3rd ORDER CIC DECIMATION FILTER DESIGN & IMPLEMENTATION

[1]Mir Mohammad Burhan, [2]Nadeem Tariq Beigh, [3]Sandeep Kr. Singh
[1]Student, Sharda University, [2]Student, Sharda University, [3]Assistant Professor, Sharda University
[1, 2, 3] Department of Electronics & Communication
[1, 2, 3]Sharda University, Greater Noida, UP, India

***Abstract:** In this paper the design of 3rd order decimation filter with a decimation factor 16 is presented using the FDA tool of DSP system toolbox of MATLAB. The filter is realized using Simulink. The filter is optimized for the size of adders and subtractors in terms of the word length by reducing the word length of consecutive adders to save area and computational complexity. The designed filter is converted to a Verilog RTL using the HDL coder in MATLAB. This filter is implemented in Xilinx and simulated in ISIM simulator for white noise, impulse and step response for 1000 samples. An optimized Verilog RTL implementation of the same filter is also presented. This optimized code is less complex and understandable and has the same functionality with lesser resource utilization.*

Index Terms – Decimation filters, CIC filters, Verilog RTL, Filter design, Filter implementation.

I. INTRODUCTION

Delta-sigma analog-to-digital converters (ADCs) are among the most popular converters that are suitable for low-to-medium speed and high resolution applications such as communications systems, weighing scales and precision measurement applications. These converters use clock oversampling along with noise-shaping to achieve high signal-to-noise ratio (SNR) and, thus, higher effective number of bits (ENOB). Noise-shaping occurs when noise is pushed to higher frequencies that are out of the band of interest. When the out-of-band noise is chopped off, SNR increases. After noise-shaping, the sampling rate is reduced back to its Nyquist rate by means of decimation filters. In fact, delta-sigma converters can be partitioned into two main building blocks: modulator and decimation filters. With a good understanding of these building blocks you can arrive at a robust and efficient design. Each of these main building blocks, in turn, is made of several different building blocks themselves. Unlike conventional converters that sample the analog input signal at Nyquist frequency or slightly higher, delta-sigma modulators, regardless of their order, sample the input data at rates that are much higher than Nyquist rate. The oversampling ratio (OSR) usually is expressed in the form of 2m, where m=1, 2, 3… The modulator is an analog block in nature that needs little accuracy in its components. Thus, the burden of design can be pushed onto the decimation filter. The most important role of a decimation filter is to decrease the sampling rate by discarding every few samples. Presented in this article is a quick overview of decimation filters, along with their operation and requirements. Their requirements are based on the order of the delta-sigma modulator used in the converter, along with the overall number of bits being output by the ADC. [2]

II. DIGITAL DECIMATION FILTER

The decimation filter described in this paper is used in the field of instrumentation. At first, a bit-stream with the rate of 3.2 KHz is generated by a delta-sigma modulator which is over-sampled by 16. Then, the bit-stream passes through a decimation filter, where it is down-sampled to a signal bandwidth of, which has a pass-band ripple of 0.001dB, and the resolution of output is bits. Multiplier takes up most of the chip area and power consumption. But the cascaded-integrator-comb (CIC) filter requires neither multiplier nor coefficient storage, therefore they are widely used in decimation filters.[5]

CIC filter is a very simple digital filter, which is also the Finite Impulse Response (FIR). The transfer function of it is given below:

$$H(z) = \sum_{n=0}^{N-1} z^{-n} = \frac{1-z^{-n}}{1-z^{-1}} \qquad (1)$$

The research shows that the stop-band attenuation of CIC filter cannot meet the practical application. How to improve its stop-band attenuation? We can use the modified form of transfer function, which is given below.

$$H(z) = \left(\frac{1}{N}\sum_{n=0}^{N-1} z^{-n}\right)^{k} = \left(\frac{1}{N}\frac{1-z^{-n}}{1-z^{-1}}\right)^{k} \qquad (2)$$

In this function, N is the normalization factor which represents the decimation factor, and K is the CIC filter's order. The stop-band attenuation is fold-increase with K and the edge of pass-band becomes steeper, so the characteristics of the filter become better. In this paper, N=16.

In the CIC filter, K- the order of the transfer function determines the ability of noise suppression at zeros, and it has a certain relation with the output of the modulator. Noise Power Spectrum Density of the modulator is given by (3)

$$N(f) = \hat{e}\sqrt{2\tau}(2\sin(\pi f\tau))^{L}, 0 \le f \le f_0 \qquad (3)$$

L represents the modulator's order and τ represents the sampling period of the analog modulator, and $\hat{e}$ represents quantization error. If the order of the CIC filter, K, is equal to the order of the modulator, L, the Noise Power Spectrum Density of the signal that output of the CIC filter is shown by (4)

$$N(f) = \hat{e}\sqrt{2N\tau}(2\sin(\pi f\tau))^{L}, 0 \le f \le f_0 \qquad (4)$$

If K=L+1, the Noise Power Spectrum Density of the signal that output of the CIC filter is given by (5)

$$N(f) = \hat{e}\sqrt{2N\tau}(2\sin(\pi f\tau))^{L}\frac{sinc(f\tau)}{sinc(Kf\tau)}, 0 \le f \le f_0 \qquad (5)$$

Compare with (3), (4) and (5), it is not difficult to find that the change of Noise Power Spectrum Density after the down-sample by CIC filter is very small when K=L+1. From (5), we can find that with the increase of K, the performance of the output signal becomes better, but it will make hardware consumption largely increased. So we make K=L+1=3 in our design. The transfer function is given below. [3]

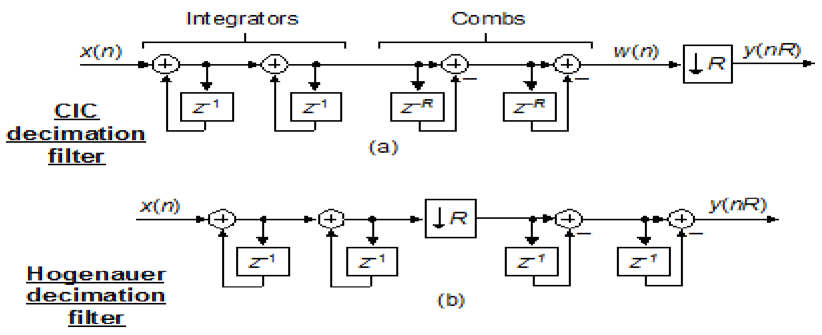


Fig.1: Hongernauer decimation filter.[4]

$$H(z) = \left(\frac{1}{16}\sum_{n=0}^{15} z^{-n}\right)^3 = \left(\frac{1}{16}\frac{1-z^{-16}}{1-z^{-1}}\right)^3 \qquad (6)$$

A proper structure can effectively reduce the power consumption and area. As we studied, there are four popular structures [1]. Figure 2 presents the more efficient structures, which are Hongernauer CIC structure and multi-rate structure. They have different advantages. For example, the advantage of the first one is when over-sampling rate grows up area will change with it; and the advantage of the second one is when the over-sampling rate grows up and the word length becomes small, the power consumption will be down, in this paper, we use Hongernauer CIC structure.[6]

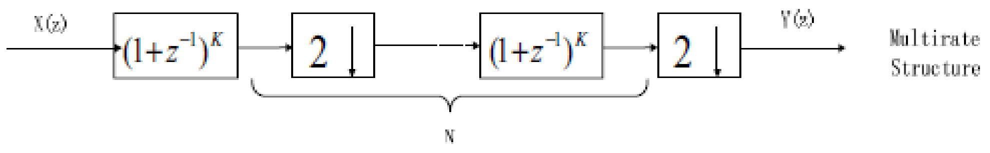


Fig.2: Multi rate structure.[4]

III. DESIGN OF DECIMATION FILTER

In this paper the 3rd order CIC decimator with a decimation factor of 16 is presented. The filter is designed using MATLAB Simulink FDA tool and implemented using Verilog RTL. The designed filter is compared with another CIC Verilog implementation for resource utilization. Given below is the parameter specification of the filter designed using the Simulink FDA tool. The Verilog code for the filter is given in Appendix. The Frequency – Phase response and the Pole-Zero plot is given below as obtaoned from the FDA tool.

Table 1: Filter specifications.

Parameter	Value
Filter Structure	Cascaded Integrator Comb
Decimation factor	16
Differential Delay	1
Number of sections	3
Stable	Yes
Linear Phase	Yes (Type 2)
Number of Multipliers	0
Number of Adders	6
Number of States	6
Multiplicatins per input sample	0
Additions per input sample	3.186
Sampling Frequency	3.2Khz
Wordlengths	[21 20 19 19 17 16]

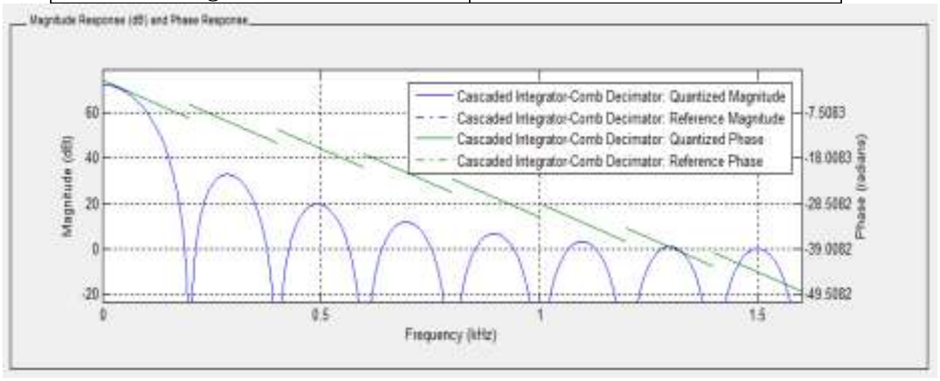


Fig.3: Frequency – Phase response of the filter

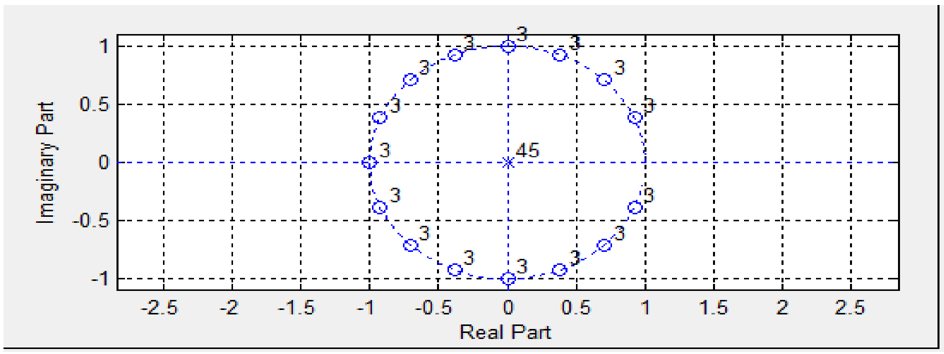


Fig.4: Pole-Zero plot of the filter.

It can be seen that the pass band gain of the filter is around 70 dB and the phase is linear over this region with a range of -6 to – 20 degrees. The PZ plot has all zeros on the unit circle with a poles at the origin. The generated Simulink model is given below with three integrators and three differentiators connected with a down sample switch of 16. [8]

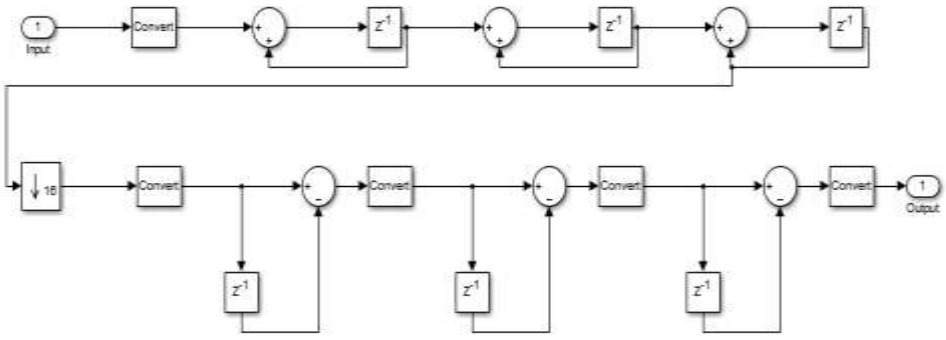
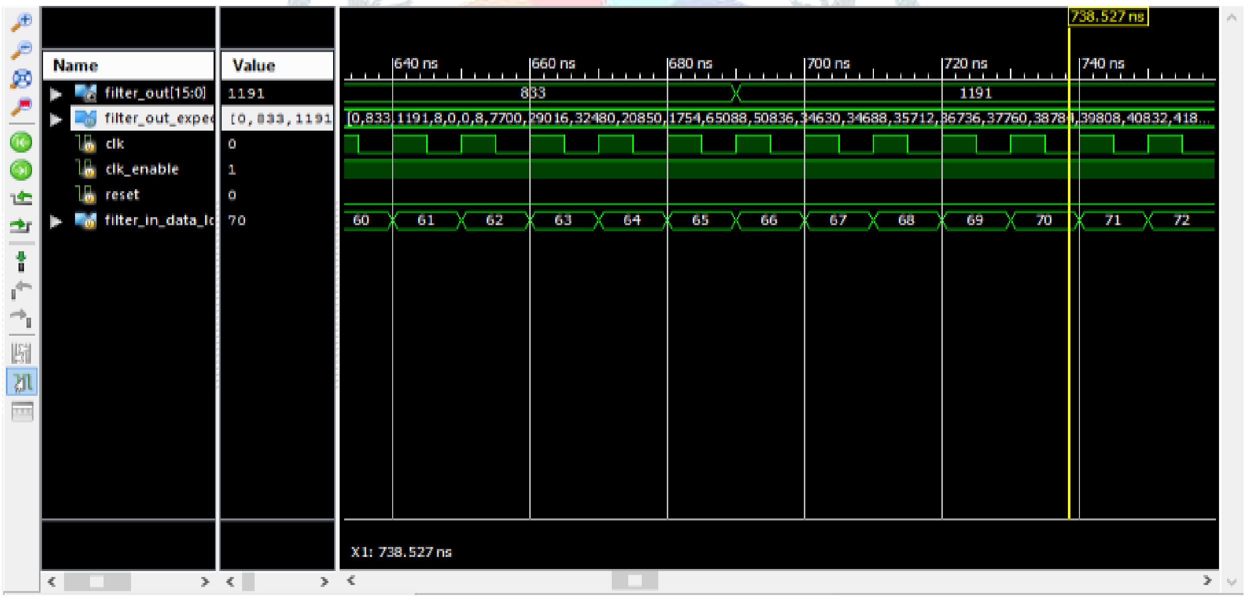


Fig.5: Simulink model of the filter.



Name	Value
filter_out[15:0]	1191
filter_out_exped	[0,833,1191
clk	0
clk_enable	1
reset	0
filter_in_data_lc	70

Fig.6: Simulation output.

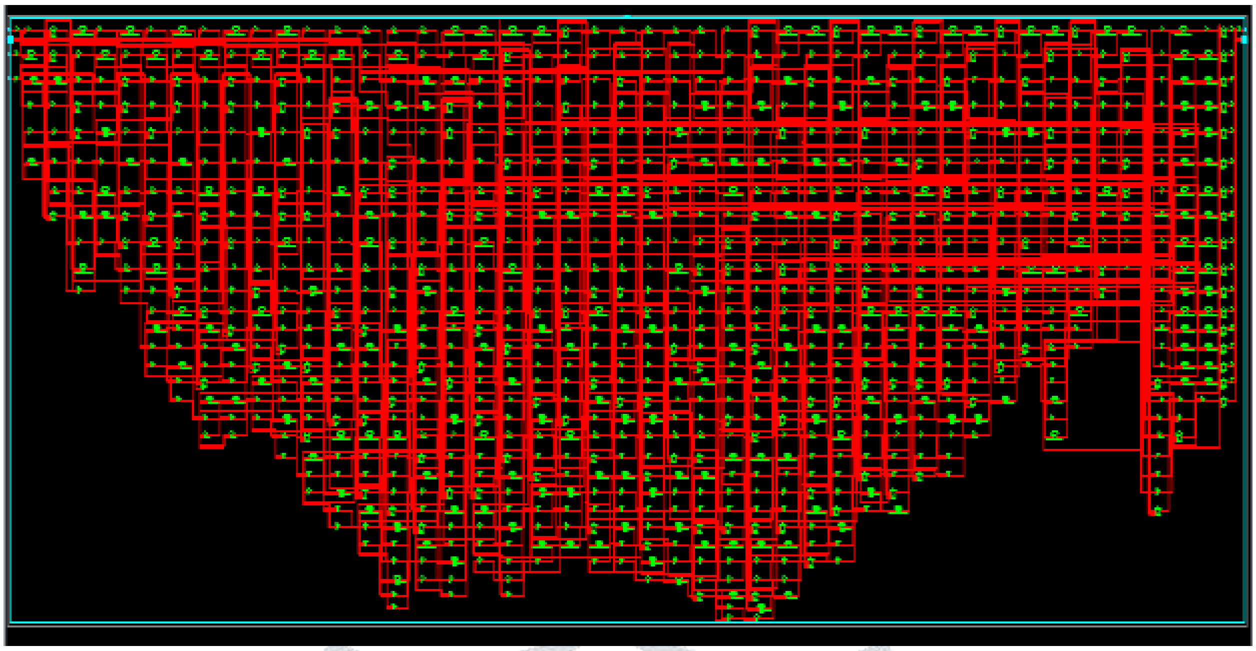


Fig.7: Technology Schematics.

Table 2: Synthesis Report.

Adders/Subtractors	**6**
17-bit subtractor	1
18-bit subtractor	1
20-bit adder	1
20-bit subtractor	1
21-bit adder	1
22-bit adder	1
Counters	**1**
4-bit up counter	1
Registers	**3**
Flip-Flops	3

Table 3: Design Statistics

IOs	36
Cell Usage	
BELS	
AND2	206
AND3	7
AND4	1
INV	136
OR2	197
OR3	6
XOR2	217
Flip Flops/Latches	
FDC	1
FDCE	176

IO Buffers	
IBUF	12
OBUF	17

An optimized decimation filter code is presented in Appendix, with the simulation output and other parameters given below.

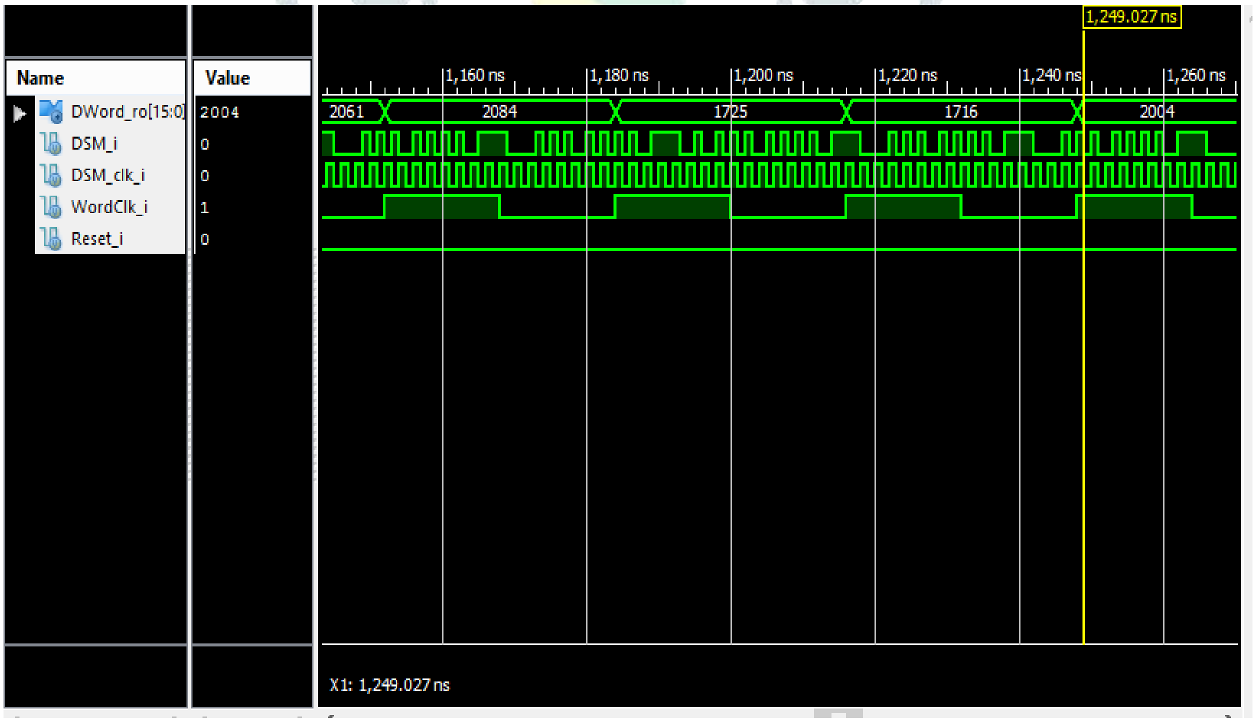


Fig. 8: Output Simulation of optimized filter.

Fig.9: Technology schematics of optimized filter.

Table 4: Synthesis Report.

Adders/Subtractors	**6**
17-bit subtractor	1
18-bit subtractor	1
20-bit adder	1
20-bit subtractor	1
21-bit adder	1
22-bit adder	1
Counters	**1**
4-bit up counter	1
Tristates	**1**
8-bit Tristates	1

Table 5: Design statistics

IOs	12
Cell Usage	
BELS	
AND2	13
AND8	1
INV	2
XOR2	14
XOR2	14
Flip Flops/Latches	
FDC	16
IO Buffers	
IBUF	4
OBUFE	8

IV. Conclusion

In this paper the design of 3rd order decimation filter with a decimation factor 16 was presented using the FDA tool of DSP system toolbox of MATLAB. The filter was realized using Simulink. The filter was optimized for the size of adders and subtractors in terms of the word length by reducing the word length of consecutive adders to save area and computational complexity. The designed filter was converted to a Verilog RTL using the HDL coder in MATLAB. This filter was implemented in Xilinx and simulated in ISIM simulator for white noise, impulse and step reponse for 1000 samples. An optimized Verilog RTL implementation of the same filter was also presented. This optimized code is less complex and understandable and has the same functionality with lesser resource utilization as verified by the tables given above. The area utilization in terms of I/O buffers is 1/3rd .the cell usage reduces considerably aby an amount of around 600.Ths reducing the area by about 64%.This shows that the MATLAB generated HDL code is less efficient as compared to the presented code.

References

[1] Bibin John, Fabian Wagner and Wolfgang H. Krautschneider, 2010. "Comparison of Decimation Filter Architectures for a Sigma-Delta Analog to Digital Converter", TUHH Telematics.

[2] Arash loloee, 2014. “Exploring Decimation Filters”, Journal of High Frequency Electronics”, 31-40.

[3] Kiran Agarwal Gupta, Tejashree Patil, 2017. “Low Pass Reconfigurable Decimation Filter Architecture for 3.8 MHz Frequency”, 2nd IEEE International Conference on Recent Trends in Electronics Information & Communication Technology (RTEICT) India.

[4] Karuna Grover, Rajesh Mehra, Chandani, 2017. “FPGA Based Decimator Using Fully Parallel Technique for Hearing Aid Applications”, 3rd IEEE International Conference on "Computational Intelligence and Communication Technology".

[5] Ricardo Garcia Baez, Gordana Jovanovic Dolecek, 2016. “Modified Comb Decimation Filter: Design and Implementation”, IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS) Abu Dhabi UAE.

[6] YongSheng Wang, YaQin Ru, Yang Liu, Xiao Zhou, Shan Li, Bei Cao, XiaoWei Liu, 2016. “An Area-Efficient Multi-Bit Sigma-Delta modulator”, IEEE.

[7] Latha Raja Gopal, 2016. "Power and Area Efficient Decimation Filter Architectures of Wireless Receivers", Springer.

[8] Nasir Nabi Hurrah, Zubair Jan, Anil Bhardwaj, Shabir Ahmad Parah, Amit Kant Pandit, 2015. “Oversampled Sigma Delta ADC Decimation Filter: Design Techniques, Challenges, Trade-offs and Optimization”, Proceedings of 2015 RAECS UIET Panjab University Chandigarh 21-22nd December 2015.

[9] K.Shanthi, Dr.B.K.Madhavi, 2015. “Implementation of Optimized Cascaded Integrator Comb Filters for Digital Up, Down Conversions”, International Journal of Science Engineering and Technology Research (IJSETR), Vol 4, Issue 7.

[10] Sudhir Rao Rupanagudi, Varsha G. Bhat, Hemalatha S.G., Bhavana N., Archana M., Chandrika B. V., Ashwini R., Keerti G. Torvi, Darshan S. R., Abhilash B.G., Anil K.S., Vinayak Swamy K. M, 2014.“Design of a Low Power Digital Down Converter for 802.16m – 4G WiMAX on FPGA”, International Conference on Advances in Computing, Communications and Informatics (ICACCI).

[11] Suraj R. Gaikwad, Gopal S. Gawande, 2014."Design and Implementation of Efficient FIR Filter Structures using Xilinx System Generator", International Journal of scientific research and management (IJSRM).

APPENDIX

Verilog RTL code of CIC decimation filter designed by MATLAB HDL Coder.

```
MODULE DECIMATION_16
        (
         CLK,   CLK_ENABLE,   RESET,   FILTER_IN,   FILTER_OUT,
CE_OUT
         );
  INPUT    CLK;  INPUT    CLK_ENABLE;  INPUT    RESET;  INPUT
SIGNED [15:0] FILTER_IN; //SFIX16_EN15
  OUTPUT SIGNED [15:0] FILTER_OUT; OUTPUT CE_OUT;
////////////////////////////////////////////////////////////////
//MODULE ARCHITECTURE: DECIMATION_16
////////////////////////////////////////////////////////////////
  REG [3:0] CUR_COUNT; // UFIX4
  WIRE PHASE_1; // BOOLEAN
  WIRE CE_DELAYLINE; // BOOLEAN
  REG INT_DELAY_PIPE [0:1]; // BOOLEAN
  WIRE CE_GATED; // BOOLEAN
  REG CE_OUT_REG; // BOOLEAN
  //
  REG SIGNED [15:0] INPUT_REGISTER; // SFIX16_EN15
  //   -- SECTION 1 SIGNALS
  WIRE SIGNED [15:0] SECTION_IN1; // SFIX16_EN15
  WIRE SIGNED [20:0] SECTION_CAST1; // SFIX21_EN8
  WIRE SIGNED [20:0] SUM1; // SFIX21_EN8
  REG SIGNED [20:0] SECTION_OUT1; // SFIX21_EN8
  WIRE SIGNED [20:0] ADD_CAST; // SFIX21_EN8
  WIRE SIGNED [20:0] ADD_CAST_1; // SFIX21_EN8
  WIRE SIGNED [21:0] ADD_TEMP; // SFIX22_EN8
  //   -- SECTION 2 SIGNALS
  WIRE SIGNED [20:0] SECTION_IN2; // SFIX21_EN8
  WIRE SIGNED [19:0] SECTION_CAST2; // SFIX20_EN7
  WIRE SIGNED [19:0] SUM2; // SFIX20_EN7
  REG SIGNED [19:0] SECTION_OUT2; // SFIX20_EN7
  WIRE SIGNED [19:0] ADD_CAST_2; // SFIX20_EN7
  WIRE SIGNED [19:0] ADD_CAST_3; // SFIX20_EN7
  WIRE SIGNED [20:0] ADD_TEMP_1; // SFIX21_EN7
  //   -- SECTION 3 SIGNALS
  WIRE SIGNED [19:0] SECTION_IN3; // SFIX20_EN7
  WIRE SIGNED [18:0] SECTION_CAST3; // SFIX19_EN6
  WIRE SIGNED [18:0] SUM3; // SFIX19_EN6
  REG SIGNED [18:0] SECTION_OUT3; // SFIX19_EN6
  WIRE SIGNED [18:0] ADD_CAST_4; // SFIX19_EN6
  WIRE SIGNED [18:0] ADD_CAST_5; // SFIX19_EN6
  WIRE SIGNED [19:0] ADD_TEMP_2; // SFIX20_EN6
  //   -- SECTION 4 SIGNALS
  WIRE SIGNED [18:0] SECTION_IN4; // SFIX19_EN6
  REG SIGNED [18:0] DIFF1; // SFIX19_EN6
  WIRE SIGNED [18:0] SECTION_OUT4; // SFIX19_EN6
  WIRE SIGNED [18:0] SUB_CAST; // SFIX19_EN6
  WIRE SIGNED [18:0] SUB_CAST_1; // SFIX19_EN6
  WIRE SIGNED [19:0] SUB_TEMP; // SFIX20_EN6
  REG SIGNED [18:0] CIC_PIPELINE4; // SFIX19_EN6
  //   -- SECTION 5 SIGNALS
  WIRE SIGNED [18:0] SECTION_IN5; // SFIX19_EN6
  WIRE SIGNED [16:0] SECTION_CAST5; // SFIX17_EN4
  REG SIGNED [16:0] DIFF2; // SFIX17_EN4
  WIRE SIGNED [16:0] SECTION_OUT5; // SFIX17_EN4
  WIRE SIGNED [16:0] SUB_CAST_2; // SFIX17_EN4
  WIRE SIGNED [16:0] SUB_CAST_3; // SFIX17_EN4
  WIRE SIGNED [17:0] SUB_TEMP_1; // SFIX18_EN4
  REG SIGNED [16:0] CIC_PIPELINE5; // SFIX17_EN4
  //   -- SECTION 6 SIGNALS
  WIRE SIGNED [16:0] SECTION_IN6; // SFIX17_EN4
  WIRE SIGNED [15:0] SECTION_CAST6; // SFIX16_EN3
  REG SIGNED [15:0] DIFF3; // SFIX16_EN3
  WIRE SIGNED [15:0] SECTION_OUT6; // SFIX16_EN3
  WIRE SIGNED [15:0] SUB_CAST_4; // SFIX16_EN3
  WIRE SIGNED [15:0] SUB_CAST_5; // SFIX16_EN3
  WIRE SIGNED [16:0] SUB_TEMP_2; // SFIX17_EN3
  REG SIGNED [15:0] OUTPUT_REGISTER; // SFIX16_EN3
  //   ------------------ CE OUTPUT GENERATION ------------------
  ALWAYS @ (POSEDGE CLK OR POSEDGE RESET)
   BEGIN: CE_OUTPUT
     IF (RESET == 1'B1) BEGIN
       CUR_COUNT <= 4'B0000;
       END
       ELSE BEGIN
         IF (CLK_ENABLE == 1'B1) BEGIN
           IF (CUR_COUNT == 4'B1111) BEGIN
             CUR_COUNT <= 4'B0000;
           END
           ELSE BEGIN
             CUR_COUNT <= CUR_COUNT + 1;
           END
         END
       END
     END // CE_OUTPUT
   ASSIGN PHASE_1 = (CUR_COUNT == 4'B0001 && CLK_ENABLE ==
1'B1)? 1: 0;
   ALWAYS @ (POSEDGE CLK OR POSEDGE RESET)
     BEGIN: CE_DELAY
       IF (RESET == 1'B1) BEGIN
         INT_DELAY_PIPE [0] <= 1'B0;
         INT_DELAY_PIPE [1] <= 1'B0;
       END
       ELSE BEGIN
         IF (PHASE_1 == 1'B1) BEGIN
           INT_DELAY_PIPE [1] <= INT_DELAY_PIPE [0];
           INT_DELAY_PIPE [0] <= CLK_ENABLE;
         END
       END
     END // CE_DELAY
   ASSIGN CE_DELAYLINE = INT_DELAY_PIPE [1];
   ASSIGN CE_GATED = CE_DELAYLINE & PHASE_1;
   //   ------------------ CE OUTPUT REGISTER ------------------
   ALWAYS @ (POSEDGE CLK OR POSEDGE RESET)
     BEGIN: CE_OUTPUT_REGISTER
       IF (RESET == 1'B1) BEGIN
         CE_OUT_REG <= 1'B0;
       END
       ELSE BEGIN
           CE_OUT_REG <= CE_GATED;
       END
     END // CE_OUTPUT_REGISTER
   //   ------------------ INPUT REGISTER -----------------
   ALWAYS @ (POSEDGE CLK OR POSEDGE RESET)
     BEGIN: INPUT_REG_PROCESS
       IF (RESET == 1'B1) BEGIN
         INPUT_REGISTER <= 0;
       END
       ELSE BEGIN
         IF (CLK_ENABLE == 1'B1) BEGIN
           INPUT_REGISTER <= FILTER_IN;
         END
       END
     END // INPUT_REG_PROCESS
   //   ------------------ SECTION # 1: INTEGRATOR ------------------
   ASSIGN SECTION_IN1 = INPUT_REGISTER;
   ASSIGN  SECTION_CAST1  =  $SIGNED  ({{12{SECTION_IN1  [15]}},
SECTION_IN1 [15:7]});
   ASSIGN ADD_CAST = SECTION_CAST1;
   ASSIGN ADD_CAST_1 = SECTION_OUT1;
   ASSIGN ADD_TEMP = ADD_CAST + ADD_CAST_1;
   ASSIGN SUM1 = ADD_TEMP [20:0];
   ALWAYS @ (POSEDGE CLK OR POSEDGE RESET)
     BEGIN: INTEGRATOR_DELAY_SECTION1
       IF (RESET == 1'B1) BEGIN
         SECTION_OUT1 <= 0;
       END
       ELSE BEGIN
         IF (CLK_ENABLE == 1'B1) BEGIN
           SECTION_OUT1 <= SUM1;
         END
       END
     END // INTEGRATOR_DELAY_SECTION1
   //   ------------------ SECTION # 2: INTEGRATOR ------------------
   ASSIGN SECTION_IN2 = SECTION_OUT1;
   ASSIGN SECTION_CAST2 = SECTION_IN2 [20:1];
   ASSIGN ADD_CAST_2 = SECTION_CAST2;
ASSIGN ADD_CAST_3 = SECTION_OUT2;
ASSIGN ADD_TEMP_1 = ADD_CAST_2 + ADD_CAST_3;
ASSIGN SUM2 = ADD_TEMP_1 [19:0];
ALWAYS @ (POSEDGE CLK OR POSEDGE RESET)
 BEGIN: INTEGRATOR_DELAY_SECTION2
  IF (RESET == 1'B1) BEGIN
   SECTION_OUT2 <= 0;
  END
  ELSE BEGIN
   IF (CLK_ENABLE == 1'B1) BEGIN
    SECTION_OUT2 <= SUM2;
   END
  END
 END // INTEGRATOR_DELAY_SECTION2
//   ------------------ SECTION # 3: INTEGRATOR ------------------
ASSIGN SECTION_IN3 = SECTION_OUT2;
ASSIGN SECTION_CAST3 = SECTION_IN3 [19:1];
ASSIGN ADD_CAST_4 = SECTION_CAST3;
ASSIGN ADD_CAST_5 = SECTION_OUT3;
ASSIGN ADD_TEMP_2 = ADD_CAST_4 + ADD_CAST_5;
ASSIGN SUM3 = ADD_TEMP_2 [18:0];
ALWAYS @ (POSEDGE CLK OR POSEDGE RESET)
 BEGIN: INTEGRATOR_DELAY_SECTION3
  IF (RESET == 1'B1) BEGIN
   SECTION_OUT3 <= 0;
  END
  ELSE BEGIN
   IF (CLK_ENABLE == 1'B1) BEGIN
    SECTION_OUT3 <= SUM3;
   END
  END
 END // INTEGRATOR_DELAY_SECTION3
//   ------------------ SECTION # 4: COMB ------------------
ASSIGN SECTION_IN4 = SECTION_OUT3;
ASSIGN SUB_CAST = SECTION_IN4;
ASSIGN SUB_CAST_1 = DIFF1;
ASSIGN SUB_TEMP = SUB_CAST - SUB_CAST_1;
ASSIGN SECTION_OUT4 = SUB_TEMP [18:0];
ALWAYS @ (POSEDGE CLK OR POSEDGE RESET)
 BEGIN: COMB_DELAY_SECTION4
  IF (RESET == 1'B1) BEGIN
   DIFF1 <= 0;
  END
  ELSE BEGIN
   IF (PHASE_1 == 1'B1) BEGIN
    DIFF1 <= SECTION_IN4;
   END
  END
 END // COMB_DELAY_SECTION4
ALWAYS @ (POSEDGE CLK OR POSEDGE RESET)
 BEGIN: CIC_PIPELINE_PROCESS_SECTION4
  IF (RESET == 1'B1) BEGIN
   CIC_PIPELINE4 <= 0;
  END
  ELSE BEGIN
   IF (PHASE_1 == 1'B1) BEGIN
    CIC_PIPELINE4 <= SECTION_OUT4;
   END
  END
 END // CIC_PIPELINE_PROCESS_SECTION4
//   ------------------ SECTION # 5: COMB ------------------
ASSIGN SECTION_IN5 = CIC_PIPELINE4;
ASSIGN SECTION_CAST5 = SECTION_IN5 [18:2];
ASSIGN SUB_CAST_2 = SECTION_CAST5;
ASSIGN SUB_CAST_3 = DIFF2;
ASSIGN SUB_TEMP_1 = SUB_CAST_2 - SUB_CAST_3;
ASSIGN SECTION_OUT5 = SUB_TEMP_1 [16:0];
ALWAYS @ (POSEDGE CLK OR POSEDGE RESET)
 BEGIN: COMB_DELAY_SECTION5
  IF (RESET == 1'B1) BEGIN
   DIFF2 <= 0;
  END
  ELSE BEGIN
   IF (PHASE_1 == 1'B1) BEGIN
    DIFF2 <= SECTION_CAST5;
   END
  END
 END // COMB_DELAY_SECTION5
ALWAYS @ (POSEDGE CLK OR POSEDGE RESET)
 BEGIN: CIC_PIPELINE_PROCESS_SECTION5
  IF (RESET == 1'B1) BEGIN
   CIC_PIPELINE5 <= 0;
  END
  ELSE BEGIN
   IF (PHASE_1 == 1'B1) BEGIN
    CIC_PIPELINE5 <= SECTION_OUT5;
   END
  END
 END // CIC_PIPELINE_PROCESS_SECTION5
//   ------------------ SECTION # 6: COMB ------------------
ASSIGN SECTION_IN6 = CIC_PIPELINE5;
ASSIGN SECTION_CAST6 = SECTION_IN6 [16:1];
ASSIGN SUB_CAST_4 = SECTION_CAST6;
ASSIGN SUB_CAST_5 = DIFF3;
ASSIGN SUB_TEMP_2 = SUB_CAST_4 - SUB_CAST_5;
ASSIGN SECTION_OUT6 = SUB_TEMP_2 [15:0];
ALWAYS @ (POSEDGE CLK OR POSEDGE RESET)
 BEGIN: COMB_DELAY_SECTION6
  IF (RESET == 1'B1) BEGIN
   DIFF3 <= 0;
  END
  ELSE BEGIN
   IF (PHASE_1 == 1'B1) BEGIN
    DIFF3 <= SECTION_CAST6;
   END
  END
 END // COMB_DELAY_SECTION6
//   ------------------ OUTPUT REGISTER ------------------
ALWAYS @ (POSEDGE CLK OR POSEDGE RESET)
 BEGIN: OUTPUT_REG_PROCESS
  IF (RESET == 1'B1) BEGIN
   OUTPUT_REGISTER <= 0;
  END
  ELSE BEGIN
   IF (PHASE_1 == 1'B1) BEGIN
    OUTPUT_REGISTER <= SECTION_OUT6;
   END
  END
 END // OUTPUT_REG_PROCESS
// ASSIGNMENT STATEMENTS
ASSIGN CE_OUT = CE_OUT_REG;
ASSIGN FILTER_OUT = OUTPUT_REGISTER;
ENDMODULE // DECIMATION_16
```

Verilog RTL code for the optimized decimation filter

```
MODULE SINC_IMLEMENTATION (
DSM_I, DSM_CLK_I, WORDCLK_I, RESET_I, DWORD_RO);
INPUT DSM_CLK_I; // DSM-RATE CLOCK (BIT CLOCK)
INPUT WORDCLK_I; // OUTPUT WORD-RATE CLOCK
INPUT RESET_I; // ACTIVE-HI RESET
INPUT DSM_I; // INPUT FROM MODULATOR
OUTPUT [15:0] DWORD_RO; // 16-BIT OUTPUT WORD
REG [20:0] ACC1_R; REG [19:0] ACC2_R;
REG [18:0] ACC3_R; REG [18:0] ACC3_Q2_R;
REG [18:0] DIFF1_R; REG [16:0] DIFF2_R;
REG [15:0] DIFF3_R; REG [18:0] DIFF1_Q1_R;


REG [16:0] DIFF2_Q2_R; REG [15:0] DWORD_RO;
REG [15:0] TEMP;
//REG [7:0] WORD_COUNT;
//
// INTERNAL WIRES
//
// 2'S-COMP VERSION OF DWORD
WIRE [20:0] DWORD_2COMP_W;
// SINC FILTER
ASSIGN DWORD_2COMP_W = (DSM_I==1'B0)? 21’D0: 21'D1;
// ACCUMULATOR (INTEGRATOR)
ALWAYS @ (NEGEDGE DSM_CLK_I OR POSEDGE RESET_I)
BEGIN
IF (RESET_I)
BEGIN
/* INITIALIZE ACC REGISTERS ON RESET_I */
ACC1_R<=21'D0;
ACC2_R<=20'D0;
ACC3_R<=19'D0;
END
ELSE
BEGIN
/* PERFORM ACCUMULATION PROCESS */
```

```
ACC1_R<=ACC1_R + DWORD_2COMP_W;
ACC2_R<=ACC2_R + ACC1_R;
ACC3_R<=ACC3_R + ACC2_R;
END
END





//
// DIFFERENTIATOR AND DECIMATION
//
ALWAYS @ (POSEDGE WORDCLK_I OR POSEDGE RESET_I)
BEGIN
IF (RESET_I)
 BEGIN
ACC3_Q2_R<=19'D0;
DIFF1_Q1_R<=19'D0;
DIFF2_Q2_R<=17'D0;
DIFF1_R<=19'D0;
DIFF2_R<=17'D0;
DIFF3_R<=16'D0;
TEMP<=16'D0;
END
ELSE
BEGIN
DIFF1_R<= ACC3_R - ACC3_Q2_R;
DIFF2_R<=DIFF1_R - DIFF1_Q1_R;
DIFF3_R<=DIFF2_R - DIFF2_Q2_R;
ACC3_Q2_R<=ACC3_R;
DIFF1_Q1_R<=DIFF1_R;
DIFF2_Q2_R<=DIFF2_R;
TEMP<=DIFF3_R;
DWORD_RO<=TEMP;
END
END
ENDMODULE
```