Politecnico di Torino

Department of Electronics and Telecommunications

Report for the course **Integrated Systems Architecture**

Master degree in Electronics Engineering

Group: isa34

Names: Syed Akif Ali s329233, Mohammed Sarbas s328875, Hamza Sadiq s290058

Signature 1: ____

Signature 2: ____

Signature 3: ____

Contents

CHAPTER 1

Lab 1: Design and Implementation of a FIR Digital filter

1.1 Reference model development

This section looks at the design of a reference model for digital filter with cut-off frequency $f_c = 2$kHz, sampling frequency $f_s = 10$kHz. Given that p=1 for group number 34, the digital filter to be designed is an *FIR filter.*

1.1.1 Filter design and coefficient quantization using Matlab

The reference model is developed using MATLAB, with the file available on portale *myfir_design.m.* The `f_cut_off` is set at 2000, in *myfir_design.m.* The FIR filter order `N` and Number of bits n_b are calculated using the following equations. Where x=6 (characters in *Sarbas*), y=5 (characters in *Sadiq*), and p=1.

$$N = 2^p \cdot [(x\ mod\ 2) + 1] + 6 \cdot p \rightarrow N = 8$$

$$n_b = (y\ mod\ 7) + 8 \rightarrow n_b = 13$$

The `N` and `nb` variables are set in *my_fir_filter.m* as calculated above. Using the `fir1` MATLAB function the filter coefficients `b` in floating point are:
`-0.0061 -0.0136 0.0512 0.2657 0.4057 0.2657 0.0512 -0.0136 -0.0061`

For $n_b = 13$, the co-efficients quantized on 13 bits are:
As real values (`bq`): `-0.0063 -0.0137 0.0510 0.2656 0.4055 0.2656 0.0510 -0.0137 -0.0063`
As integers (`bi`): `-26 -56 209 1088 1661 1088 209 -56 -26`

The frequency response of the filter using both coefficients in shown in fig 1.1.

1.1.2 Testing the filter and fixed point implementation

First step: Matlab pseudo-fixed-point

To test the designed filter, the script *my_fir_filter.m* is used. It declares a composite input signal `x` as the average of signals `x1` frequency = 500Hz (in-band) and `x2` frequency = 3500Hz (out-band). The function `myfir_design(N, nb)` is used to acquire `bi` and `bq`. Input `x` is filtered using `filter(bq, 1, x)` function. The overlapping and subplots are shown in fig 1.3 and 1.2. In particular fig 1.2 shows that

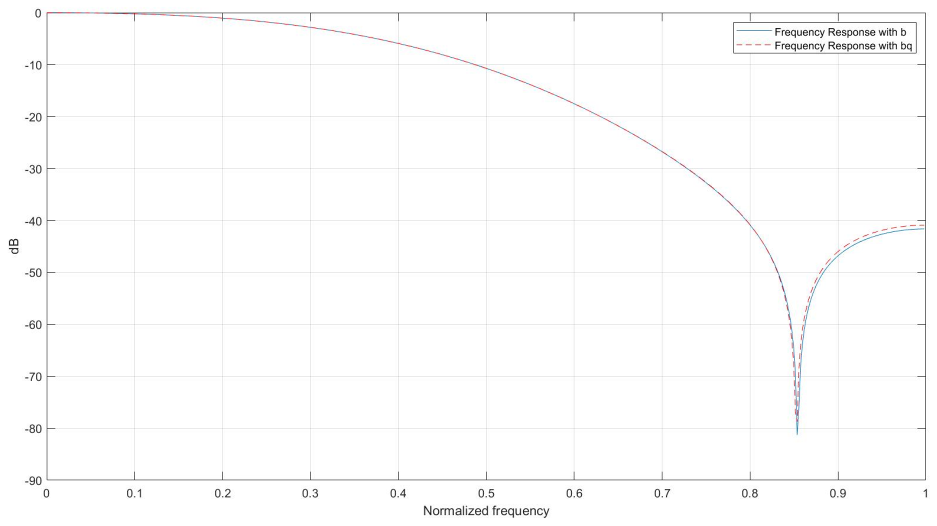


Figure 1.1: Frequency Response of Filter using b and bq

dominant frequency in output samples is about 500Hz matching that of the required in-band `x1` signal.

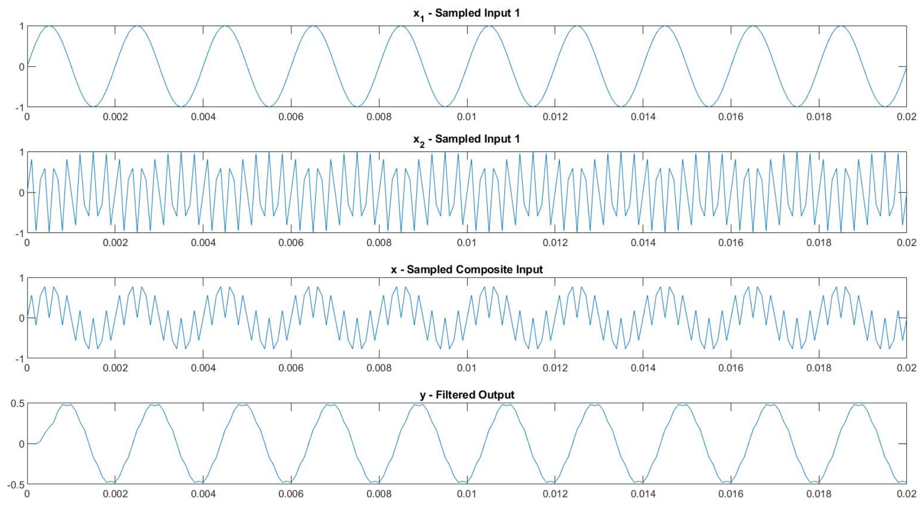


Figure 1.2: Subplots for x1, x2, x and y

The script *my_fir_filter.m* also saves the input and filtered output as quantized integer values `xq`, `yq`, using $n_b - 1$ bits for fractional part and 1 bit for whole number part. Hence a fixed point representation of Q1.12 is employed.

Second step: fixed-point C model

Using C, a pseudo hardware model for the FIR filter is developed. For this purpose the following conditions must be satisfied:

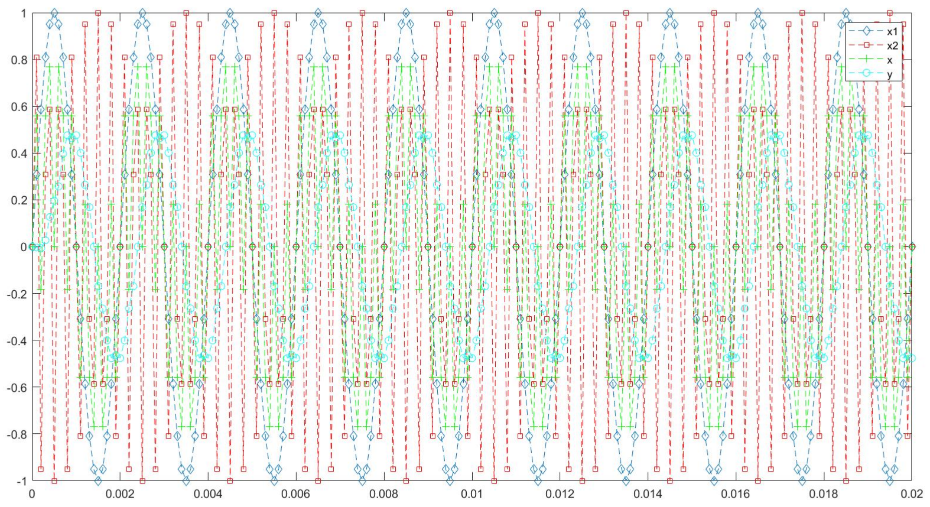


Figure 1.3: Overlapping plots for x1, x2, x and y

- Input signal `x` to the filter is limited to the range [-1, 1].
- Input maximum frequency $f_x \leq f_{nyquist}$ (5000Hz).
- 13 bits (Q1.12) is the n_b of inputs `x` and co-efficients `b`.

In *myfilterfir.c* the samples from MATLAB are read and passed to function `myfilter`. As seen in the code snippet 1.1.2 `myfilter` shifts new samples of x in the array `sx[]`. For each x shifted into `sx[]`, the accumulator loop calculates the product of ith value of `sx[i]` (input) and `bi[i]` (coefficents).

To optimize filter hardware area, the product, on $n_b + n_b = 26$bits (plus overflow) has to be shrunk on a lower bitwidth while maintaining Total Harmonic Distortion (THD) of `y` $\leq$-40 dB. If the above conditions hold true, Shift Amount `SHAMT = 16`, give THD = -46.58 dB and allows discarding 16 LSB after multiplication, through >>. Then << $(SHAMT - NB + 1)$ restores the weight. Hence `y` is presented on 14 bits.

```c
// shift and insert new sample in x shift register
for (i = NT - 1; i > 0; i--)
  sx[i] = sx[i - 1];
sx[0] = x;

/// make the convolution (Moving average part only)
y = 0;
for (i = 0; i < NT; i++){
  y += ((sx[i] * bi[i]) >> SHAMT) << (SHAMT - NB + 1);
}
return y;
```

Comparsion C and MATLAB

Comparing the outputs samples generated by C code (*results_c.txt*) with bit reduction and MATLAB (*resultsm.txt*), fig 1.4,we see that though the output plot still resembles the MATLAB model, the THD is increased from -75.72 dB (for MATLAB) to -46.58 dB (C with SHAMT = 16).

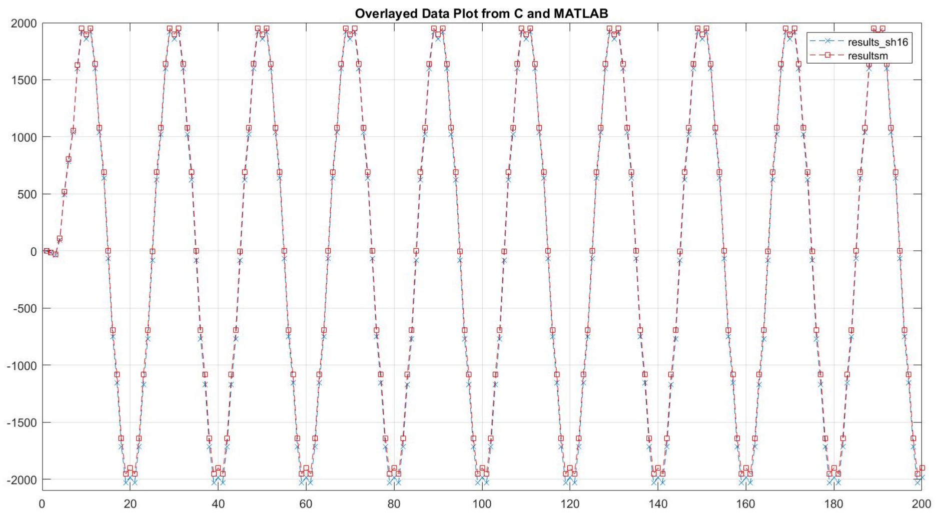


Figure 1.4: Comparing Outputs

1.2 VLSI implementation

1.2.1 Architecture

The architecture of the filter is shown in Figure 1.5. In Listing 2.1 the vhdl code describes the archi-

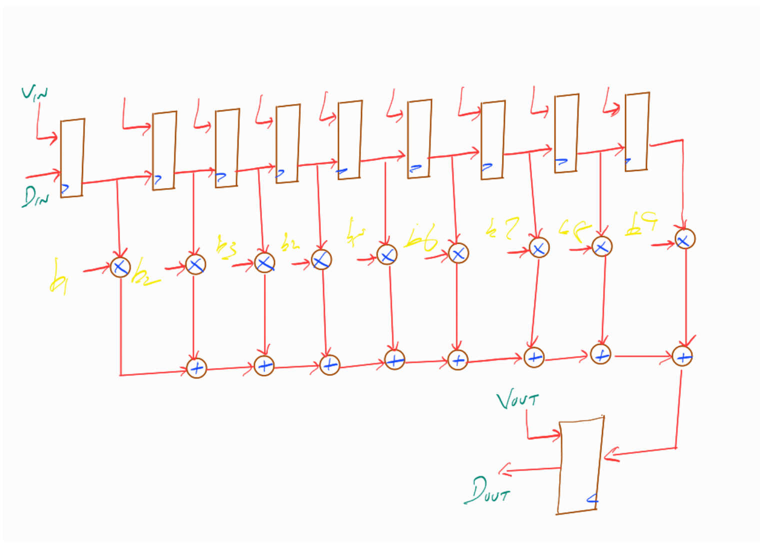


Figure 1.5: 8th order FIR filter

tecture which implements a Finite Impulse Response (FIR) filter with the following key components and functionalities:

- **Entity Definition:**
  - The FIR filter entity defines the interface for the module with generics such as:
    * `NB` - Bit width of input and filter coefficients.

    - N - Filter order.
    - SHAMT - Shift amount for scaling products.
  - Ports include clock (CLK), reset (RST_n), input valid (VIN), output valid (VOUT), data input (DIN), data output (DOUT), and filter coefficients (B0 to B8).

- **Architecture:**
  - **Registers and Signals:**
    - Uses a custom component (nb_register) to implement input, output, and shift registers for intermediate values.
    - Internal signals are declared for data shifting, product computation, and summation.
  - **Product Calculation:**
    - Computes the product of filter coefficients (B0 to B8) and shifted input samples (SHIFT_DATA_SG).
    - Products are scaled using bitwise shifts (>> SHAMT).
  - **Summation:**
    - The products are accumulated through successive summations (SUM0_SG to SUM7_SG) to generate the filter output.
  - **FSM (Finite State Machine):**
    - Manages the filter operation through three states:
      - RESET: Resets internal signals.
      - IDLE: Prepares the system for processing.
      - PROCESSING: Computes the filter output when input is valid (VIN = '1').
    - State transitions are controlled by the transition_PROC process.

- **Output Logic:**
  - The computed filter output is stored in the output register and assigned to DOUT.
  - A signal VOUT indicates when the output is valid.

1.2.2 Simulation

201 samples were generated by the reference C model. On testing the *myfir.vhd*, the *data_sink.vhd* gathers the filtered output. All 201 samples from *result_hdl* and *results_c* match. To process all the samples the simulation lasts 1329.4 ns, that is, the first VOUT is generated at 18.7 ns and the final VOUT is generated at 1348.1 ns.

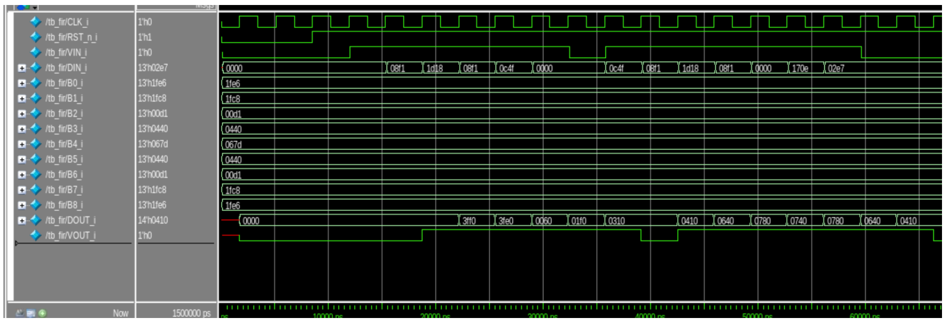


Figure 1.6: FIR Simulation Snapshot

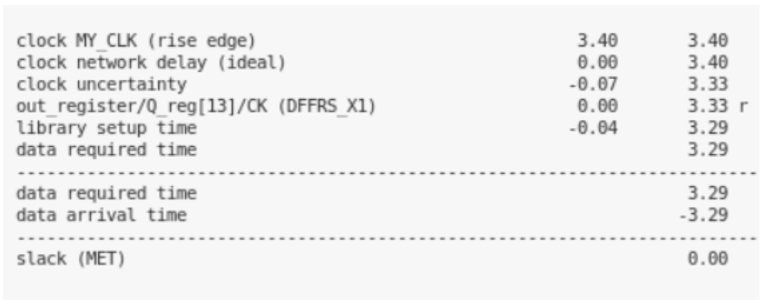
A snapshot from 0 ns to 70 ns is shown in Figure 1.6 to highlight the behaviour of the filter when VIN moves from 0 to 1 and vice versa.

1.2.3 Logic synthesis

The TCL script is used for the synthesis of FIR filter design. Below is a brief explanation of each major step:

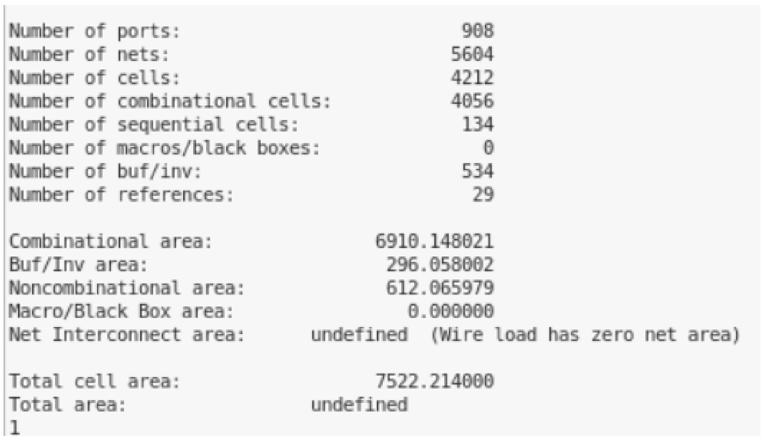
1. **Analyzing and Elaborating the Design:**
   - `analyze`: Analyzes the VHDL files (`mytypes.vhd`, `nb_register.vhd`, and `myfir.vhd`) and associates them with the `work` library.
   - `elaborate`: Elaborates the `fir_filter` entity using the architecture `arch_fir_filter`.
2. **Clock Constraints:** The design was synthesized with zero clock period and the clock period increased iteratively to **3.4 ns** with all timing constraints met and slack = 0.
   - `create_clock`: Defines a clock named `MY_CLK` with a period of 3.4 ns.
   - `set_dont_touch_network`: Prevents optimization of the clock network.
   - `set_clock_uncertainty`: Adds a clock uncertainty of 0.07 ns for `MY_CLK`.
3. **Input/Output Timing Constraints:**
   - `set_input_delay`: Specifies a maximum input delay of 0.5 ns for all inputs except `CLK`.
   - `set_output_delay`: Specifies a maximum output delay of 0.5 ns for all outputs.
4. **Output Load Setting:**
   - `set_oload`: Retrieves the output load of a buffer (`BUF_X4`) from the Nangate open cell library.
   - `set_load`: Applies this load to all output ports.
5. **Compilation:** is done without clock gating (fig 1.7, 1.8 and 1.9) and with clock gating enabled (fig 1.10, 1.11 and 1.12).
   - `compile`: Performs synthesis, mapping the design to the target technology library.
6. **Reporting:**
   - `report_timing`: Generates a timing report and saves it as `fir_timing.rpt`.
   - `report_area`: Generates an area report and saves it as `fir_area.rpt`.
   - `report_power`: Generates a power report and saves it as `fir_power.rpt`.
7. **Hierarchy Flattening:**
   - `ungroup -all -flatten`: Flattens the hierarchical structure into a single-level design.
8. **Netlist and SDF Generation:**
   - `change_names`: Renames signals and components to adhere to Verilog naming conventions.
   - `write_sdf`: Exports the timing information in SDF format (`myfir.sdf`).
   - `write`: Generates the synthesized Verilog netlist (`myfir.v`).
   - `write_sdc`: Outputs the design constraints file (`myfir.sdc`).

With clock period constrained to **3.4 ns**, the maximum frequency f_M **is 294 MHz** is obtained.



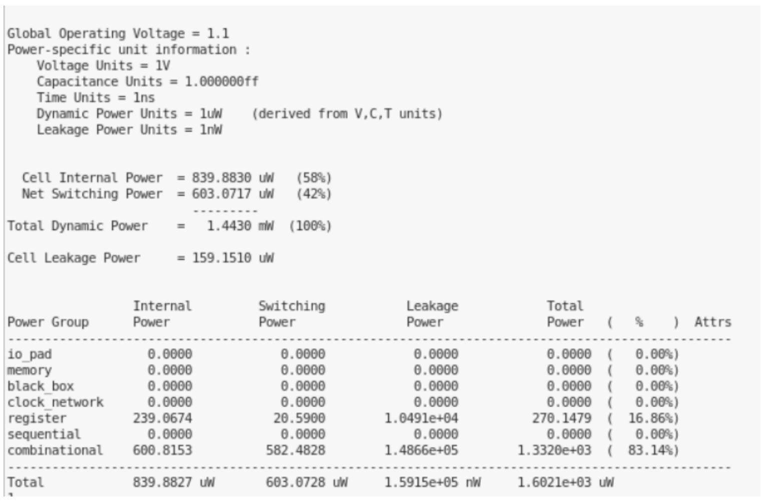
```
clock MY_CLK (rise edge)                              3.40       3.40
clock network delay (ideal)                           0.00       3.40
clock uncertainty                                    -0.07       3.33
out_register/Q_reg[13]/CK (DFFRS_X1)                  0.00       3.33 r
library setup time                                   -0.04       3.29
data required time                                               3.29
---------------------------------------------------------------------
data required time                                               3.29
data arrival time                                               -3.29
---------------------------------------------------------------------
slack (MET)                                                      0.00
```

Figure 1.7: FIR timing report without clock gating



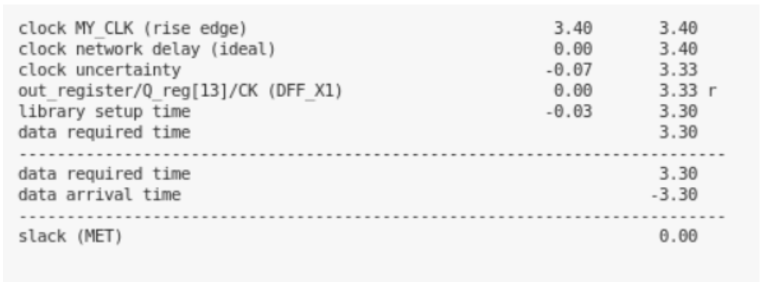
```
Number of ports:                          908
Number of nets:                          5604
Number of cells:                         4212
Number of combinational cells:           4056
Number of sequential cells:               134
Number of macros/black boxes:               0
Number of buf/inv:                        534
Number of references:                      29

Combinational area:               6910.148021
Buf/Inv area:                      296.058002
Noncombinational area:             612.065979
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:                  7522.214000
Total area:                 undefined
1
```

Figure 1.8: FIR area report without clock gating



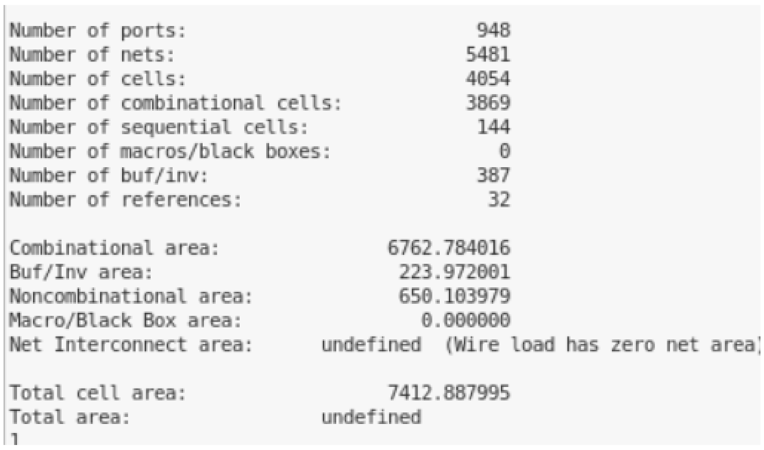
```
Global Operating Voltage = 1.1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  = 839.8830 uW   (58%)
  Net Switching Power  = 603.0717 uW   (42%)
                         ---------
Total Dynamic Power    =   1.4430 mW  (100%)

Cell Leakage Power     = 159.1510 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000            0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000            0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000            0.0000  (   0.00%)
clock_network      0.0000           0.0000            0.0000            0.0000  (   0.00%)
register         239.0674          20.5900        1.0491e+04          270.1479  (  16.86%)
sequential         0.0000           0.0000            0.0000            0.0000  (   0.00%)
combinational    600.8153         582.4828        1.4866e+05        1.3320e+03  (  83.14%)
--------------------------------------------------------------------------------------------------
Total            839.8827 uW      603.0728 uW     1.5915e+05 nW     1.6021e+03 uW
```

Figure 1.9: FIR power report without clock gating



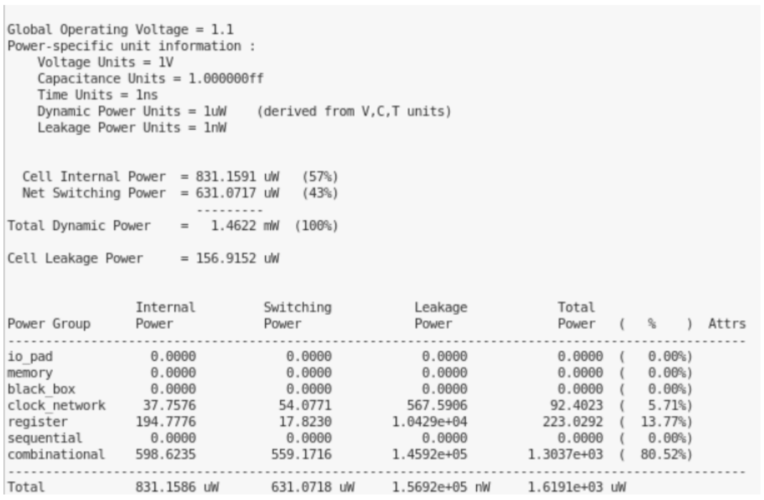
```
clock MY_CLK (rise edge)                              3.40       3.40
clock network delay (ideal)                           0.00       3.40
clock uncertainty                                    -0.07       3.33
out_register/Q_reg[13]/CK (DFF_X1)                    0.00       3.33 r
library setup time                                   -0.03       3.30
data required time                                               3.30
---------------------------------------------------------------------
data required time                                               3.30
data arrival time                                               -3.30
---------------------------------------------------------------------
slack (MET)                                                      0.00
```

Figure 1.10: FIR timing report with clock gating



```
Number of ports:                          948
Number of nets:                          5481
Number of cells:                         4054
Number of combinational cells:           3869
Number of sequential cells:               144
Number of macros/black boxes:               0
Number of buf/inv:                        387
Number of references:                      32

Combinational area:               6762.784016
Buf/Inv area:                      223.972001
Noncombinational area:             650.103979
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:                  7412.887995
Total area:                 undefined
1
```

Figure 1.11: FIR area report with clock gating



```
Global Operating Voltage = 1.1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  = 831.1591 uW   (57%)
  Net Switching Power  = 631.0717 uW   (43%)
                         ---------
Total Dynamic Power    =   1.4622 mW  (100%)

Cell Leakage Power     = 156.9152 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000            0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000            0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000            0.0000  (   0.00%)
clock_network     37.7576          54.0771          567.5906           92.4023  (   5.71%)
register         194.7776          17.8230        1.0429e+04          223.0292  (  13.77%)
sequential         0.0000           0.0000            0.0000            0.0000  (   0.00%)
combinational    598.6235         559.1716        1.4592e+05        1.3037e+03  (  80.52%)
--------------------------------------------------------------------------------------------------
Total            831.1586 uW      631.0718 uW     1.5692e+05 nW     1.6191e+03 uW
```

Figure 1.12: FIR power report with clock gating

1.2.4 Place and route

The process of place and route in Cadence Innovus involved the following steps: design import, floorplanning, power planning and routing, cell placement, clock tree synthesis, signal routing, and timing and design analysis. After importing the design, the floorplan is structured, power rings are inserted, and standard cell power routing is performed. Cells are then placed, and pre- and post-clock tree synthesis optimizations are applied. The clock tree is synthesized, and signal routing is carried out. Post-routing optimization is performed, and filler cells are added to complete the placement. Finally, parasitics are extracted, timing analysis as shown in figure (1.14, 1.15) is conducted, and design connectivity and design rules are verified.

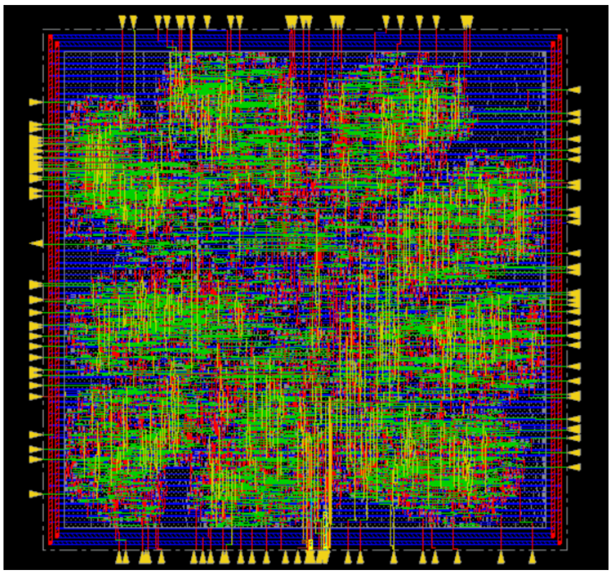
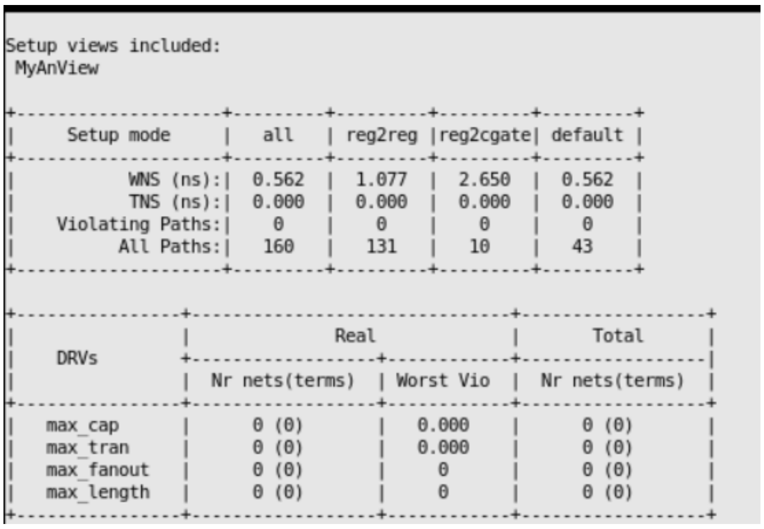
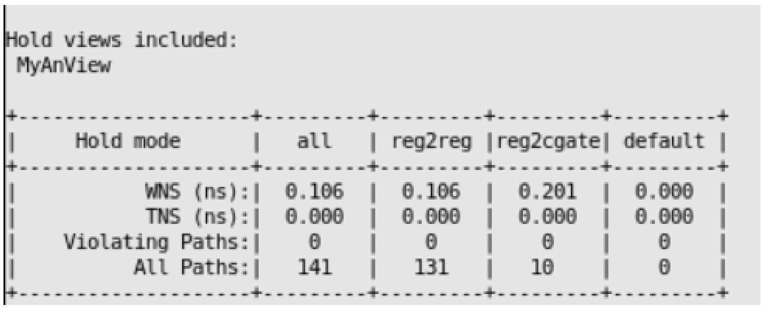


Figure 1.13: Snapshot of FIR filter



```
Setup views included:
 MyAnView

+--------------------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg |reg2cgate| default |
+--------------------+---------+---------+---------+---------+
|           WNS (ns):|  0.562  |  1.077  |  2.650  |  0.562  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |    0    |
|          All Paths:|   160   |   131   |   10    |   43    |
+--------------------+---------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +------------------+------------+------------------|
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |    0 (0)         |   0.000    |     0 (0)        |
|   max_tran     |    0 (0)         |   0.000    |     0 (0)        |
|   max_fanout   |    0 (0)         |     0      |     0 (0)        |
|   max_length   |    0 (0)         |     0      |     0 (0)        |
+----------------+------------------+------------+------------------+
```

Figure 1.14: SetUp Time Summary



```
Hold views included:
 MyAnView

+--------------------+---------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg |reg2cgate| default |
+--------------------+---------+---------+---------+---------+
|           WNS (ns):|  0.106  |  0.106  |  0.201  |  0.000  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |    0    |
|          All Paths:|   141   |   131   |   10    |    0    |
+--------------------+---------+---------+---------+---------+
```

Figure 1.15: Hold Time Summary

1.3 Advanced architecture development

1.3.1 Advanced Architecture

The architecture of the pipelined filter is shown in Figure 1.16 FIR filter is improved with the unfolding

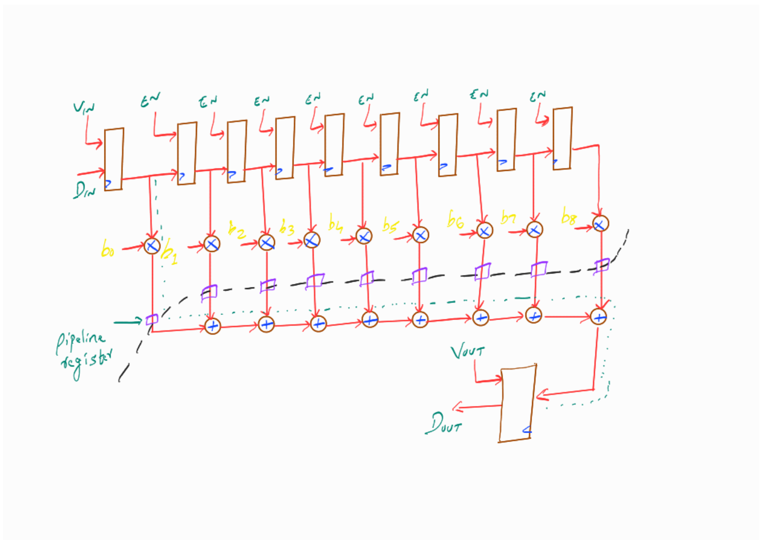


Figure 1.16: 8th order pipelined FIR filter

factor of 3 and pipelined, to improve both throughput and maximize the frequency. Added registers after each multiplication to reduce the critical path. The architecture is described in Listing 2.2.

1.3.2 Simulation

To increase the processing speed of the 201-samples through the FIR filter, a parallel processing approach is employed. By unfolding the filter three times, three samples were processed simultaneously and effectively triples the throughput. To further enhance performance, pipelining also introduced, which divides the critical path into smaller stages, allowing for concurrent processing of multiple samples. This combined approach improved the filter's overall speed and efficiency.

1.3.3 Logic synthesis

The modified TCL script is used for the synthesis of the FIR filter design with unfolded version. Below is a brief explanation of each major step:

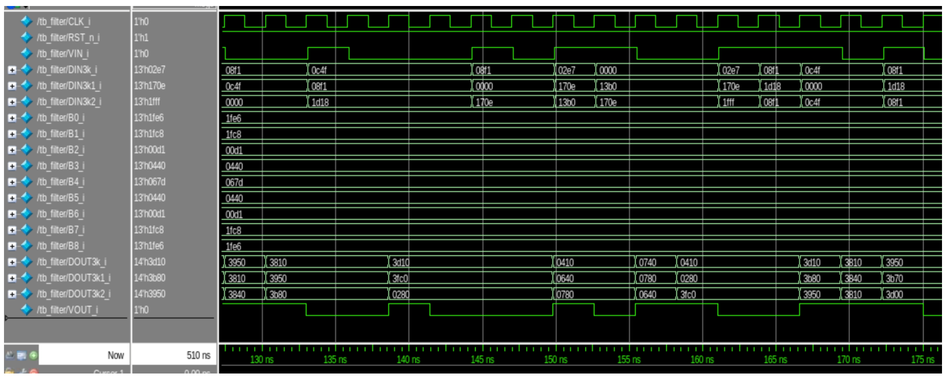
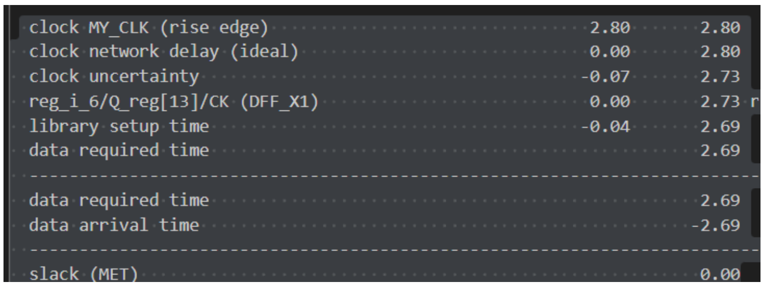


Figure 1.17: Simulation snapshot of Unfolded FIR filter

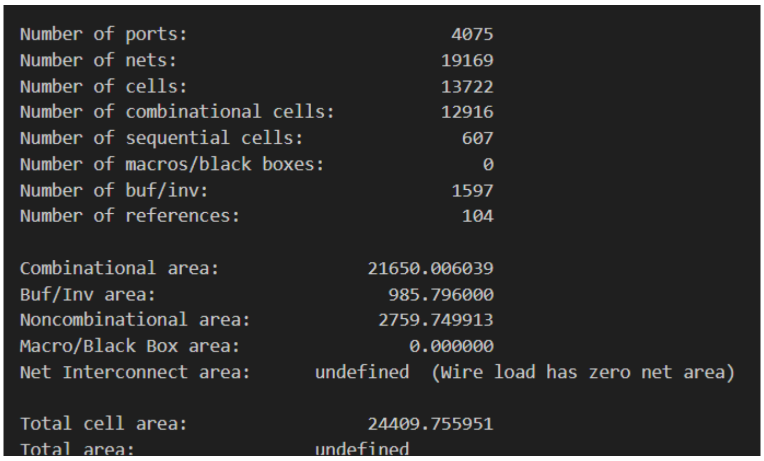
1. **Analyzing and Elaborating the FIR unfolded Design:**
   - `analyze`: Analyzes the VHDL files (`mytypes.vhd`, `nb_register.vhd`, and `myfir_unfolded.vhd`) and associates them with the `work` library.
   - `elaborate`: Elaborates the `fir_filter_advanced` entity using the architecture `arch_fir_filter_udvanced`.
2. **Clock Constraints:** The design was synthesized with zero clock period and the clock period increased iteratively to **2.8 ns** with all timing constraints met and slack = 0.
   - `create_clock`: Defines a clock named `MY_CLK` with a period of 2.8 ns.
   - `set_dont_touch_network`: Prevents optimization of the clock network.
   - `set_clock_uncertainty`: Adds a clock uncertainty of 0.07 ns for `MY_CLK`.
3. **Compilation:** is done with clock gating enabled (fig 1.18, 1.19 and 1.20).
   - `compile`: Performs synthesis, mapping the design to the target technology library.
4. **Reporting:**
   - `report_timing`: Generates a timing report and saves it as `fir_unfolded_timing.rpt`.
   - `report_area`: Generates an area report and saves it as `fir_unfolded_area.rpt`.
   - `report_power`: Generates a power report and saves it as `fir_unfolded_power.rpt`.

With clock period = 2.8 ns and the maximum frequency F_M is 357 MHz.



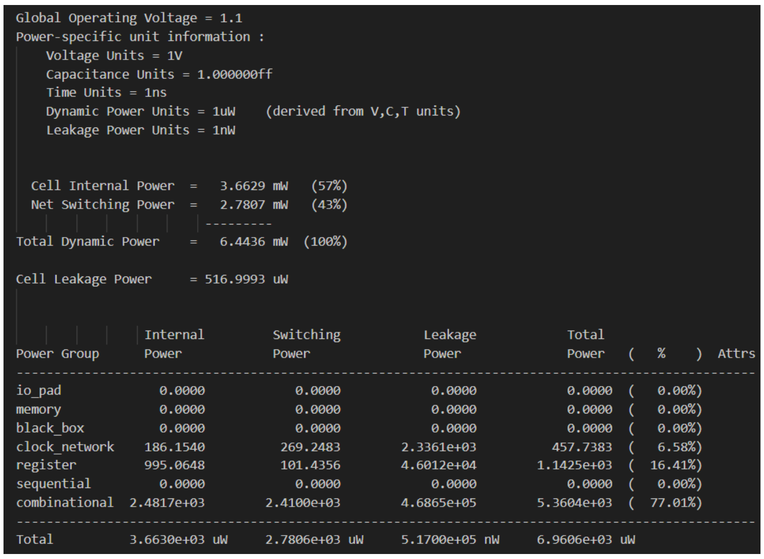
```
clock MY_CLK (rise edge)                        2.80       2.80
clock network delay (ideal)                     0.00       2.80
clock uncertainty                              -0.07       2.73
reg_i_6/Q_reg[13]/CK (DFF_X1)                   0.00       2.73 r
library setup time                             -0.04       2.69
data required time                                         2.69
-----------------------------------------------------------------
data required time                                         2.69
data arrival time                                         -2.69
-----------------------------------------------------------------
slack (MET)                                                0.00
```

Figure 1.18: FIR timing report with clock gating



```
Number of ports:                         4075
Number of nets:                         19169
Number of cells:                        13722
Number of combinational cells:          12916
Number of sequential cells:               607
Number of macros/black boxes:               0
Number of buf/inv:                       1597
Number of references:                     104

Combinational area:               21650.006039
Buf/Inv area:                       985.796000
Noncombinational area:             2759.749913
Macro/Black Box area:                 0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:                  24409.755951
Total area:                 undefined
```

Figure 1.19: FIR area report with clock gating



```
Global Operating Voltage = 1.1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  =   3.6629 mW   (57%)
  Net Switching Power  =   2.7807 mW   (43%)
                         ---------
Total Dynamic Power    =   6.4436 mW  (100%)

Cell Leakage Power     = 516.9993 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000            0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000            0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000            0.0000  (   0.00%)
clock_network    186.1540         269.2483        2.3361e+03          457.7383  (   6.58%)
register         995.0648         101.4356        4.6012e+04        1.1425e+03  (  16.41%)
sequential         0.0000           0.0000            0.0000            0.0000  (   0.00%)
combinational  2.4817e+03       2.4100e+03        4.6865e+05        5.3604e+03  (  77.01%)
--------------------------------------------------------------------------------------------------
Total          3.6630e+03 uW    2.7806e+03 uW     5.1700e+05 nW     6.9606e+03 uW
```

Figure 1.20: FIR power report with clock gating

CHAPTER 2

Appendix 1

2.1 myfir.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

use work.myTypes.all;

entity fir_filter is
    generic (
        NB : integer := 13; -- Bit Width
        N : integer := 8; -- Filter Order
        SHAMT : integer := 16); -- Shift Amount
    port (
        CLK : in  std_logic;
        RST_n : in  std_logic; -- Active Low Reset

        VIN : in  std_logic;
        VOUT : out std_logic;

        DIN : in  std_logic_vector(NB-1 downto 0);
        DOUT : out std_logic_vector(NB downto 0);

        B0 : in std_logic_vector(NB-1 downto 0);
        B1 : in std_logic_vector(NB-1 downto 0);
        B2 : in std_logic_vector(NB-1 downto 0);
        B3 : in std_logic_vector(NB-1 downto 0);
        B4 : in std_logic_vector(NB-1 downto 0);
        B5 : in std_logic_vector(NB-1 downto 0);
        B6 : in std_logic_vector(NB-1 downto 0);
        B7 : in std_logic_vector(NB-1 downto 0);
        B8 : in std_logic_vector(NB-1 downto 0));
end fir_filter;

architecture arch_fir_filter of fir_filter is

```

```vhdl
    component nb_register is
        generic (
            NB: integer := 13);
        port (
            CLK : in std_logic;
            RST_n : in std_logic;
            EN : in std_logic;
            D : in signed (NB-1 downto 0);
            Q: out signed (NB-1 downto 0));
    end component nb_register;

    -- Declare Signals
    signal DOUT_SG : signed (NB downto 0); -- Signed Output Samples

    type SHIFT_DATA_ARRAY is array (0 to N) of signed (NB-1 downto 0);
    signal SHIFT_DATA_SG : SHIFT_DATA_ARRAY;

    signal PROD0_SG, PROD1_SG, PROD2_SG, PROD3_SG, PROD4_SG, PROD5_SG,
   PROD6_SG, PROD7_SG, PROD8_SG : signed (2*NB-1 downto 0);
    signal PROD0_SHIFT_SG, PROD1_SHIFT_SG, PROD2_SHIFT_SG, PROD3_SHIFT_SG,
   PROD4_SHIFT_SG, PROD5_SHIFT_SG, PROD6_SHIFT_SG, PROD7_SHIFT_SG,
   PROD8_SHIFT_SG : signed (NB downto 0);
    signal SUM0_SG, SUM1_SG, SUM2_SG, SUM3_SG, SUM4_SG, SUM5_SG, SUM6_SG,
   SUM7_SG : signed (NB+1-1 downto 0);

    signal EN_OUT_REG_SG, EN_SHIFT_SG, RST_n_SG, EN_IN_REG_SG: std_logic;

    -- signal to hold VOUT
    signal VOUT_HOLD : std_logic := '0';

    -- FSM States
    type STATE_TYPE is (RESET, IDLE, PROCESSING);
    signal CURRENT_STATE, NEXT_STATE : STATE_TYPE;

begin

    -- Instantiate input register
    -- Acquire from DIN and send to shift_register 0
    in_register: nb_register
    generic map (NB => NB)
    port map (CLK => CLK, RST_n => RST_n_SG, EN => EN_IN_REG_SG, D => signed(
   DIN), Q => SHIFT_DATA_SG(0));

    -- Instantiate (N) shift registers.
    gEN_SHIFT_SG_registers: for i in 0 to N-1 generate
        shift_register: nb_register
            generic map (NB => NB)
            port map (CLK => CLK, RST_n => RST_n_SG, EN => EN_SHIFT_SG, D =>
   SHIFT_DATA_SG(i), Q => SHIFT_DATA_SG(i+1));
    end generate gEN_SHIFT_SG_registers;

    -- Instantiate output register
    -- Acquire data from final Adder and send to DOUT
    out_register: nb_register
    generic map (NB => NB+1) --
    port map (CLK => CLK, RST_n => RST_n_SG, EN => EN_OUT_REG_SG, D => SUM7_SG, Q => DOUT_SG);
    DOUT <= std_logic_vector(DOUT_SG);

    -- Get Products, * doesn’t require operand resizing
    PROD0_SG <= signed (B0) * SHIFT_DATA_SG(0);
    PROD1_SG <= signed (B1) * SHIFT_DATA_SG(1);
    PROD2_SG <= signed (B2) * SHIFT_DATA_SG(2);
    PROD3_SG <= signed (B3) * SHIFT_DATA_SG(3);
    PROD4_SG <= signed (B4) * SHIFT_DATA_SG(4);
    PROD5_SG <= signed (B5) * SHIFT_DATA_SG(5);
    PROD6_SG <= signed (B6) * SHIFT_DATA_SG(6);
    PROD7_SG <= signed (B7) * SHIFT_DATA_SG(7);
    PROD8_SG <= signed (B8) * SHIFT_DATA_SG(8);

    -- Shift Products
    -- ((sx[i] * bi[i]) >> SHAMT) << (SHAMT - NB + 1);
    -------------------- Shift 1, get 10b MSB from 26b keeping THD < -40dB _------------
    -------------------- Shift 2, amplification to match Matlab result ----------------
    PROD0_SHIFT_SG (NB downto SHAMT-NB+1) <= PROD0_SG (2*NB-1 downto SHAMT); -- bits 14...4 connected with 25..16
    PROD0_SHIFT_SG (SHAMT-NB downto 0) <= (others => ’0’); -- bits 3...0 connected with 0
    PROD1_SHIFT_SG (NB downto SHAMT-NB+1) <= PROD1_SG (2*NB-1 downto SHAMT); -- bits 14...4 connected with 25..16
    PROD1_SHIFT_SG (SHAMT-NB downto 0) <= (others => ’0’); -- bits 3...0 connected with 0
    PROD2_SHIFT_SG (NB downto SHAMT-NB+1) <= PROD2_SG (2*NB-1 downto SHAMT); -- bits 14...4 connected with 25..16
    PROD2_SHIFT_SG (SHAMT-NB downto 0) <= (others => ’0’); -- bits 3...0 connected with 0
    PROD3_SHIFT_SG (NB downto SHAMT-NB+1) <= PROD3_SG (2*NB-1 downto SHAMT); -- bits 14...4 connected with 25..16
    PROD3_SHIFT_SG (SHAMT-NB downto 0) <= (others => ’0’); -- bits 3...0 connected with 0
    PROD4_SHIFT_SG (NB downto SHAMT-NB+1) <= PROD4_SG (2*NB-1 downto SHAMT); -- bits 14...4 connected with 25..16
    PROD4_SHIFT_SG (SHAMT-NB downto 0) <= (others => ’0’); -- bits 3...0 connected with 0
    PROD5_SHIFT_SG (NB downto SHAMT-NB+1) <= PROD5_SG (2*NB-1 downto SHAMT); -- bits 14...4 connected with 25..16
    PROD5_SHIFT_SG (SHAMT-NB downto 0) <= (others => ’0’); -- bits 3...0 connected with 0
    PROD6_SHIFT_SG (NB downto SHAMT-NB+1) <= PROD6_SG (2*NB-1 downto SHAMT); -- bits 14...4 connected with 25..16
    PROD6_SHIFT_SG (SHAMT-NB downto 0) <= (others => ’0’); -- bits 3...0 connected with 0
    PROD7_SHIFT_SG (NB downto SHAMT-NB+1) <= PROD7_SG (2*NB-1 downto SHAMT); -- bits 14...4 connected with 25..16
    PROD7_SHIFT_SG (SHAMT-NB downto 0) <= (others => ’0’); -- bits 3...0 connected with 0
    PROD8_SHIFT_SG (NB downto SHAMT-NB+1) <= PROD8_SG (2*NB-1 downto SHAMT);
    -- bits 14...4 connected with 25..16
    PROD8_SHIFT_SG (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0
    connected with 0

    -- Get Sums, resize products to fit NB+1 bits
    SUM0_SG <= PROD0_SHIFT_SG + PROD1_SHIFT_SG;
    SUM1_SG <= SUM0_SG + PROD2_SHIFT_SG;
    SUM2_SG <= SUM1_SG + PROD3_SHIFT_SG;
    SUM3_SG <= SUM2_SG + PROD4_SHIFT_SG;
    SUM4_SG <= SUM3_SG + PROD5_SHIFT_SG;
    SUM5_SG <= SUM4_SG + PROD6_SHIFT_SG;
    SUM6_SG <= SUM5_SG + PROD7_SHIFT_SG;
    SUM7_SG <= SUM6_SG + PROD8_SHIFT_SG;

    -- Hold VOUT Logic
    VOUT_hold_PROC: process(CLK)
    begin
        if rising_edge(CLK) then
            VOUT <= VOUT_HOLD;
        end if;
    end process;

    -- FSM State Transition Process
    transition_PROC: process(CLK, RST_n)
    begin
        if (RST_n = '0') then
            CURRENT_STATE <= RESET; -- Asynchronous RST active (low), CS =
    RESET
        elsif rising_edge(CLK) then
            CURRENT_STATE <= NEXT_STATE; -- When RST inactive CS = NS on
    every CLK rise edge
        end if;
    end process;

    -- FSM Next State Logic
    next_state_PROC: process(CURRENT_STATE, VIN)
    begin
        case CURRENT_STATE is
            when RESET =>
                if RST_n = '1' then
                    NEXT_STATE <= PROCESSING;                    --
    Transition to IDLE after reset
                else
                    NEXT_STATE <= RESET;
                end if;
            when IDLE =>
                if VIN = '1' then
                    NEXT_STATE <= PROCESSING;              -- Move to
    processing when valid input is detected
                else
                    NEXT_STATE <= IDLE;
                end if;
            when PROCESSING =>
```

```
                if VIN = '1' then
                    NEXT_STATE <= PROCESSING;                  -- Continue
    processing if VIN is still valid
                else
                    NEXT_STATE <= IDLE;                        -- Otherwise,
    return to IDLE
                end if;
        end case;
    end process;

    output_PROC : process(CURRENT_STATE)
    begin
        EN_IN_REG_SG <= '0';
        EN_OUT_REG_SG <= '0';
        EN_SHIFT_SG <= '0';
        RST_n_SG <= '1';
        case CURRENT_STATE is
            when RESET =>
                RST_n_SG <= '0';
                VOUT_HOLD <= '0';
            when IDLE =>
                EN_IN_REG_SG <= '1';
                EN_SHIFT_SG <= '0';
                EN_OUT_REG_SG <= '0';
                VOUT_HOLD <= '0';
            when PROCESSING =>
                EN_IN_REG_SG <= '1';
                EN_SHIFT_SG <= '1';
                EN_OUT_REG_SG <= '1';
                VOUT_HOLD <= '1';
        end case;
    end process;

end architecture arch_fir_filter;
```

2.2 myfir_unfolded.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

use work.myTypes.all;

entity fir_filter_advanced is
    generic (
        NB : integer := 13;   -- Bit Width
        N : integer := 8;     -- Filter Order
        SHAMT : integer := 16  -- Shift Amount
    );
    port (
```

```vhdl
        CLK : in std_logic;
        RST_n : in std_logic; -- Active Low Reset
        VIN : in std_logic;   -- Valid Input Signal
        VOUT : out std_logic; -- Valid Output Signal
        DIN3k : in  std_logic_vector(NB-1 downto 0);
        DIN3k1 : in  std_logic_vector(NB-1 downto 0);
        DIN3k2 : in  std_logic_vector(NB-1 downto 0);

        DOUT3k : out  std_logic_vector(NB downto 0);
        DOUT3k1 : out  std_logic_vector(NB downto 0);
        DOUT3k2 : out std_logic_vector(NB downto 0);

        B0 : in std_logic_vector(NB-1 downto 0);
        B1 : in std_logic_vector(NB-1 downto 0);
        B2 : in std_logic_vector(NB-1 downto 0);
        B3 : in std_logic_vector(NB-1 downto 0);
        B4 : in std_logic_vector(NB-1 downto 0);
        B5 : in std_logic_vector(NB-1 downto 0);
        B6 : in std_logic_vector(NB-1 downto 0);
        B7 : in std_logic_vector(NB-1 downto 0);
        B8 : in std_logic_vector(NB-1 downto 0)
    );
end fir_filter_advanced;

architecture arch_fir_filter_advanced of fir_filter_advanced is

    -- Component Declaration for nb_register
    component nb_register is
        generic (
            NB: integer := 13);
        port (
            CLK : in std_logic;
            RST_n : in std_logic;
            EN : in std_logic;
            D : in signed (NB-1 downto 0);
            Q: out signed (NB-1 downto 0));
    end component;



    ------ Declare Signals
    signal DOUT3k_SG, DOUT3k1_SG, DOUT3k2_SG : signed (NB downto 0);   ----
    signed output samples

    -- Signal Declarations for Input Registers and Shift Registers
    type SHIFT_DATA_ARRAY1 is array (0 to 2) of signed (NB-1 downto 0);
    signal SHIFT_DATA_SG1 : SHIFT_DATA_ARRAY1;

    type SHIFT_DATA_ARRAY2 is array (0 to 3) of signed (NB-1 downto 0);
    signal SHIFT_DATA_SG2, SHIFT_DATA_SG3 : SHIFT_DATA_ARRAY2;

    -- Signals for Product Storage
    type PROD_ARRAY is array (0 to 8) of signed(2*NB-1 downto 0);
    signal PROD1_SG, PROD2_SG, PROD3_SG : PROD_ARRAY;

    type SHIFTED_PROD is array (0 to 8) of signed (NB downto 0);
    signal PROD1_SHIFT_SG , PROD2_SHIFT_SG , PROD3_SHIFT_SG : SHIFTED_PROD;

    -- Pipeline Registers for storing shifted products
    signal PIPELINE_PROD1 , PIPELINE_PROD2 , PIPELINE_PROD3 : SHIFTED_PROD;

    -- Signals for Accumulation

    type SUM_SG is array (0 to 7) of signed (NB+1-1 downto 0);
    signal SUM1_SG , SUM2_SG , SUM3_SG : SHIFTED_PROD;

  --  signal SUM1 , SUM2 , SUM3 : signed(NB+1-1 downto 0);

    -- FSM State Signals
    type STATE_TYPE is (RESET , IDLE , PROCESSING);
    signal CURRENT_STATE , NEXT_STATE : STATE_TYPE;

    -- Control Signals
    signal EN_OUT_REG_SG , EN_SHIFT_SG , RST_n_SG , EN_IN_REG_SG ,
   EN_PIPELINE_REG : std_logic;

     -- signal to hold VOUT
     signal VOUT_HOLD : std_logic := '0';



begin

    --instantiate Input Registers using nb_register
    in_reg1: nb_register
    generic map (NB => NB)
        port map (
            CLK => CLK ,
            RST_n => RST_n_SG ,
            EN => EN_IN_REG_SG ,
            D => signed(DIN3k) ,
            Q => SHIFT_DATA_SG1(0)
        );

    in_reg2: nb_register
    generic map (NB => NB)
        port map (
            CLK => CLK ,
            RST_n => RST_n_SG ,
            EN => EN_IN_REG_SG ,
            D => signed(DIN3k1) ,
            Q => SHIFT_DATA_SG2(0)
        );

    in_reg3: nb_register
    generic map (NB => NB)
        port map (
            CLK => CLK ,
            RST_n => RST_n_SG,
            EN => EN_IN_REG_SG,
            D => signed(DIN3k2),
            Q => SHIFT_DATA_SG3(0)
        );

    -- registers for xn[3k]
shift_reg_3k: for i in 0 to 1 generate
  reg_i: nb_register generic map (NB => NB)
        port map (
            CLK => CLK,
            RST_n => RST_n_SG,
            EN => EN_SHIFT_SG,
            D => SHIFT_DATA_SG1(i),
            Q => SHIFT_DATA_SG1(i+1)
        );
end generate shift_reg_3k;

-- registers for xn[3k+1]
shift_reg_3k_plus_1: for i in 0 to 2 generate
  reg_i: nb_register generic map (NB => NB)
                 port map (
                     CLK => CLK,
                     RST_n => RST_n_SG,
                     EN => EN_SHIFT_SG,
                     D => SHIFT_DATA_SG2(i),
                     Q => SHIFT_DATA_SG2(i+1)
                 );
end generate shift_reg_3k_plus_1;

-- registers for xn[3k+2]
shift_reg_3k_plus_2: for i in 0 to 2 generate
  reg_i: nb_register generic map (NB => NB)
        port map (
            CLK => CLK,
            RST_n => RST_n_SG,
            EN => EN_SHIFT_SG,
            D => SHIFT_DATA_SG3(i),
            Q => SHIFT_DATA_SG3(i+1)
        );
end generate shift_reg_3k_plus_2;


    -- Instantiate Output Registers using nb_register
    out_reg1: nb_register
    generic map (NB => NB+1)
    port map (
        CLK => CLK,
        RST_n => RST_n_SG,
        EN => EN_OUT_REG_SG,
        D => SUM1_SG(7),
        Q => DOUT3k_SG
    );
    DOUT3k <= std_logic_vector(DOUT3k_SG);
```

```vhdl

    out_reg2: nb_register
        generic map (NB => NB+1)
        port map (
            CLK => CLK,
            RST_n => RST_n_SG,
            EN => EN_OUT_REG_SG,
            D => SUM2_SG(7),
            Q => DOUT3k1_SG
        );
    DOUT3k1 <= std_logic_vector(DOUT3k1_SG);

    out_reg3: nb_register
        generic map (NB => NB+1)
        port map (
            CLK => CLK,
            RST_n => RST_n_SG,
            EN => EN_OUT_REG_SG,
            D => SUM3_SG(7),
            Q => DOUT3k2_SG
        );
    DOUT3k2 <= std_logic_vector(DOUT3k2_SG);



                    -- Get Products for Data Stream 1
                    PROD1_SG(0) <= signed(B0) * SHIFT_DATA_SG1(0);
                    PROD1_SG(1) <= signed(B1) * SHIFT_DATA_SG3(1);
                    PROD1_SG(2) <= signed(B2) * SHIFT_DATA_SG2(1);
                    PROD1_SG(3) <= signed(B3) * SHIFT_DATA_SG1(1);
                    PROD1_SG(4) <= signed(B4) * SHIFT_DATA_SG3(2);
                    PROD1_SG(5) <= signed(B5) * SHIFT_DATA_SG2(2);
                    PROD1_SG(6) <= signed(B6) * SHIFT_DATA_SG1(2);
                    PROD1_SG(7) <= signed(B7) * SHIFT_DATA_SG3(3);
                    PROD1_SG(8) <= signed(B8) * SHIFT_DATA_SG2(3);

                    -- Get Products for Data Stream 2
                    PROD2_SG(0) <= signed(B0) * SHIFT_DATA_SG2(0);
                    PROD2_SG(1) <= signed(B1) * SHIFT_DATA_SG1(0);
                    PROD2_SG(2) <= signed(B2) * SHIFT_DATA_SG3(1);
                    PROD2_SG(3) <= signed(B3) * SHIFT_DATA_SG2(1);
                    PROD2_SG(4) <= signed(B4) * SHIFT_DATA_SG1(1);
                    PROD2_SG(5) <= signed(B5) * SHIFT_DATA_SG3(2);
                    PROD2_SG(6) <= signed(B6) * SHIFT_DATA_SG2(2);
                    PROD2_SG(7) <= signed(B7) * SHIFT_DATA_SG1(2);
                    PROD2_SG(8) <= signed(B8) * SHIFT_DATA_SG3(3);

                    -- Get Products for Data Stream 3
                    PROD3_SG(0) <= signed(B0) * SHIFT_DATA_SG3(0);
                    PROD3_SG(1) <= signed(B1) * SHIFT_DATA_SG2(0);
                    PROD3_SG(2) <= signed(B2) * SHIFT_DATA_SG1(0);
                    PROD3_SG(3) <= signed(B3) * SHIFT_DATA_SG3(1);
                    PROD3_SG(4) <= signed(B4) * SHIFT_DATA_SG2(1);
                    PROD3_SG(5) <= signed(B5) * SHIFT_DATA_SG1(1);
```

```
                PROD3_SG(6) <= signed(B6) * SHIFT_DATA_SG3(2);
                PROD3_SG(7) <= signed(B7) * SHIFT_DATA_SG2(2);
                PROD3_SG(8) <= signed(B8) * SHIFT_DATA_SG1(2);


                -- Shift Products for Data Stream 1

                PROD1_SHIFT_SG(0) (NB downto SHAMT-NB+1) <= PROD1_SG(0)(2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                PROD1_SHIFT_SG(0) (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0 connected with 0
                PROD1_SHIFT_SG(1) (NB downto SHAMT-NB+1) <= PROD1_SG(1)(2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                PROD1_SHIFT_SG(1) (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0 connected with 0
                PROD1_SHIFT_SG(2) (NB downto SHAMT-NB+1) <= PROD1_SG(2)(2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                PROD1_SHIFT_SG(2) (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0 connected with 0
                PROD1_SHIFT_SG(3) (NB downto SHAMT-NB+1) <= PROD1_SG(3)(2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                PROD1_SHIFT_SG(3) (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0 connected with 0
                PROD1_SHIFT_SG(4) (NB downto SHAMT-NB+1) <= PROD1_SG(4)(2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                PROD1_SHIFT_SG(4) (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0 connected with 0
                PROD1_SHIFT_SG(5) (NB downto SHAMT-NB+1) <= PROD1_SG(5)(2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                PROD1_SHIFT_SG(5) (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0 connected with 0
                PROD1_SHIFT_SG(6) (NB downto SHAMT-NB+1) <= PROD1_SG(6)(2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                PROD1_SHIFT_SG(6) (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0 connected with 0
                PROD1_SHIFT_SG(7) (NB downto SHAMT-NB+1) <= PROD1_SG(7)(2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                PROD1_SHIFT_SG(7) (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0 connected with 0
                PROD1_SHIFT_SG(8) (NB downto SHAMT-NB+1) <= PROD1_SG(8)(2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                PROD1_SHIFT_SG(8) (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0 connected with 0


                -- Shift Products for Data Stream 2
               -- process(PROD2_SG)
               PROD2_SHIFT_SG(0) (NB downto SHAMT-NB+1) <= PROD2_SG(0)(2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
               PROD2_SHIFT_SG(0) (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0 connected with 0
               PROD2_SHIFT_SG(1) (NB downto SHAMT-NB+1) <= PROD2_SG(1)(2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
               PROD2_SHIFT_SG(1) (SHAMT-NB downto 0) <= (others => '0');
```

```
    -- bits 3...0 connected with 0
                    PROD2_SHIFT_SG(2) (NB downto SHAMT-NB+1) <= PROD2_SG(2)
    (2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                    PROD2_SHIFT_SG(2) (SHAMT-NB downto 0) <= (others => '0');
    -- bits 3...0 connected with 0
                    PROD2_SHIFT_SG(3) (NB downto SHAMT-NB+1) <= PROD2_SG(3)
    (2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                    PROD2_SHIFT_SG(3) (SHAMT-NB downto 0) <= (others => '0');
    -- bits 3...0 connected with 0
                    PROD2_SHIFT_SG(4) (NB downto SHAMT-NB+1) <= PROD2_SG(4)
    (2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                    PROD2_SHIFT_SG(4) (SHAMT-NB downto 0) <= (others => '0');
    -- bits 3...0 connected with 0
                    PROD2_SHIFT_SG(5) (NB downto SHAMT-NB+1) <= PROD2_SG(5)
    (2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                    PROD2_SHIFT_SG(5) (SHAMT-NB downto 0) <= (others => '0');
    -- bits 3...0 connected with 0
                    PROD2_SHIFT_SG(6) (NB downto SHAMT-NB+1) <= PROD2_SG(6)
    (2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                    PROD2_SHIFT_SG(6) (SHAMT-NB downto 0) <= (others => '0');
    -- bits 3...0 connected with 0
                    PROD2_SHIFT_SG(7) (NB downto SHAMT-NB+1) <= PROD2_SG(7)
    (2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                    PROD2_SHIFT_SG(7) (SHAMT-NB downto 0) <= (others => '0');
    -- bits 3...0 connected with 0
                    PROD2_SHIFT_SG(8) (NB downto SHAMT-NB+1) <= PROD2_SG(8)
    (2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                    PROD2_SHIFT_SG(8) (SHAMT-NB downto 0) <= (others => '0');
    -- bits 3...0 connected with 0

                     -- Shift Products for Data Stream 3
                     --process(PROD3_SG)
                     PROD3_SHIFT_SG(0) (NB downto SHAMT-NB+1) <= PROD3_SG(0)
    (2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                     PROD3_SHIFT_SG(0) (SHAMT-NB downto 0) <= (others => '0');
     -- bits 3...0 connected with 0
                     PROD3_SHIFT_SG(1) (NB downto SHAMT-NB+1) <= PROD3_SG(1)
    (2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                     PROD3_SHIFT_SG(1) (SHAMT-NB downto 0) <= (others => '0');
     -- bits 3...0 connected with 0
                     PROD3_SHIFT_SG(2) (NB downto SHAMT-NB+1) <= PROD3_SG(2)
    (2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                     PROD3_SHIFT_SG(2) (SHAMT-NB downto 0) <= (others => '0');
     -- bits 3...0 connected with 0
                     PROD3_SHIFT_SG(3) (NB downto SHAMT-NB+1) <= PROD3_SG(3)
    (2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                     PROD3_SHIFT_SG(3) (SHAMT-NB downto 0) <= (others => '0');
     -- bits 3...0 connected with 0
                     PROD3_SHIFT_SG(4) (NB downto SHAMT-NB+1) <= PROD3_SG(4)
    (2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                     PROD3_SHIFT_SG(4) (SHAMT-NB downto 0) <= (others => '0');
     -- bits 3...0 connected with 0
                     PROD3_SHIFT_SG(5) (NB downto SHAMT-NB+1) <= PROD3_SG(5)
    (2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
```

```
                    PROD3_SHIFT_SG(5) (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0 connected with 0
                    PROD3_SHIFT_SG(6) (NB downto SHAMT-NB+1) <= PROD3_SG(6)(2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                    PROD3_SHIFT_SG(6) (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0 connected with 0
                    PROD3_SHIFT_SG(7) (NB downto SHAMT-NB+1) <= PROD3_SG(7)(2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                    PROD3_SHIFT_SG(7) (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0 connected with 0
                    PROD3_SHIFT_SG(8) (NB downto SHAMT-NB+1) <= PROD3_SG(8)(2*NB-1 downto SHAMT); -- bits 13...4 connected with 25..16
                    PROD3_SHIFT_SG(8) (SHAMT-NB downto 0) <= (others => '0'); -- bits 3...0 connected with 0


    -- registers at the output of multipliers


    multipliers_out_regs1: for i in 0 to 8 generate
        reg_i: nb_register
            generic map (NB => NB+1)
            port map (
                CLK => CLK,
                RST_n => RST_n_SG,
                EN => EN_PIPELINE_REG,
                D => PROD1_SHIFT_SG(i),
                Q => PIPELINE_PROD1(i)
            );
    end generate multipliers_out_regs1;

    -- registers at the output of multipliers

    multipliers_out_regs2: for i in 0 to 8 generate
        reg_i: nb_register
            generic map (NB => NB+1)
            port map (
                CLK => CLK,
                RST_n => RST_n_SG,
                EN => EN_PIPELINE_REG,
                D => PROD2_SHIFT_SG(i),
                Q => PIPELINE_PROD2(i)
            );
    end generate multipliers_out_regs2;

    -- registers at the output of multipliers

    multipliers_out_regs3: for i in 0 to 8 generate
        reg_i: nb_register
            generic map (NB => NB+1)
            port map (
                CLK => CLK,
                RST_n => RST_n_SG,
                EN => EN_PIPELINE_REG,
                D => PROD3_SHIFT_SG(i),
                Q => PIPELINE_PROD3(i)
            );
    end generate multipliers_out_regs3;


 -- Accumulation for Data Stream 1
    SUM1_SG(0) <= PIPELINE_PROD1(0) + PIPELINE_PROD1(1);
    SUM1_SG(1) <= SUM1_SG(0) + PIPELINE_PROD1(2);
    SUM1_SG(2) <= SUM1_SG(1) + PIPELINE_PROD1(3);
    SUM1_SG(3) <= SUM1_SG(2) + PIPELINE_PROD1(4);
    SUM1_SG(4) <= SUM1_SG(3) + PIPELINE_PROD1(5);
    SUM1_SG(5) <= SUM1_SG(4) + PIPELINE_PROD1(6);
    SUM1_SG(6) <= SUM1_SG(5) + PIPELINE_PROD1(7);
    SUM1_SG(7) <= SUM1_SG(6) + PIPELINE_PROD1(8);

        -- Accumulation for Data Stream 2
        SUM2_SG(0) <= PIPELINE_PROD2(0) + PIPELINE_PROD2(1);
        SUM2_SG(1) <= SUM2_SG(0) + PIPELINE_PROD2(2);
        SUM2_SG(2) <= SUM2_SG(1) + PIPELINE_PROD2(3);
        SUM2_SG(3) <= SUM2_SG(2) + PIPELINE_PROD2(4);
        SUM2_SG(4) <= SUM2_SG(3) + PIPELINE_PROD2(5);
        SUM2_SG(5) <= SUM2_SG(4) + PIPELINE_PROD2(6);
        SUM2_SG(6) <= SUM2_SG(5) + PIPELINE_PROD2(7);
        SUM2_SG(7) <= SUM2_SG(6) + PIPELINE_PROD2(8);

        -- Accumulation for Data Stream 3
    SUM3_SG(0) <= PIPELINE_PROD3(0) + PIPELINE_PROD3(1);
    SUM3_SG(1) <= SUM3_SG(0) + PIPELINE_PROD3(2);
    SUM3_SG(2) <= SUM3_SG(1) + PIPELINE_PROD3(3);
    SUM3_SG(3) <= SUM3_SG(2) + PIPELINE_PROD3(4);
    SUM3_SG(4) <= SUM3_SG(3) + PIPELINE_PROD3(5);
    SUM3_SG(5) <= SUM3_SG(4) + PIPELINE_PROD3(6);
    SUM3_SG(6) <= SUM3_SG(5) + PIPELINE_PROD3(7);
    SUM3_SG(7) <= SUM3_SG(6) + PIPELINE_PROD3(8);

 --Hold VOUT Logic
VOUT_hold_PROC: process(CLK)
begin
    if rising_edge(CLK) then
        VOUT <= VOUT_HOLD;
    end if;
end process;



-- FSM State Transition Process
transition_PROC: process(CLK, RST_n)
begin
if (RST_n = '0') then
    CURRENT_STATE <= RESET;  -- Asynchronous reset (active low), state =
    RESET
elsif rising_edge(CLK) then
    CURRENT_STATE <= NEXT_STATE;  -- When RST inactive, next state is
    assigned on each rising edge of CLK
end if;
end process;

-- FSM Next State Logic
next_state_PROC: process(CURRENT_STATE, VIN)
begin
case CURRENT_STATE is
    when RESET =>
        if RST_n = '1' then
            NEXT_STATE <= IDLE;  -- Transition to IDLE after reset
        else
            NEXT_STATE <= RESET;  -- Stay in RESET until RST_n is high
        end if;

    when IDLE =>
        if VIN = '1' then
            NEXT_STATE <= PROCESSING;  -- Transition to PROCESSING when valid
     input is detected
        else
            NEXT_STATE <= IDLE;  -- Stay in IDLE if VIN is not valid
        end if;

    when PROCESSING =>
        if VIN = '1' then
            NEXT_STATE <= PROCESSING;  -- Continue processing if VIN is valid
        else
            NEXT_STATE <= IDLE;  -- If VIN is no longer valid, return to IDLE
        end if;

    end case;
end process;

-- FSM Output Logic (Control Signals)
output_PROC: process(CURRENT_STATE)
begin
-- Default values for all output signals

    EN_IN_REG_SG <= '0';
    EN_SHIFT_SG <= '0';
    EN_PIPELINE_REG <= '0';
    EN_OUT_REG_SG <= '0';
    RST_n_SG <= '1';

    case CURRENT_STATE is
        when RESET =>
            RST_n_SG <= '0';  -- Assert reset signal during RESET state
            VOUT_HOLD <= '0';  -- Hold VOUT low during RESET state


        when IDLE =>
            EN_IN_REG_SG <= '1';   -- Enable input registers for data capture
            EN_SHIFT_SG <= '0';        -- Disable shift register in IDLE
            EN_PIPELINE_REG <= '0';  -- Disable pipeline registers
            EN_OUT_REG_SG <= '0';  -- Disable output register
            VOUT_HOLD <= '0';            -- Set VOUT to low in IDLE state


        when PROCESSING =>
            EN_IN_REG_SG <= '1';       -- Enable input registers for 3 data streams
            EN_SHIFT_SG <= '1';            -- Enable shift register to process 3 streams in parallel
            EN_PIPELINE_REG <= '1'; -- Enable pipeline registers to store shifted data
            EN_OUT_REG_SG <= '1';      -- Enable output register to store results from all 3 streams
            VOUT_HOLD <= '1';               -- Assert VOUT to indicate the data is ready for output



    end case;
end process;

end arch_fir_filter_advanced;
```