(2**i)) downto 0) & right(2**i-1 downto 0) when shamt(i) = '1' else left_dsd_array(i);
    end generate left_shift_loop;

    right_shift_loop: for i in shift_bits-1 downto 0 generate
        right_dsd_array(i+1) <=  left(2**i-1 downto 0) &  right_dsd_array(i)((2**shift_bits) downto (2**i) ) when shamt(i) = '1' else  right_dsd_array(i);
    end generate right_shift_loop;

    with func select dout <=
        din when "00",
        left_dsd_array(shift_bits)((2**shift_bits)-1 downto 0) when "01",
        right_dsd_array(shift_bits)((2**shift_bits) downto 1) when others;

    with func select co <=
        '0' when "00",
        left_dsd_array(shift_bits)(2**shift_bits) when "01",
        right_dsd_array(shift_bits)((0)) when others;
end Behavioral;

```

Testbench VHDL



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


entity generic_shifter_tb is
end generic_shifter_tb;

architecture Behavioral of generic_shifter_tb is
    constant shift_bits : natural := 2;
    constant delay1 : natural := 2**((2**shift_bits)) * 10;
    constant delay2 : natural := 2**((2**shift_bits)+shift_bits) * 10;
    signal din : STD_LOGIC_VECTOR ((2**shift_bits)-1 downto 0) := (others => '0');
    signal dout : STD_LOGIC_VECTOR ((2**shift_bits)-1 downto 0) := (others => '0');
    signal shamt : STD_LOGIC_VECTOR (shift_bits-1 downto 0) := (others => '0');
    signal func : std_logic_vector (1 downto 0) := (others => '0');
    signal co : STD_LOGIC := '0';

begin

uut : entity work.generic_shifter ( Behavioral )
    generic map ( shift_bits => shift_bits)
    port map (
        din => din,
        dout => dout,
        shamt => shamt,
        func => func,
        co => co);

din <= std_logic_vector( unsigned ( din ) + 1 ) after 10 ns;
shamt <= std_logic_vector( unsigned ( shamt ) + 1 ) after delay1 * 1 ns;
func <= std_logic_vector( unsigned ( func ) + 1 ) after delay2 * 1 ns;

end Behavioral;

```

Constraint File



```
set_property -dict {PACKAGE_PIN J15 IOSTANDARD LVCMOS33} [get_ports {din[0]}]
set_property -dict {PACKAGE_PIN L16 IOSTANDARD LVCMOS33} [get_ports {din[1]}]
set_property -dict {PACKAGE_PIN M13 IOSTANDARD LVCMOS33} [get_ports {din[2]}]
set_property -dict {PACKAGE_PIN R15 IOSTANDARD LVCMOS33} [get_ports {din[3]}]
set_property -dict {PACKAGE_PIN R17 IOSTANDARD LVCMOS33} [get_ports {din[4]}]
set_property -dict {PACKAGE_PIN T18 IOSTANDARD LVCMOS33} [get_ports {din[5]}]
set_property -dict {PACKAGE_PIN U18 IOSTANDARD LVCMOS33} [get_ports {din[6]}]
set_property -dict {PACKAGE_PIN R13 IOSTANDARD LVCMOS33} [get_ports {din[7]}]
#set_property -dict {PACKAGE_PIN T8 IOSTANDARD LVCMOS18} [get_ports {}]
#set_property -dict {PACKAGE_PIN U8 IOSTANDARD LVCMOS18} [get_ports {}]
set_property -dict {PACKAGE_PIN R16 IOSTANDARD LVCMOS33} [get_ports {func[0]}]
set_property -dict {PACKAGE_PIN T13 IOSTANDARD LVCMOS33} [get_ports {func[1]}]
#set_property -dict {PACKAGE_PIN H6 IOSTANDARD LVCMOS33} [get_ports {}]
set_property -dict {PACKAGE_PIN U12 IOSTANDARD LVCMOS33} [get_ports {shamt[0]}]
set_property -dict {PACKAGE_PIN U11 IOSTANDARD LVCMOS33} [get_ports {shamt[1]}]
set_property -dict {PACKAGE_PIN V10 IOSTANDARD LVCMOS33} [get_ports {shamt[2]}]

## LEDs
set_property -dict {PACKAGE_PIN H17 IOSTANDARD LVCMOS33} [get_ports {dout[0]}]
set_property -dict {PACKAGE_PIN K15 IOSTANDARD LVCMOS33} [get_ports {dout[1]}]
set_property -dict {PACKAGE_PIN J13 IOSTANDARD LVCMOS33} [get_ports {dout[2]}]
set_property -dict {PACKAGE_PIN N14 IOSTANDARD LVCMOS33} [get_ports {dout[3]}]
set_property -dict {PACKAGE_PIN R18 IOSTANDARD LVCMOS33} [get_ports {dout[4]}]
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports {dout[5]}]
set_property -dict {PACKAGE_PIN U17 IOSTANDARD LVCMOS33} [get_ports {dout[6]}]
set_property -dict {PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports {dout[7]}]
#set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports {leds[8]}]
#set_property -dict {PACKAGE_PIN T15 IOSTANDARD LVCMOS33} [get_ports {leds[9]}]
#set_property -dict {PACKAGE_PIN U14 IOSTANDARD LVCMOS33} [get_ports {leds[10]}]
#set_property -dict {PACKAGE_PIN T16 IOSTANDARD LVCMOS33} [get_ports {leds[11]}]
#set_property -dict {PACKAGE_PIN V15 IOSTANDARD LVCMOS33} [get_ports {leds[12]}]
#set_property -dict {PACKAGE_PIN V14 IOSTANDARD LVCMOS33} [get_ports {leds[13]}]
#set_property -dict {PACKAGE_PIN V12 IOSTANDARD LVCMOS33} [get_ports {carry_out}]
set_property -dict {PACKAGE_PIN V11 IOSTANDARD LVCMOS33} [get_ports {co}]
```