Lab 4 Report

Adder/Subtractor and Barrel Shifter

CENG 342 Digital Systems

Samuel Donovan

Samantha Pfeiffer

2021-02-16

OVERVIEW

The purpose of this lab was to build a generic two's complement adder/subtractor and a generic multi-function barrel shifter.

--- FULL ADDER ---

A single full-adder has three inputs - A, B, and C_{in} - and two outputs - S and C_{out}. The adder adds A, B and C_{in}, and outputs their sum to S. The carry bit C_{out} is set if needed. The truth table for a full adder is shown below, in **Table 1**.

Table 1 Full Adder

IN			OUT	
A	B	C_{in}	S	C_{out}
0	0	0	0	*0*
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

--- ADDER/SUBTRACTOR ---

An adder/subtractor is constructed from a series of full-adder circuits. An n-bit adder is constructed by connecting the C_{in}'s and C_{out}'s of n full-adders in series.

The then functional adder can also be used as a subtractor by feeding in $\overline{B}$ and setting the first C_{in} to '1', as this is effectively adding A and B's 2's complement. In summary, see **Figure 1** and **Table 2**.

Figure 1 n-bit Adder/Subtractor Diagram

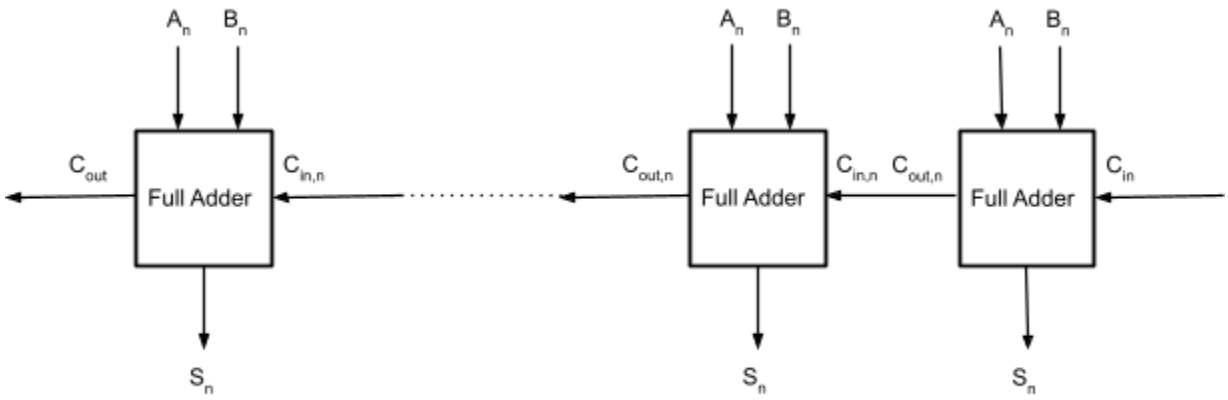


Table 2 Adder/Subtractor Operations

F	Op	Description	Function
00	ADD	Add	$A + B + 0$
01	ADC	Add w/ Carry	$A + B + C_{in}$
10	SUB	Subtract	$A + \overline{B} + 1$
11	SBC	Subtract w/ Carry	$A + \overline{B} + C_{in}$

The last C_{out} represents a carry out of the whole addition. Adding two *n*-bit numbers will at most result in an *n+1*-bit number (e.g. `1111+1111=11110`). The bit at *n+1* is C_{out}.

When subtracting two numbers, an overflow can also occur; this is different from having a carry-out. At a high level, overflow occurs when an n-bit number cannot be represented in it's 2's complement form (negative form) with the same amount of bits. At a low level, the overflow bit is set when the most significant bits of *A* and *B* are the same, but different from the most significant bit of *S*. For example, in 8-bit arithmetic, 15-150 would result in an overflow, as even though 15 and 150 are both able to be represented in 8 bits, -150 is not.

Finally, to select between these different modes of operation, two bits, denoted by *F* are used. What value of *F* corresponds to what operation is shown in **Figure 1**.

As all operations are a variant of addition, only the logical operation for adding needs to be defined.

--- BARREL SHIFTER ---

A shifter takes a string of bits and a shift amount as input. The input string is then shifted left or right by the amount of bits specified. As bits are shifted out, new bits are shifted in, their value depending on the type of shift operation being performed. This shifted string is the output.

In a logical shift left, each bit is shifted to the left by the shift amount specified, with 0s filling their place.

In a logical shift right, each bit is shifted to the right, with 0'swith 0s filling their place.

In an arithmetic shift right, instead of filling with 0's, the bits shifted in are the same value as the original string's most significant bit.

The shifter in this lab can perform all three of the shift operations described above, or perform no shift at all. The following table shows which operation is performed, according to the third input value `func`.

Table 3 Barrel Shifter Modes of Operation

F	Op	Description
00	NOP	No Shift
01	LSL	Logical Shift Left
10	LSR	Logical Shift Right
11	ASR	Arithmetic Shift Right

DESIGN

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

--- FULL ADDER ---

Using the truth table **Table 1**, S and C_{out} are able to defined as a sum of minterms:

$$S(A, B, C_i) = \sum m(1, 2, 4, 7)$$

$$C_{out}(A, B, C_i) = \sum m(3, 5, 6, 7)$$

Using sum-of-products, C_{out} was able to be simplified, but not S. Both results were verified by constructing truth tables by hand.

S

$$S(A, B, C_i) = \sum m(1, 2, 4, 7)$$

Number of 1's	Term	Binary
1	$m1$	001
	$m2$	010
	$m3$	100
3	$m7$	111

				✔	✔	✔	✔
Term	PAttern		Expr	$m1$	$m2$	$m4$	$m7$
$m1$	001	✔	$\overline{A}\overline{B}C_{in}$	X			
$m2$	010	✔	$\overline{A}B\overline{C_{in}}$		X		
$m4$	100	✔	$A\overline{B}\overline{C_{in}}$			X	
$m7$	111	✔	ABC_{in}				X

$$S = \overline{A}\overline{B}C_{in} + \overline{A}B\overline{C_{in}} + A\overline{B}\overline{C_{in}} + AB$$

$\boldsymbol{C_{out}}$

$$C_{out}(A, B, C_{in}) = \sum m(3, 5, 6, 7)$$

Number of 1's	Term	Binary	
2	$m3$	011	✔
	$m5$	101	✔
	$m6$	110	✔
3	$m7$	111	✔

Size 2	Term	Binary	
1	$m(3,7)$	-11	✔
	$m(5,7)$	1-1	✔
	$m(6,7)$	11-	✔

				✔	✔	✔	✔
Term	PAttern		Expr	$m1$	$m2$	$m4$	$m7$
$m(3,7)$	-11	✔	BC_{in}	X			X
$m(5,7)$	1-1	✔	AC_{in}		X		X
$m(6,7)$	11-	✔	AB			X	X

$$C_{out} = BC_i + AC_i + AB$$

--- ADDER/SUBTRACTOR ---

The adder/subtractor circuit was written generically, with *N* being the size of the busses *A*, *B*, and *S*. Using a `generate`, one full-adder was added for each bit.

Each full-adder's input *A* is connected to the respective bit on the *A* input bus.

Each input *B* is connected to a bit on the intermediary signal bus *B_MOD*. When addition is being performed, $BMOD = B$, when subtracting, $BMOD = \overline{B}$.

There is also an intermediary signal bus *Carry*, which connects each full-adder's C_{out} to the next Cin, as well as connecting the adder/subtractors C_{in} and C_{out} to the first and last's full-adder's C_{in} and C_{out}s, respectively.

The first bit in the *Carry* bus is conditionally set to '1' if the operation being performed (determined by *F*) requires it. See **Table 2**.

Each full-adder's *S* bit is connected to its respective bit on the *SUM* signal bus, which intern directly connects to the *S* output bus, but is also used to find *V*.

V is set according to the following logic: $V = A_{MSB}B_{MSB}\overline{SUM_{MSB}} + \overline{A_{MSB}B_{MSB}}SUM_{MSB}$
The *B* referenced is taken before it is inverted for subtraction. See the explanation in **Overview: Adder/Subtractor**.

When writing the adder/subtractor, first the adder was completed and tested, before adding the modifications needed to support subtraction.

--- BARREL SHIFTER ---

The size of the generic multi-function barrel shifter is determined by the amount of bits in the shift amount input, *N*. To generalize this, a shifter entity of variable length, `nbit_shifter`, was created. The logic of this shifter was created using a `generate` statement to chain together *N* 2-to-1 multiplexors. The *A* input for the *Nth* multiplexor is the output of the previous multiplexor. The *B* input is the previous output, but shifted by 2^N bits. For the first multiplexor, *A* is simply the

original input to the shifter.

The value of the input `func` determines which shift operation is performed. This is done in a when else statement when assigning the B value of each multiplexor.

When `func` is ‘00’ no shift is needed so the value of B is set to the value of A.

When `func` is ‘01’ a logical shift left is needed, so the value of B is created by concatenating 2^N 0’s onto the end of the (*BUS_SIZE - 2^N)* lower bits from the previous output.

When `func` is ‘10’ a logical shift right is needed, so the value of B is created by concatenating the (*BUS_SIZE - 2^N)* upper bits from the previous output onto the end of 2^N 0’s.

When `func` is ‘10’ an arithmetic shift right is needed, so the value of B is created by concatenating the (*BUS_SIZE - 2^N)* upper bits from the previous output onto the end of 2^N copies of the most significant bit from the original input.

--- CONSTRAINTS ---

To synthesize the 4-bit Adder/Subtractor, switches controlled input values of A, B, and FUNC, a push button controlled the input value of C_i, and LEDs displayed the output values of S, C_o and V. For more details, see the constraints file for the Adder/Subtractor in **Appendix E.**

To synthesize the 8-bit Barrel Shifter, switches controlled the input values of DIN, SHAMT, and FUNC, and LEDs displayed the output value of DOUT. See the constraints file in **Appendix H** for more details.

SIMULATION RESULTS

Figure 2 Full-Adder Testbench Simulation Result

Figure 3 Adder Testbench Simulation Result

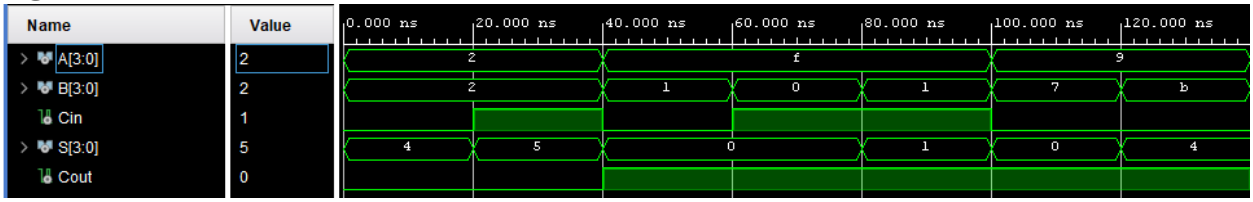


Figure 4 Adder/Subtractor Testbench Simulation Result

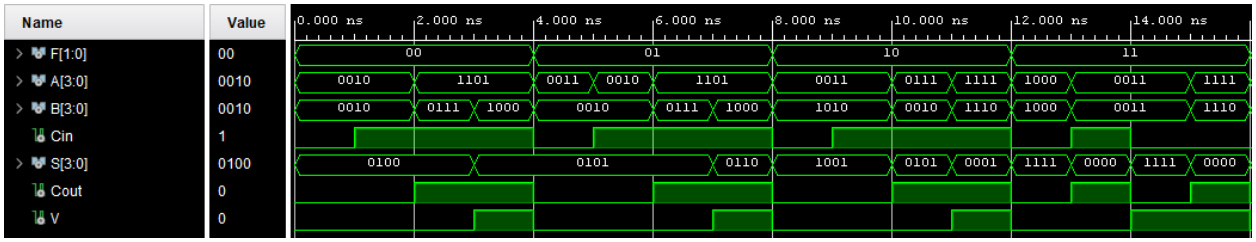
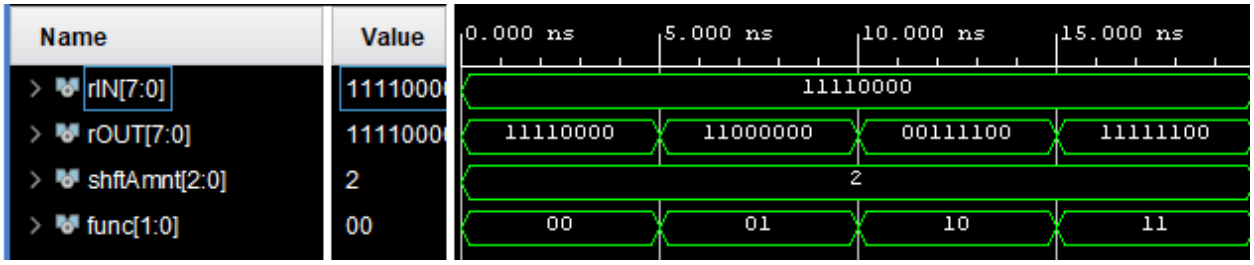


Figure 5 Barrel Shifter Testbench Simulation Result



SYNTHESIS RESULTS

--- ADDER/SUBTRACTOR (4-BIT) ---

Figure 6 F=ADD, A=0b1000, B=0b1000 | S=0000, C_{out}=1, V=1

Figure 7 F=ADC, A=0b0010, B=0b0100, C_i=1 | S=0111, C_{out}=0, V=0

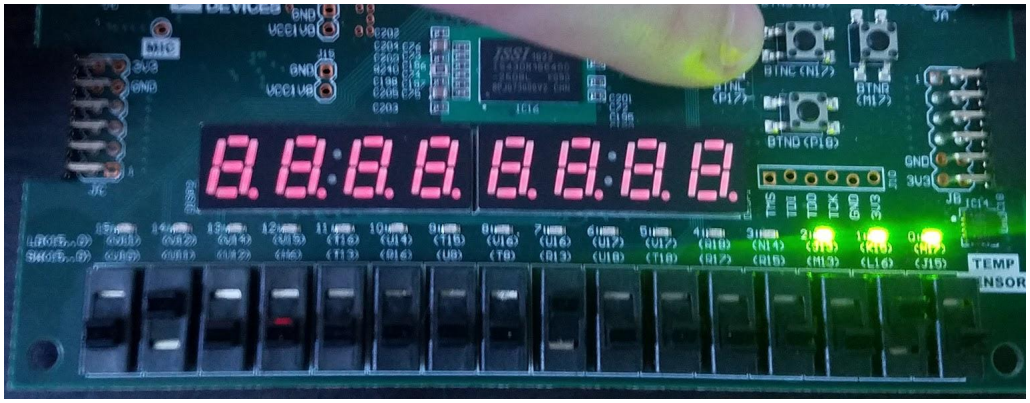


Figure 8 F=SUB, A=0b0001, B=0b1001 | S=1000, C_{out}=0, V=0

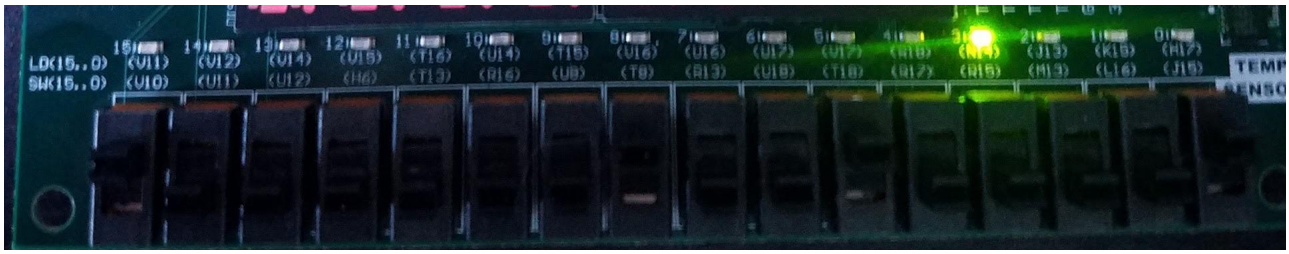
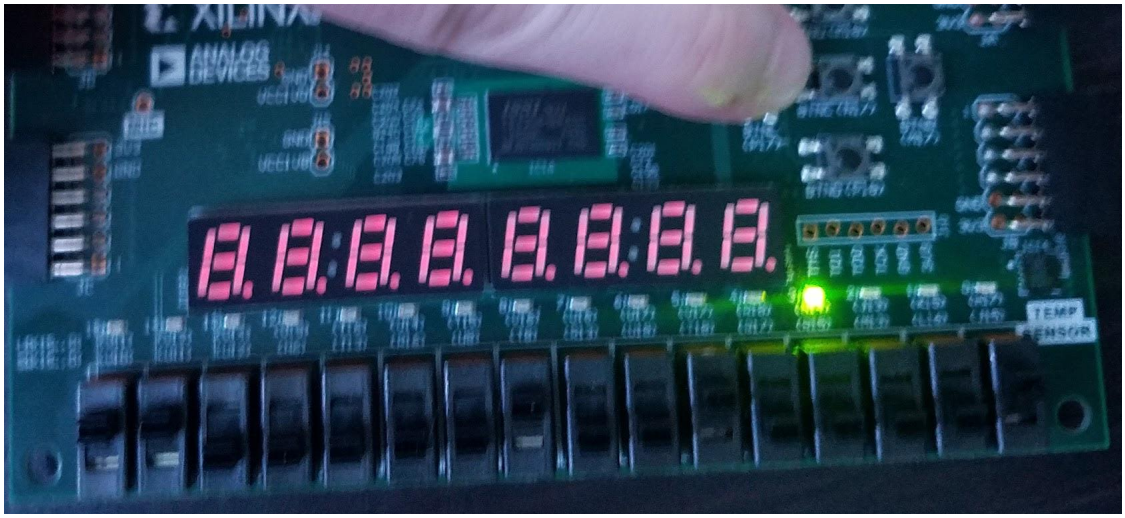


Figure 9 F=SBC, A=0b0001, B=0b1001, C_{in}=0 | S=0111, C_{out}=0, V=0

Figure 10 F=SBC, A=0b0001, B=0b1001, C_{in}=1 | S=1000, C_{out}=0, V=0



--- BARREL SHIFTER (8-BIT) ---

Figure 11 F=ASR, shftAmnt=4, rIN=0b10000011 | rOUT=0b11111000

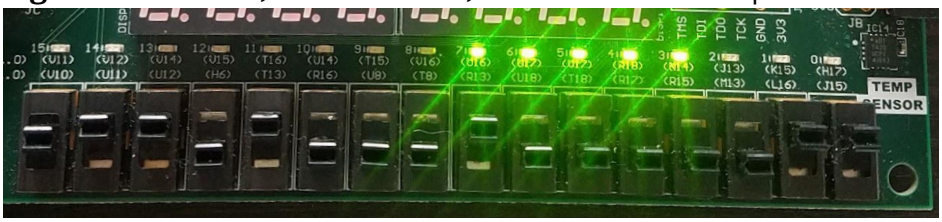


Figure 12 F=LSR, shftAmnt=4 rIN=0b11000000 | rOUT=0b00001100

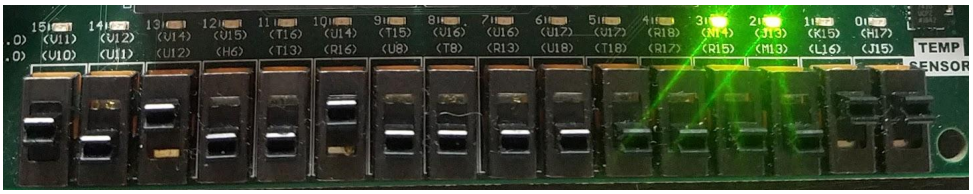
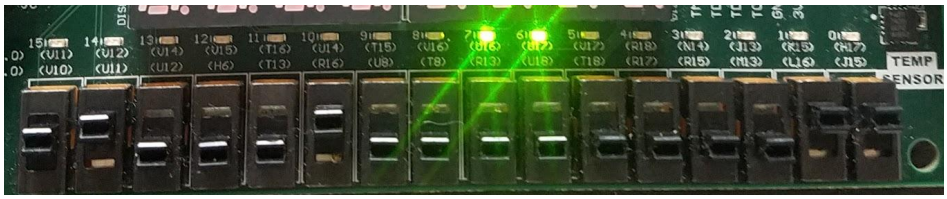


Figure 13 F=LSL, shftAmnt=6 rIN=0b00000011 | rOUT=0b11000000



ISSUES ENCOUNTERED

We found it rather difficult implementing a generically-sized barrel shifter , as it required us to store a 2-D array of bits (bus for each level). Luckily, though much googling, we were able to find a means to accomplish this.

While some bugs were found during testing, they were found to be trivial. Many of them were caused by Sam's ineptitude with the english language and how to spell. Miraculously, after fixed a few typos here and there, the barrel shifter worked flawlessly.

There was also some confusion when it came to the overflow bit in the adder/subtractor, as no real info was given to us regarding when it should be set. It's current state is set by equations shown in **Design: Adder/Subtractor**.

CONCLUSIONS

Though difficult, implementing components generically allows for ease in expansion so any size bus can be accounted for in roughly the same amount of code. Thus, the 4-bit adder/subtractor and the 8-bit shifter could easily be doubled or tripled in size, if only the boards had enough switches. This demonstrates just how powerful generic entities and generate statements are.

In simulation and synthesis, each component worked as expected, with (surprisingly) few issues.

APPENDIX A: Full Adder VHDL Code (full_adder.vhdl)

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

entity full_adder is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           Cin : in STD_LOGIC;
           S : out STD_LOGIC;
           Cout : out STD_LOGIC);
end full_adder;

architecture behavioral of full_adder is

begin
    S <= (NOT A AND NOT B AND Cin)
        OR (NOT A AND B AND NOT Cin)
        OR (A AND NOT B AND NOT Cin)
        OR (A AND B AND Cin);

    Cout <= (B AND Cin)
        OR (A AND Cin)
        OR (A AND B);
end behavioral;
```

APPENDIX B: N-bit Adder VHDL Code (nbit_adder.vhdl)

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

entity nbit_adder is
    generic(N: integer:=4);
    Port ( A : in STD_LOGIC_VECTOR (N-1 downto 0);
           B : in STD_LOGIC_VECTOR (N-1 downto 0);
           Cin : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (N-1 downto 0);
           Cout : out STD_LOGIC);
end nbit_adder;

architecture Behavioral of nbit_adder is
    signal C: STD_LOGIC_VECTOR (N downto 0);
begin
    C(0) <= Cin;

    gen_adders: for i in N-1 downto 0 generate
        full_adderN: entity work.full_adder (Behvioral)
            port map(A=>A(i), B=>B(i), Cin=>C(i), S=>S(i),
Cout=>C(i+1));
    end generate gen_adders;

    Cout <= C(N);
end Behavioral;
```

APPENDIX C: Adder/Subtractor VHDL Code (nbit_adder_subtractor.vhdl)

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

entity nbit_adder_subtractor is
    generic(N: integer:=4);
    Port ( A : in STD_LOGIC_VECTOR (N-1 downto 0);
           B : in STD_LOGIC_VECTOR (N-1 downto 0);
           Cin : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (N-1 downto 0);
           Cout : out STD_LOGIC;
           V : out STD_LOGIC;
           F: in STD_LOGIC_VECTOR (1 downto 0));
end nbit_adder_subtractor;

architecture Behavioral of nbit_adder_subtractor is
    signal Carry: STD_LOGIC_VECTOR (N downto 0);
    signal B_MOD: STD_LOGIC_VECTOR (N-1 downto 0);
    signal SUM: STD_LOGIC_VECTOR (N-1 downto 0);
begin
    Carry(0) <= Cin when F(0)='1' else F(1); --Only set during
carry
    B_MOD <= NOT B when F(1)='1' else B; --Only set during
subtraction

    gen_adders: for i in N-1 downto 0 generate
        full_adderN: entity work.full_adder
            port map(A=>A(i), B=>B_MOD(i), Cin=>Carry(i),
S=>SUM(i), Cout=>Carry(i+1));
    end generate gen_adders;

    V <= (NOT A(N-1) AND NOT B(N-1) AND SUM(N-1)) OR (A(N-1)
AND B(N-1) AND NOT SUM(N-1));
    S <= SUM;
    Cout <= Carry(N);
end Behavioral;
```

APPENDIX D: Adder/Subtractor Testbench VHDL Code (nbit_adder_subtractor_sim.vhdl)

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

entity nbit_adder_subtractor_sim is
end nbit_adder_subtractor_sim;

architecture testbench of nbit_adder_subtractor_sim is
    constant N: integer := 4;
    signal A,B,S: std_logic_vector(N-1 downto 0);
    signal F: std_logic_vector(1 downto 0);
    signal Cin,Cout,V: std_logic;
begin

fourbit_adder: entity work.nbit_adder_subtractor (behavioral)
    generic map(N=>N)
    port map(A=>A, B=>B, Cin=>Cin, S=>S, Cout=>Cout, V=>V,
F=>F);

test: process
    begin
    --ADD
    F <= "00";

    A <= "0010"; --2
    B <= "0010"; --2
    Cin <= '0';  --No Carry
    wait for 1ns; --Expected: 0100

    A <= "0010"; --2
    B <= "0010"; --2
    Cin <= '1';  --Carry
    wait for 1ns; --Expected: 0100

    A <= "1101"; --13
    B <= "0111"; --7
    Cin <= '1';  --Carry
    wait for 1ns; --Expected: 10100 (0100 + Carry)
```

```
A <= "1101"; --13
B <= "1000"; --8
Cin <= '1';  --Carry
wait for 1ns; --Expected: 10101 (0101 + Carry + Overflow)


--ADC
F <= "01";

A <= "0011"; --3
B <= "0010"; --2
Cin <= '0';  --No Carry
wait for 1ns; --Expected: 0101

A <= "0010"; --2
B <= "0010"; --2
Cin <= '1';  --Carry
wait for 1ns; --Expected: 0101

A <= "1101"; --13
B <= "0111"; --7
Cin <= '1';  --Carry
wait for 1ns; --Expected: 10101 (0101 + Carry)


A <= "1101"; --13
B <= "1000"; --8
Cin <= '1';  --Carry
wait for 1ns; --Expected: 10110 (0100 + Carry + Overflow)

--SUB
F <= "10";

A <= "0011"; --3
B <= "1010"; --12 -> 0101
Cin <= '0';  --No Carry (Gets set to 1)
wait for 1ns; --Expected: 1001

A <= "0011"; --3
B <= "1010"; --12 -> 0101
Cin <= '1';  --Carry (Gets set to 1)
wait for 1ns; --Expected: 1001
```

```
    A <= "0111"; --7
    B <= "0010"; --2 -> 1101
    Cin <= '1';  --Carry (Gets set to 1)
    wait for 1ns; --Expected: 10101 (0101 + Carry)

    A <= "1111"; --15
    B <= "1110"; --14 -> 0001
    Cin <= '1';  --Carry (Gets set to 1)
    wait for 1ns; --Expected: 10000 (0000 + Carry + Overflow)

     --SBC
    F <= "11";

    A <= "1000"; --8
    B <= "1000"; --8 -> 0111
    Cin <= '0';  --No Carry
    wait for 1ns; --Expected: 1111

    A <= "0011"; --3
    B <= "0011"; --3 -> 1100
    Cin <= '1';  --Carry
    wait for 1ns; --Expected: 10000 (0000 + Carry)

    A <= "0011"; --3
    B <= "0011"; --3 -> 1100
    Cin <= '0';  --Carry
    wait for 1ns; --Expected: 1111 (Overflow)

    A <= "1111"; --15
    B <= "1110"; --14 -> 0001
    Cin <= '0';  --Carry
    wait for 1ns; --Expected: 10000 (0000 + Carry + Overflow)
end process;
end testbench;
```

APPENDIX E: Adder/Subtractor Constraints (constraints.xdc)

```
##Switches
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 T18   IOSTANDARD LVCMOS33 }
[get_ports { B[0] }];
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 }
[get_ports { B[1] }];
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 }
[get_ports { B[2] }];
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 }
[get_ports { B[3] }];
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 }
[get_ports { F[0] }];
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 }
[get_ports { F[1] }];

## LED
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 }
[get_ports { S[0] }];
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 }
[get_ports { S[1] }];
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 }
[get_ports { S[2] }];
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 }
[get_ports { S[3] }];
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 }
[get_ports { Cout }];
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 }
[get_ports { V }];

##Buttons
set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 }
[get_ports { Cin }];
```

APPENDIX F: Barrel Shifter VHDL Code

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

entity nbit_shifter is
    generic(N_LAYERS : natural := 3);
    port(rIN : in STD_LOGIC_VECTOR(2**N_LAYERS-1 downto 0);
         rOUT : out STD_LOGIC_VECTOR(2**N_LAYERS-1 downto 0);
         shftAmnt : in STD_LOGIC_VECTOR(N_LAYERS-1 downto 0);
         Cout : out STD_LOGIC;
         func : in STD_LOGIC_VECTOR (1 downto 0)
         );
end nbit_shifter;

architecture behavioral of nbit_shifter is
constant BUS_SIZE : natural := 2**N_LAYERS;

type STD_LOGIC_VECTOR_2 is array (integer range<>) of
STD_LOGIC_VECTOR(BUS_SIZE-1 downto 0);
    signal A : STD_LOGIC_VECTOR_2(N_LAYERS-1 downto 0);
    signal B : STD_LOGIC_VECTOR_2(N_LAYERS-1 downto 0);
    signal Y : STD_LOGIC_VECTOR_2(N_LAYERS-1 downto -1);
    signal ZeroVec : STD_LOGIC_VECTOR(BUS_SIZE-1 downto 0);
    signal OneVec : STD_LOGIC_VECTOR(BUS_SIZE-1 downto 0);
begin
    ZeroVec <= (others => '0');
    OneVec <= (others => '1');
    Y(-1) <= rIN;

    muxes: for layer in N_LAYERS-1 downto 0 generate
        muxN: entity work.two_to_one_mux (behavioral)
            generic map(BUS_SIZE=>BUS_SIZE)
            port map(A=>A(layer),
                     B=> B(layer),
                     SEL=>shftAmnt(layer),
                     Y=>Y(layer));
        A(layer) <= Y(layer-1);

        B(layer) <= Y(layer-1)(BUS_SIZE-1-2**layer downto 0) &
ZeroVec(2**layer-1 downto 0) when func="01" else --LSL
```

```
                ZeroVec(2**layer-1 downto 0) &
Y(layer-1)(BUS_SIZE-1 downto 2**layer) when func="10" else
--LSR
                ZeroVec(2**layer-1 downto 0) &
Y(layer-1)(BUS_SIZE-1 downto 2**layer) when (func="11" and
rIN(BUS_SIZE-1)='0') else --ASR (0)
                OneVec(2**layer-1 downto 0) &
Y(layer-1)(BUS_SIZE-1 downto 2**layer) when (func="11" and
rIN(BUS_SIZE-1)='1') else  --ASR (1)
                A(layer); --No shift
    end generate muxes;



    rOUT <= Y(N_LAYERS-1);
end behavioral;
```

APPENDIX G: Barrel Shifter Testbench VHDL Code

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

entity shifter_sim is
end shifter_sim;

architecture testbench of shifter_sim is
    constant N_LAYERS : natural := 3;
    constant BUS_SIZE : natural := 2**N_LAYERS;
    signal rIN : STD_LOGIC_VECTOR(BUS_SIZE-1 downto 0);
    signal rOUT : STD_LOGIC_VECTOR(BUS_SIZE-1 downto 0);
    signal shftAmnt : STD_LOGIC_VECTOR(N_LAYERS-1 downto 0);
    signal Cout : STD_LOGIC;
    signal func : STD_LOGIC_VECTOR(1 downto 0);
begin


shifter: entity work.nbit_shifter (behavioral)
    generic map(N_LAYERS=>N_LAYERS)
    port map(rIN=>rIN, rOUT=>rOUT, shftAmnt=>shftAmnt,
Cout=>Cout, func=>func);


test:   process begin
        func <= "00"; --None
        rIN <= "11110000"; --7
        shftAmnt <= "010";
        wait for 5ns;
        func <= "01"; --LSL
        wait for 5ns;
        func <= "10"; --LSR
        wait for 5ns;
        func <= "11"; --ASR
        wait for 5ns;

    end process;

end testbench;
```

APPENDIX H: Barrel Shifter Constraints (constraints.xdc)

```
##Switches
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 U8    IOSTANDARD LVCMOS18 }
[get_ports { SHAMT[0] }];
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 }
[get_ports { SHAMT[1] }];
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 }
[get_ports { SHAMT[2] }];
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 }
[get_ports { FUNC[0] }];
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 }
[get_ports { FUNC[1] }];

## 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 T15   IOSTANDARD LVCMOS33 }
[get_ports { CO }];
```