FAMU - FSU College of Engineering

Department of Electrical and Computer Engineering Spring 2021 Semester

EEL 4905L - FPLDs Lab Report

Section No: 3

Lab Instructor: Rajesh Thomas

Lab No: Project

Lab Title: FIR Filters

Name: Marc Abad

Date Performed: 4/7/21

Date Delivered: 4/9/21

Contents

1 Introduction

The purpose of this lab is to introduce students to direct and transpose forms of Finite Impulse Response (FIR) Digital Filters. Main objectives of this lab are to develop a direct form FIR filter, a transpose FIR filter, a transpose FIR filter with pipelining, and to compare balance, performance, power, and area optimizations.

2 Requirements

Inputs:

Name	Description
rst	Input clock that drives the simulation.
clk	Reset input to reset internal signals
x	Discrete input at current time

Local:

Name	Description
Shift_reg	Signal of shifted input x
prod	Signal of products of coefficients and input
sum	Signal of sums of products

Outputs:

Name	Description
y	Discrete output at current time

3 Theoretical Design

3.1 Design Narrative

This design is a finite impulse response digital filter. It is designed to implement the following equation.

$$y(n) = \sum_{k=0}^{N-1} h(n)x(n-k)$$

This lab uses both the direct form and transpose form of the FIR filter. Both forms are created by continually adding products. The following figures represent the networks of the direct and transpose form.

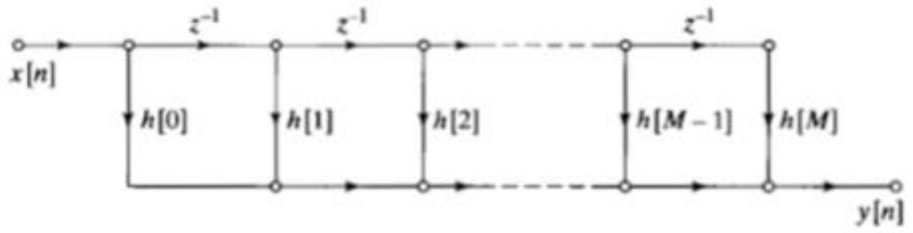


Figure 1. Direct Form Realization of an FIR System

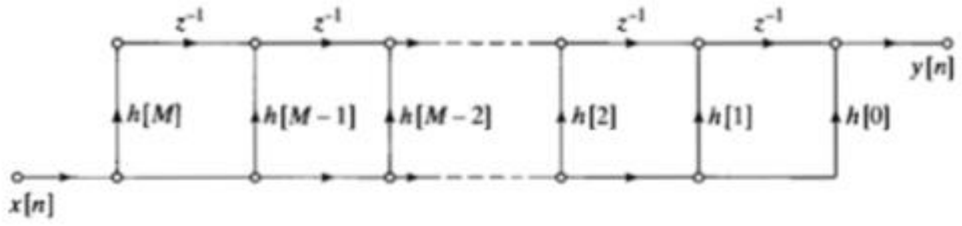
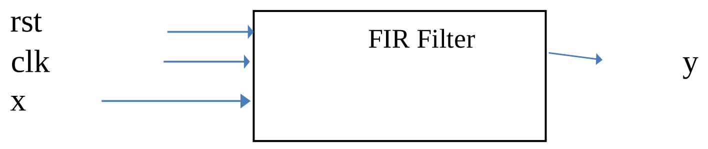


Figure 2. Transposition of the network of Figure 1

3.2 Top-level design



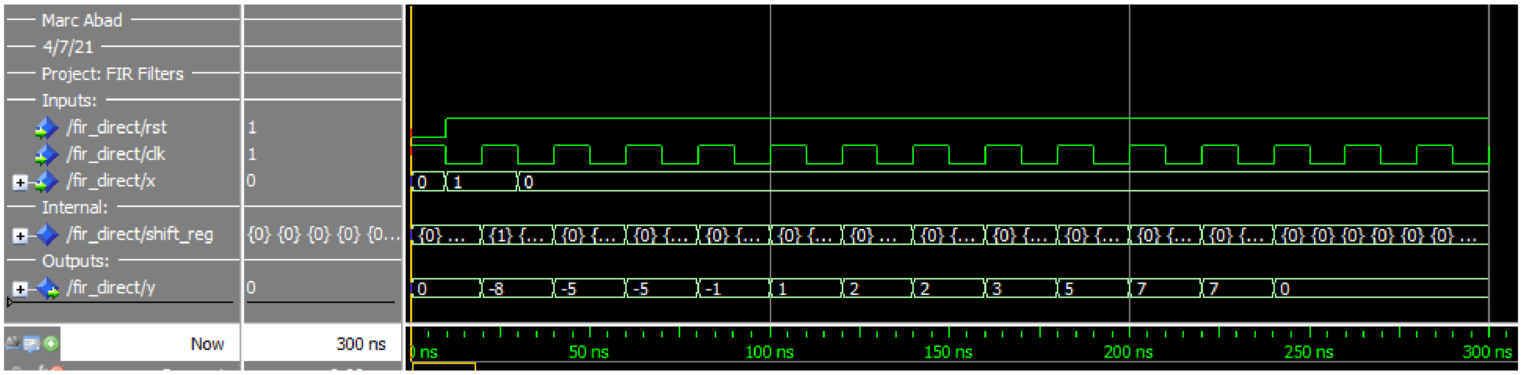
rst – Input signal that will reset all internal signals to 0 and restart the main filter process (shift_reg, prod, sum).
clk - Input signal with period of 20 ns that will either assign shift_reg or sum on rising edge.
x – Input std_logic_vector that is multiplied by filter coefficients in the multipliers.

shift_reg – Signed array equal to the shifted signed value of x
prod – Signed array equal to either the product of coefficients and shift_reg for direct form, or coefficients and x for transpose form.
sum – Signed array that adds the current and previous products.

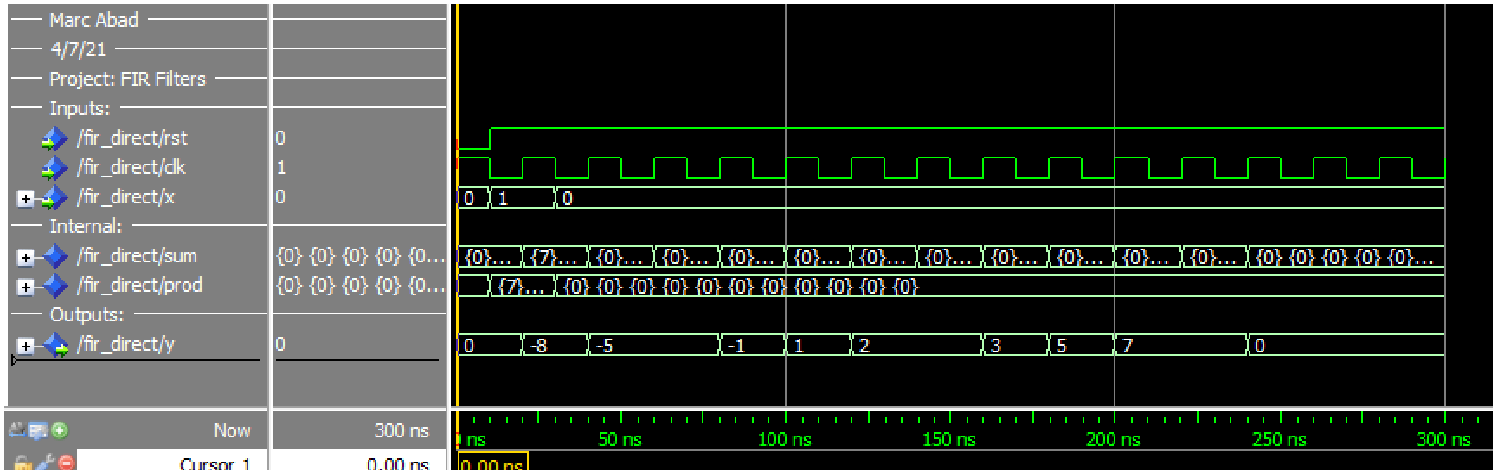
y – Output std_logic_vector equal to the final sum value (NUM_COEF-1 for direct, 1 for transpose).

4 Simulation Results

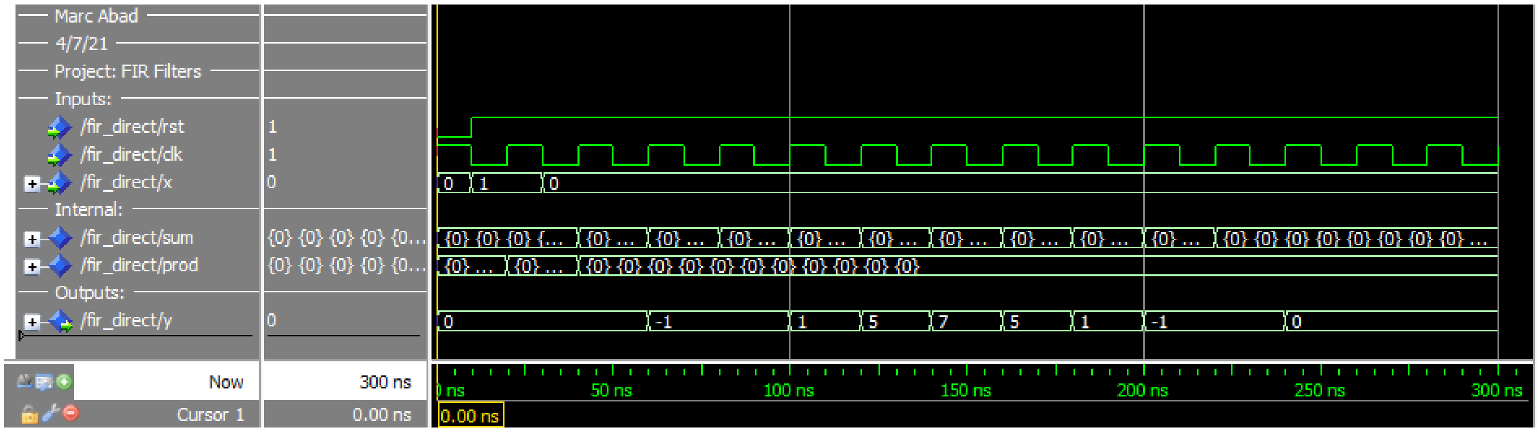
Appendix 1 Simulation Results



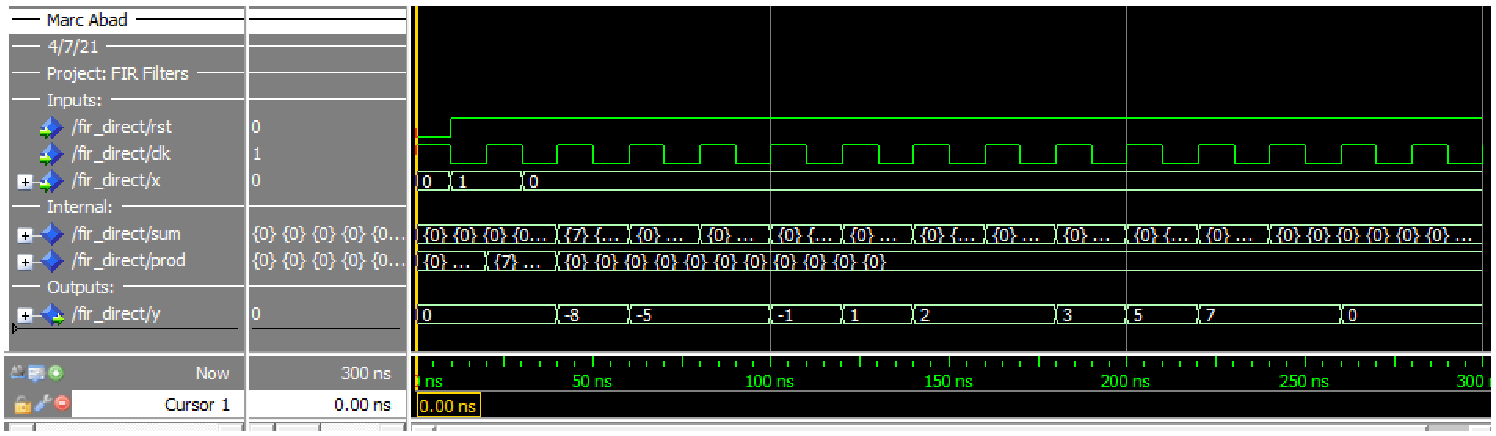
Appendix 3 Simulation Results

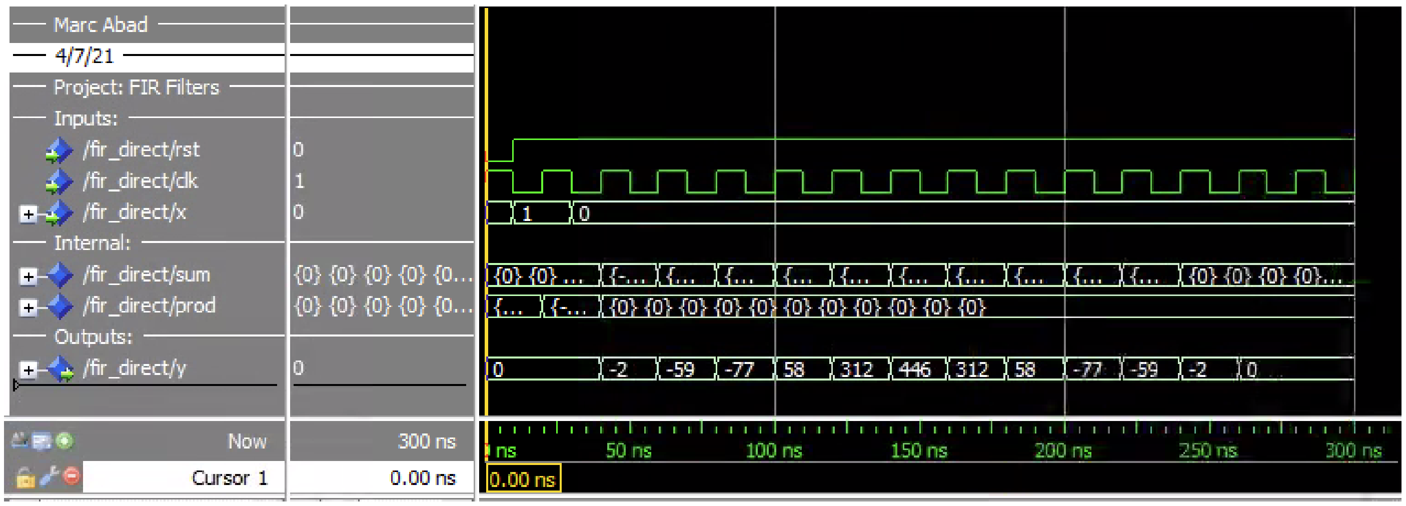


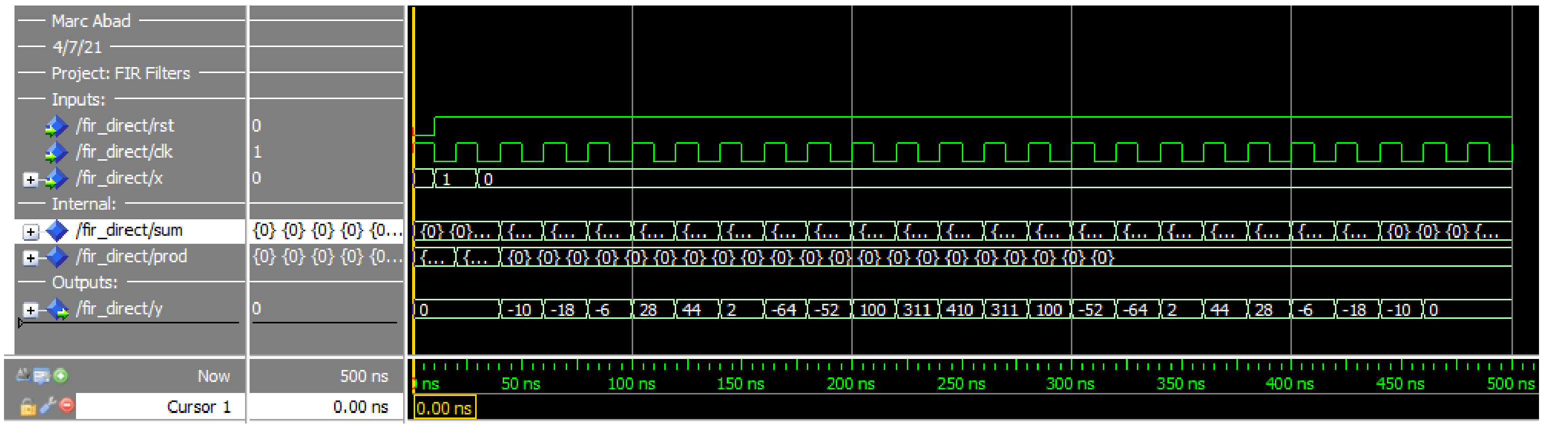
Appendix 4 Simulation Results



Appendix 5 Simulation Results (Low-pass, equiripple FIR filters)







5 Optimization Results

Direct Form FIR Filter

Optimization	Logic Utilization (ALM)	Multipliers	Fmax (MHz)	Power (mW)
Balanced	47	119	494.56	354.39
Performance	47	119	574.05	354.39
Power	47	119	572.41	354.39
Area	47	119	557.41	354.39

Transpose Form FIR Filter

Optimization	Logic Utilization (ALM)	Multipliers	Fmax (MHz)	Power (mW)
Balanced	56	119	334	354.33
Performance	56	119	335.8	354.33
Power	56	119	346.5	354.33
Area	57	119	344	354.33

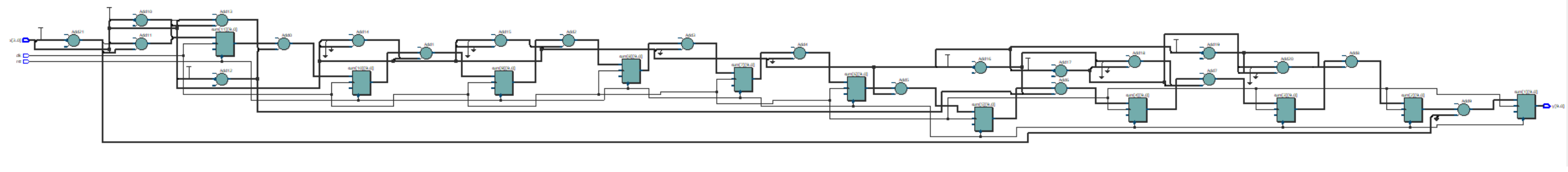
Transpose Form Pipelined FIR Filter

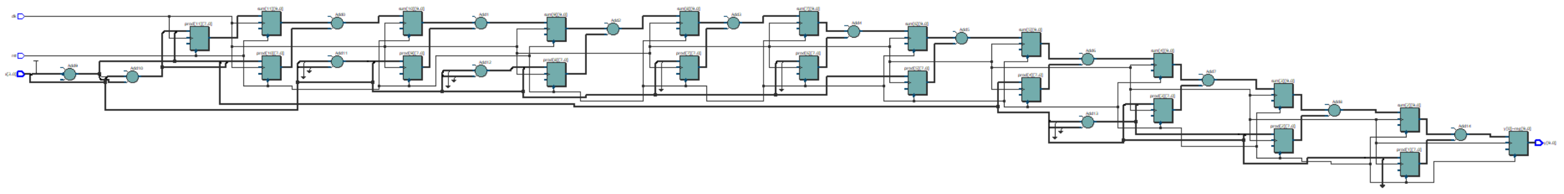
Optimization	Logic Utilization (ALM)	Multipliers	Fmax (MHz)	Power (mW)
Balanced	56	119	304.32	354.39
Performance	56	119	289.94	354.39
Power	56	119	308.07	354.39
Area	56	119	344.35	354.39

6 Discussion

1) A direct form FIR filter was implemented in Appendix 1 and verified via simulation.
2) A transpose form FIR filter without pipelining was implemented in Appendix 3. This was done by converting the affected ranges from (0 TO NUM_COEF-1) to (NUMCOEF DOWNTO 1). This was done in order to follow the transposition figure provided in the Lab Manual (displayed above in 3.1 Design Narrative). An additional change not already implemented would be calculating sum (0 TO NUM_COEF-2) and manually calculating the y output to be sum(2) + prod(1). This would eliminate an extra resistor in the RTL Viewer. This design was verified via simulation.

3) A transpose form FIR filter with pipelining was implemented in Appendix 4. This was done by moving the multiplier and y output statements to the positive edge of the process. This design was verified via simulation. This design also changed the RTL view below and lowered the Fmax overall.





4) The direct form optimizations all utilized 47 ALMs. The transpose form filters all utilized 56 ALMs, except for Area using 1 more at times. All of the filter optimizations utilized 119 multipliers. Balanced optimization had the lowest Fmax, except for the pipelined filter. Performance optimization had the highest Fmax with the direct form filter. Power optimization had the highest Fmax with the transpose form unpipelined filter. Area optimization had the highest Fmax with the transpose form pipelined filter. Estimated power consumption was mostly constant with less used in the transpose form unpipelined filter.

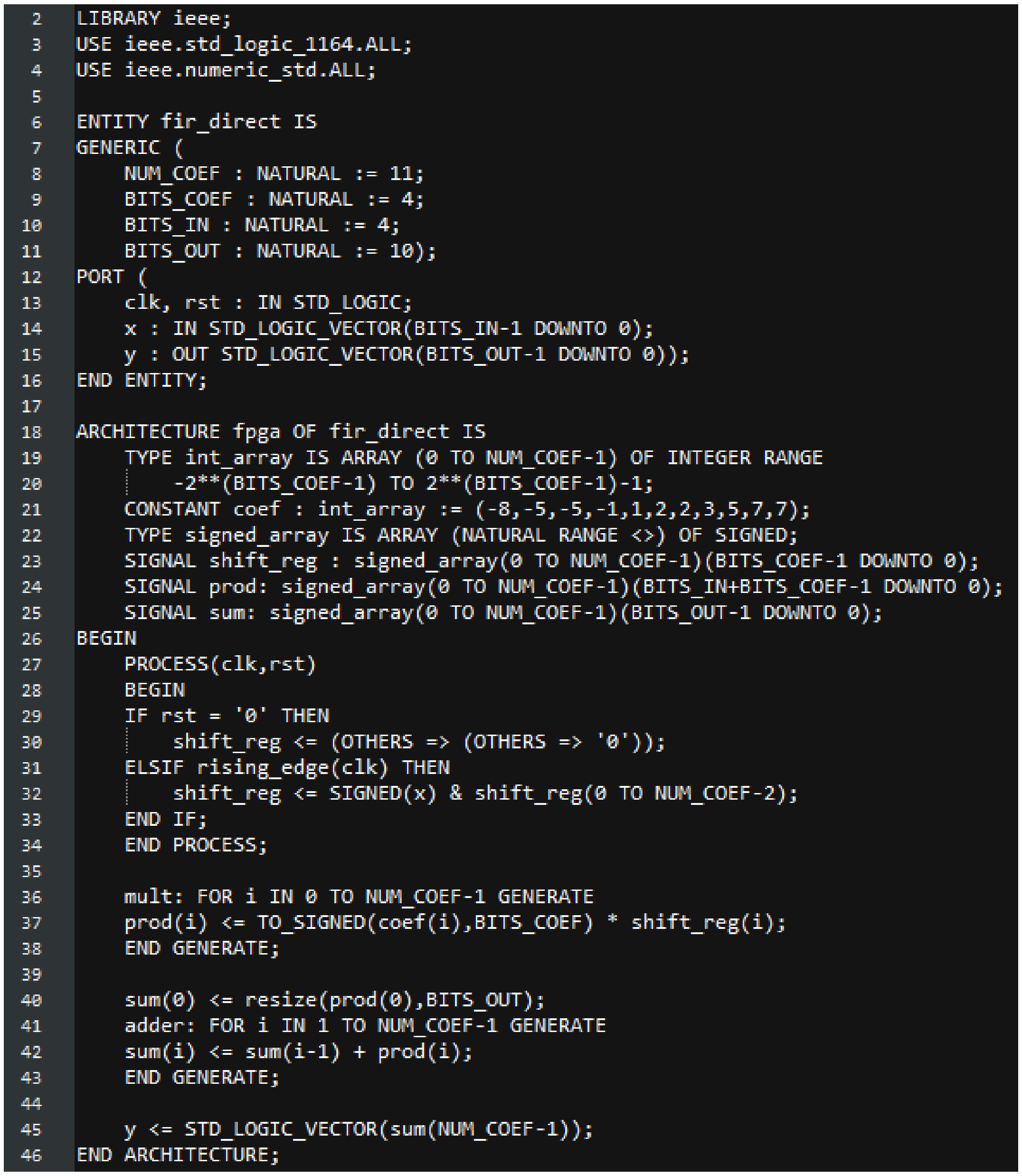
7 Summary and Lessons Learned

In conclusion, the direct form FIR filter, the transpose form FIR filter, and the transpose form pipeline FIR filter were successfully created, tested, and verified through simulation.

One lesson I learned was to create my DO file first and simulate my code while making changes so I can find mistakes faster.

8 Appendix 1

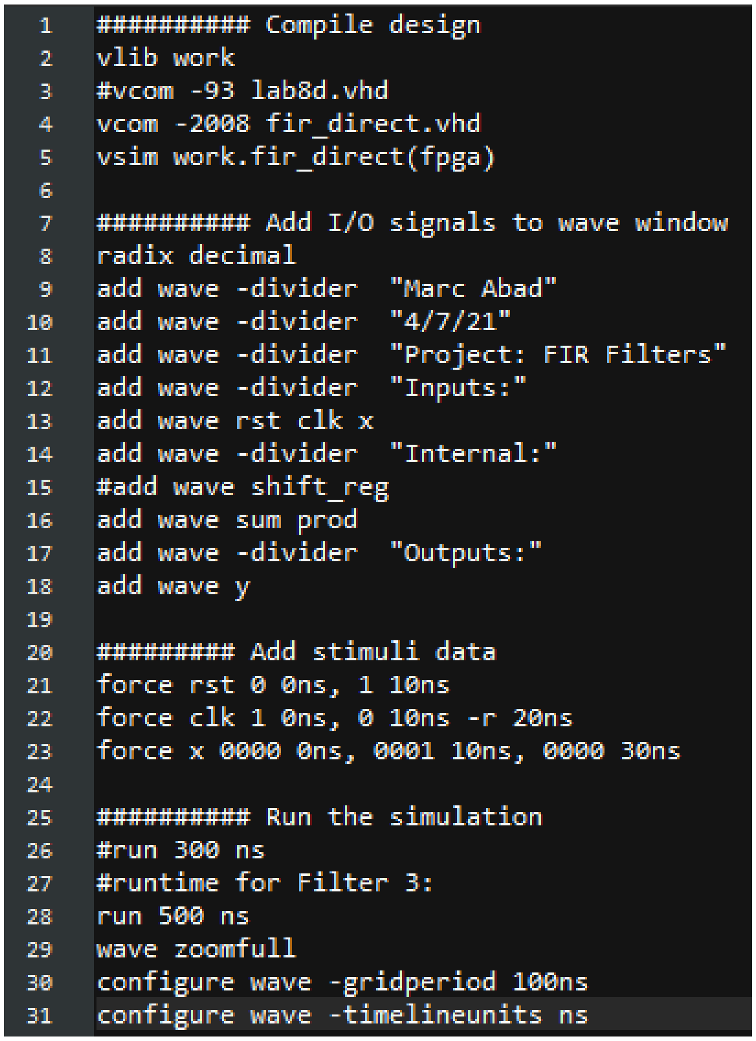
VHDL Code for Direct Form FIR Digital Filter (provided from lab manual)



```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY fir_direct IS
GENERIC (
    NUM_COEF : NATURAL := 11;
    BITS_COEF : NATURAL := 4;
    BITS_IN : NATURAL := 4;
    BITS_OUT : NATURAL := 10);
PORT (
    clk, rst : IN STD_LOGIC;
    x : IN STD_LOGIC_VECTOR(BITS_IN-1 DOWNTO 0);
    y : OUT STD_LOGIC_VECTOR(BITS_OUT-1 DOWNTO 0));
END ENTITY;

ARCHITECTURE fpga OF fir_direct IS
    TYPE int_array IS ARRAY (0 TO NUM_COEF-1) OF INTEGER RANGE
        -2**(BITS_COEF-1) TO 2**(BITS_COEF-1)-1;
    CONSTANT coef : int_array := (-8,-5,-5,-1,1,2,2,3,5,7,7);
    TYPE signed_array IS ARRAY (NATURAL RANGE <>) OF SIGNED;
    SIGNAL shift_reg : signed_array(0 TO NUM_COEF-1)(BITS_COEF-1 DOWNTO 0);
    SIGNAL prod: signed_array(0 TO NUM_COEF-1)(BITS_IN+BITS_COEF-1 DOWNTO 0);
    SIGNAL sum: signed_array(0 TO NUM_COEF-1)(BITS_OUT-1 DOWNTO 0);
BEGIN
    PROCESS(clk,rst)
    BEGIN
    IF rst = '0' THEN
        shift_reg <= (OTHERS => (OTHERS => '0'));
    ELSIF rising_edge(clk) THEN
        shift_reg <= SIGNED(x) & shift_reg(0 TO NUM_COEF-2);
    END IF;
    END PROCESS;

    mult: FOR i IN 0 TO NUM_COEF-1 GENERATE
    prod(i) <= TO_SIGNED(coef(i),BITS_COEF) * shift_reg(i);
    END GENERATE;

    sum(0) <= resize(prod(0),BITS_OUT);
    adder: FOR i IN 1 TO NUM_COEF-1 GENERATE
    sum(i) <= sum(i-1) + prod(i);
    END GENERATE;

    y <= STD_LOGIC_VECTOR(sum(NUM_COEF-1));
END ARCHITECTURE;
```

9 Appendix 2

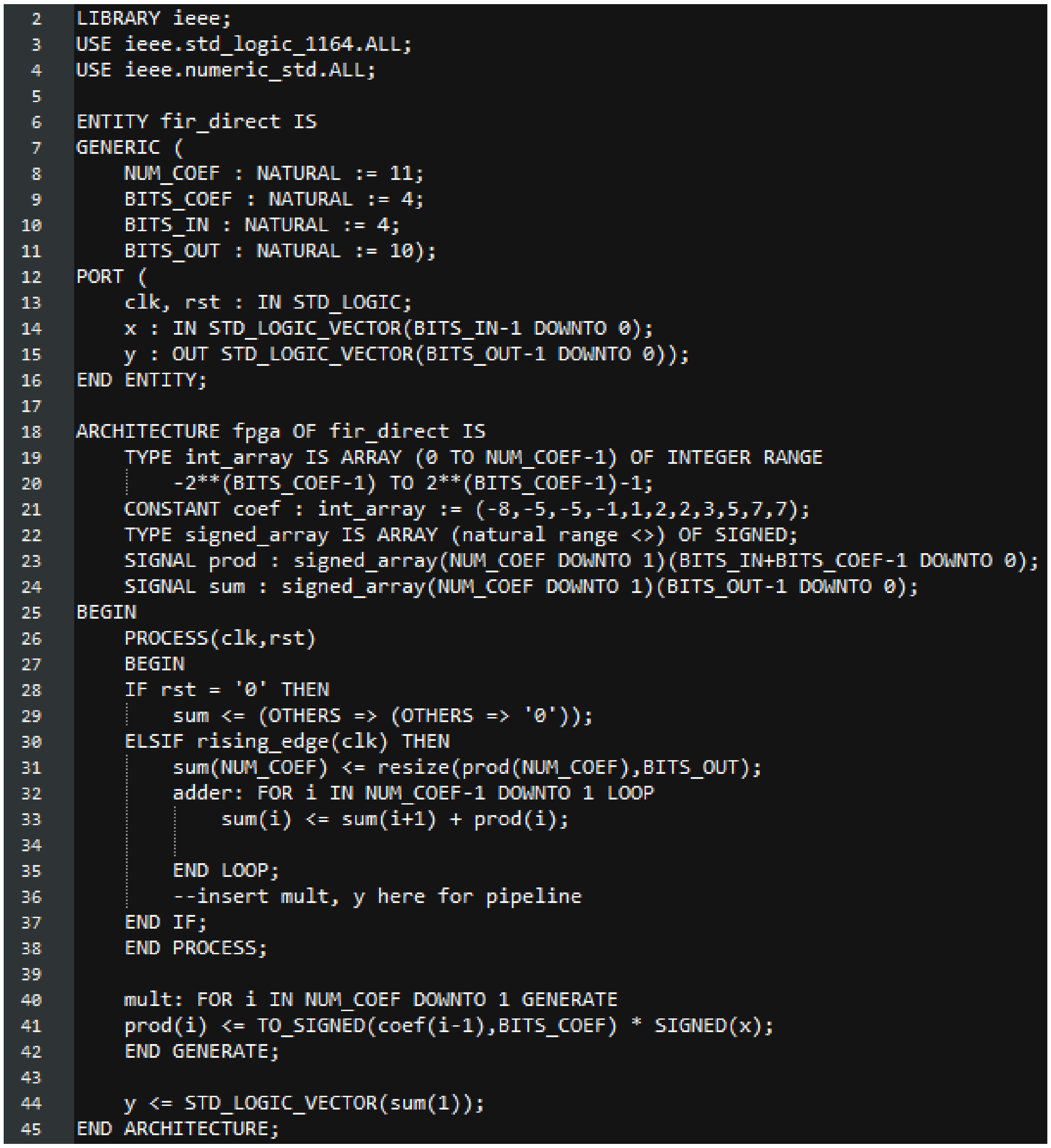
DO Code for ModelSim simulation



```
########## Compile design
vlib work
#vcom -93 lab8d.vhd
vcom -2008 fir_direct.vhd
vsim work.fir_direct(fpga)

########## Add I/O signals to wave window
radix decimal
add wave -divider  "Marc Abad"
add wave -divider  "4/7/21"
add wave -divider  "Project: FIR Filters"
add wave -divider  "Inputs:"
add wave rst clk x
add wave -divider  "Internal:"
#add wave shift_reg
add wave sum prod
add wave -divider  "Outputs:"
add wave y

######### Add stimuli data
force rst 0 0ns, 1 10ns
force clk 1 0ns, 0 10ns -r 20ns
force x 0000 0ns, 0001 10ns, 0000 30ns

########## Run the simulation
#run 300 ns
#runtime for Filter 3:
run 500 ns
wave zoomfull
configure wave -gridperiod 100ns
configure wave -timelineunits ns
```

10 Appendix 3

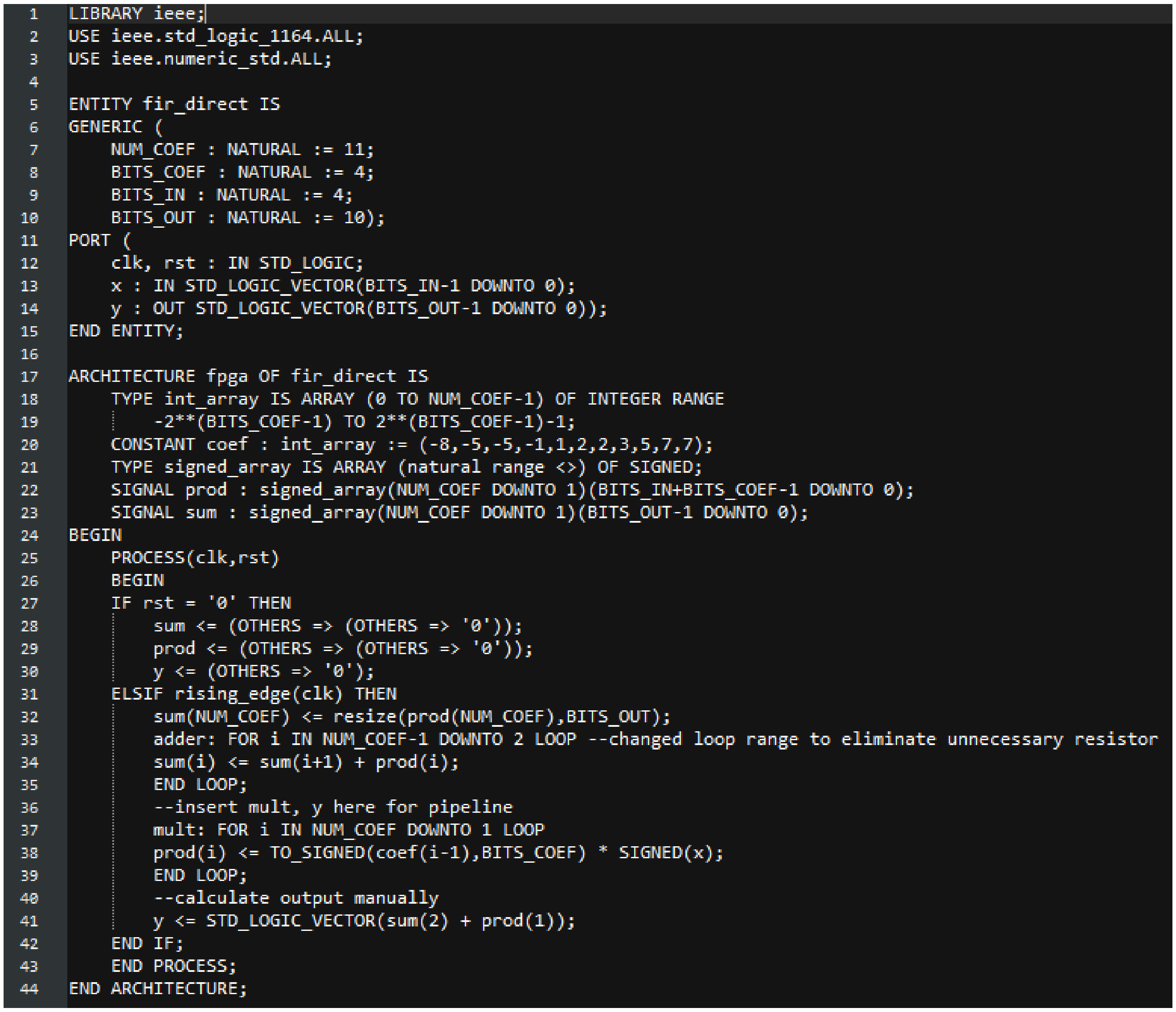
VHDL Code for Transpose Form FIR Digital Filter



```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY fir_direct IS
GENERIC (
    NUM_COEF : NATURAL := 11;
    BITS_COEF : NATURAL := 4;
    BITS_IN : NATURAL := 4;
    BITS_OUT : NATURAL := 10);
PORT (
    clk, rst : IN STD_LOGIC;
    x : IN STD_LOGIC_VECTOR(BITS_IN-1 DOWNTO 0);
    y : OUT STD_LOGIC_VECTOR(BITS_OUT-1 DOWNTO 0));
END ENTITY;

ARCHITECTURE fpga OF fir_direct IS
    TYPE int_array IS ARRAY (0 TO NUM_COEF-1) OF INTEGER RANGE
        -2**(BITS_COEF-1) TO 2**(BITS_COEF-1)-1;
    CONSTANT coef : int_array := (-8,-5,-5,-1,1,2,2,3,5,7,7);
    TYPE signed_array IS ARRAY (natural range <>) OF SIGNED;
    SIGNAL prod : signed_array(NUM_COEF DOWNTO 1)(BITS_IN+BITS_COEF-1 DOWNTO 0);
    SIGNAL sum : signed_array(NUM_COEF DOWNTO 1)(BITS_OUT-1 DOWNTO 0);
BEGIN
    PROCESS(clk,rst)
    BEGIN
    IF rst = '0' THEN
        sum <= (OTHERS => (OTHERS => '0'));
    ELSIF rising_edge(clk) THEN
        sum(NUM_COEF) <= resize(prod(NUM_COEF),BITS_OUT);
        adder: FOR i IN NUM_COEF-1 DOWNTO 1 LOOP
            sum(i) <= sum(i+1) + prod(i);

        END LOOP;
        --insert mult, y here for pipeline
    END IF;
    END PROCESS;

    mult: FOR i IN NUM_COEF DOWNTO 1 GENERATE
    prod(i) <= TO_SIGNED(coef(i-1),BITS_COEF) * SIGNED(x);
    END GENERATE;

    y <= STD_LOGIC_VECTOR(sum(1));
END ARCHITECTURE;
```

11 Appendix 4

VHDL Code for Transpose Form FIR Digital Filter (Pipelined Ver.)

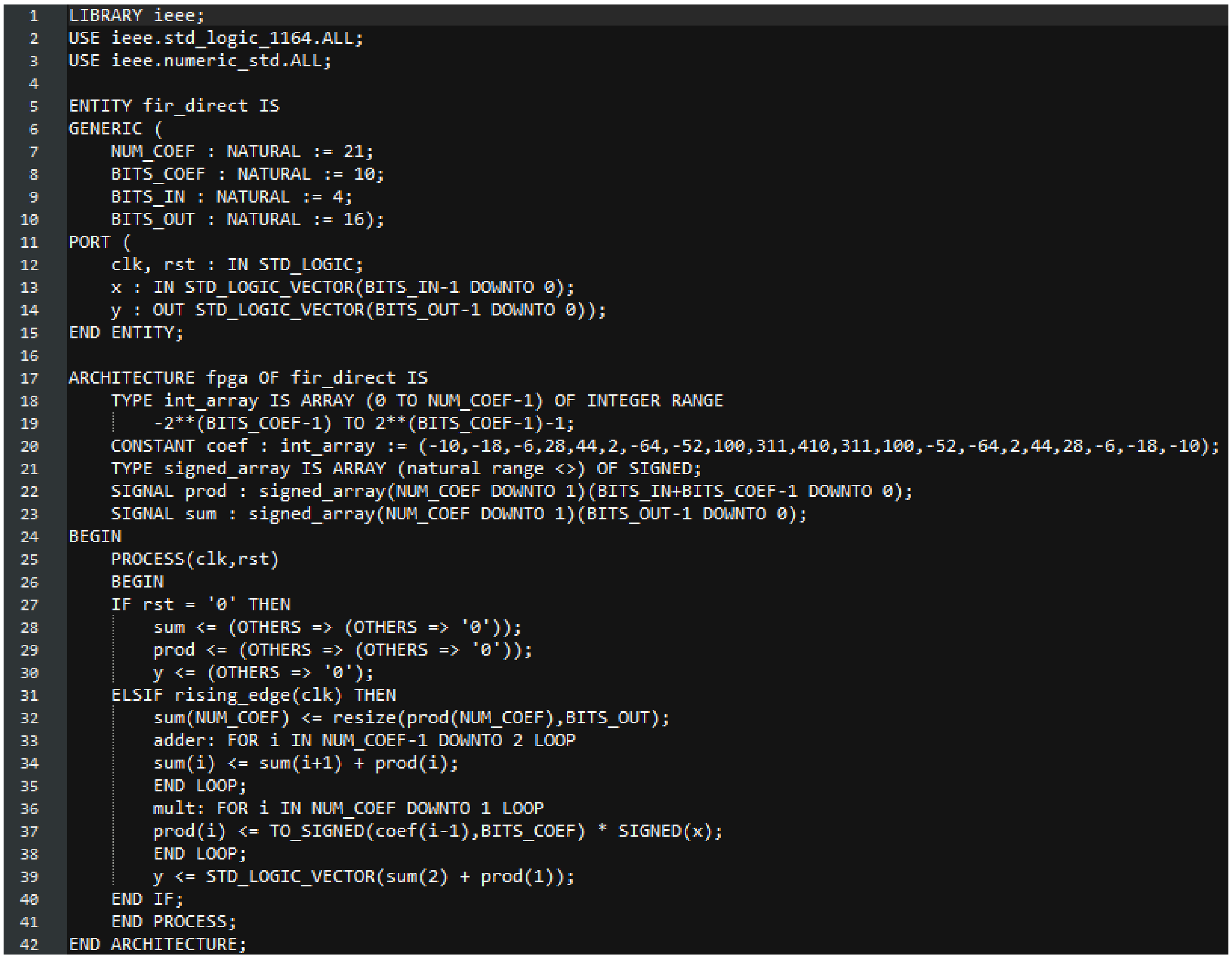


```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY fir_direct IS
GENERIC (
    NUM_COEF : NATURAL := 11;
    BITS_COEF : NATURAL := 4;
    BITS_IN : NATURAL := 4;
    BITS_OUT : NATURAL := 10);
PORT (
    clk, rst : IN STD_LOGIC;
    x : IN STD_LOGIC_VECTOR(BITS_IN-1 DOWNTO 0);
    y : OUT STD_LOGIC_VECTOR(BITS_OUT-1 DOWNTO 0));
END ENTITY;

ARCHITECTURE fpga OF fir_direct IS
    TYPE int_array IS ARRAY (0 TO NUM_COEF-1) OF INTEGER RANGE
        -2**(BITS_COEF-1) TO 2**(BITS_COEF-1)-1;
    CONSTANT coef : int_array := (-8,-5,-5,-1,1,2,2,3,5,7,7);
    TYPE signed_array IS ARRAY (natural range <>) OF SIGNED;
    SIGNAL prod : signed_array(NUM_COEF DOWNTO 1)(BITS_IN+BITS_COEF-1 DOWNTO 0);
    SIGNAL sum : signed_array(NUM_COEF DOWNTO 1)(BITS_OUT-1 DOWNTO 0);
BEGIN
    PROCESS(clk,rst)
    BEGIN
    IF rst = '0' THEN
        sum <= (OTHERS => (OTHERS => '0'));
        prod <= (OTHERS => (OTHERS => '0'));
        y <= (OTHERS => '0');
    ELSIF rising_edge(clk) THEN
        sum(NUM_COEF) <= resize(prod(NUM_COEF),BITS_OUT);
        adder: FOR i IN NUM_COEF-1 DOWNTO 2 LOOP --changed loop range to eliminate unnecessary resistor
        sum(i) <= sum(i+1) + prod(i);
        END LOOP;
        --insert mult, y here for pipeline
        mult: FOR i IN NUM_COEF DOWNTO 1 LOOP
        prod(i) <= TO_SIGNED(coef(i-1),BITS_COEF) * SIGNED(x);
        END LOOP;
        --calculate output manually
        y <= STD_LOGIC_VECTOR(sum(2) + prod(1));
    END IF;
    END PROCESS;
END ARCHITECTURE;
```

12 Appendix 5

VHDL Code for Transposed Form FIR Digital Filter (Pipelined Ver.)

- Used for Step 6 Low Pass Filter Implementation



```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY fir_direct IS
GENERIC (
    NUM_COEF : NATURAL := 21;
    BITS_COEF : NATURAL := 10;
    BITS_IN : NATURAL := 4;
    BITS_OUT : NATURAL := 16);
PORT (
    clk, rst : IN STD_LOGIC;
    x : IN STD_LOGIC_VECTOR(BITS_IN-1 DOWNTO 0);
    y : OUT STD_LOGIC_VECTOR(BITS_OUT-1 DOWNTO 0));
END ENTITY;

ARCHITECTURE fpga OF fir_direct IS
    TYPE int_array IS ARRAY (0 TO NUM_COEF-1) OF INTEGER RANGE
        -2**(BITS_COEF-1) TO 2**(BITS_COEF-1)-1;
    CONSTANT coef : int_array := (-10,-18,-6,28,44,2,-64,-52,100,311,410,311,100,-52,-64,2,44,28,-6,-18,-10);
    TYPE signed_array IS ARRAY (natural range <>) OF SIGNED;
    SIGNAL prod : signed_array(NUM_COEF DOWNTO 1)(BITS_IN+BITS_COEF-1 DOWNTO 0);
    SIGNAL sum : signed_array(NUM_COEF DOWNTO 1)(BITS_OUT-1 DOWNTO 0);
BEGIN
    PROCESS(clk,rst)
    BEGIN
    IF rst = '0' THEN
        sum <= (OTHERS => (OTHERS => '0'));
        prod <= (OTHERS => (OTHERS => '0'));
        y <= (OTHERS => '0');
    ELSIF rising_edge(clk) THEN
        sum(NUM_COEF) <= resize(prod(NUM_COEF),BITS_OUT);
        adder: FOR i IN NUM_COEF-1 DOWNTO 2 LOOP
        sum(i) <= sum(i+1) + prod(i);
        END LOOP;
        mult: FOR i IN NUM_COEF DOWNTO 1 LOOP
        prod(i) <= TO_SIGNED(coef(i-1),BITS_COEF) * SIGNED(x);
        END LOOP;
        y <= STD_LOGIC_VECTOR(sum(2) + prod(1));
    END IF;
    END PROCESS;
END ARCHITECTURE;
```

END OF DOCUMENT