**ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY**

# DIGITAL INTERPOLATION AND MODULATION SYSTEM DESIGN FOR COMMUNICATION DACS

**M.Sc. THESIS**

**Gürer ÖZBEK**

**Department of Electronics and Communication Engineering**

**Electronics Engineering Programme**

**JANUARY 2013**

**ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY**

# DIGITAL INTERPOLATION AND MODULATION SYSTEM DESIGN FOR COMMUNICATION DACS

**M.Sc. THESIS**

**Gürer ÖZBEK**
**504091264**

**Department of Electronics and Communication Engineering**

**Electronics Engineering Programme**

**Thesis Advisor: Asst. Prof. Dr. Türker Küyel**

**JANUARY 2013**

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

# HABERLEŞME S/A DÖNÜŞTÜRÜCÜLERİ İÇİN SAYISAL ARA DEĞERLEME VE MODÜLASYON SİSTEMİ TASARIMI

YÜKSEK LİSANS TEZİ

Gürer ÖZBEK
504091264

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Yrd. Doç. Dr. Türker Küyel

OCAK 2013

**Gürer Özbek**, an **M.Sc.** student of ITU **Graduate School of Science Engineering and Technology** student ID **504091264**, successfully defended the **thesis** entitled "**Digital Interpolation and Modulation System Design for Communication DACs**", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

| | | |
|---|---|---|
| **Thesis Advisor :** | **Asst. Prof. Dr. Türker KÜYEL**<br>İstanbul Technical University | .............................. |
| **Jury Members :** | **Prof. Dr. Ali TOKER**<br>İstanbul Technical University | ............................. |
| | **Prof. Dr. Günhan DÜNDAR**<br>Boğaziçi University | ............................. |

**Date of Submission : 14 December 2012**
**Date of Defense : 22 January 2013**

## FOREWORD

I would like to thank my supervisor Assist. Prof. Türker Küyel who has guided me in my studies.
I would like to thank Tübitak for giving me a scholarship.
Finally, I would like express my gratefulness to my family for their unconditional support.

January 2013

Gürer ÖZBEK
Electronics Engineer

**TABLE OF CONTENTS**

## ABBREVIATIONS

| | |
|---|---|
| **ADC** | : Analog to Digital Converter |
| **BPF** | : Band Pass Filter |
| **CML** | : Common Mode Logic |
| **CMOS** | : Complementary Metal Oxide Semiconductor |
| **DAC** | : Digital to Analog Converter |
| **DSP** | : Digital Signal Processing |
| **F1** | : Filter-1 |
| **F2** | : Filter-2 |
| **F3** | : Filter-3 |
| **FFT** | : Fast Fourier Transform |
| **FIR** | : Finite Impulse Response |
| **FPGA** | : Field Programmable Gate Array |
| **HBF** | : Half Band Filter |
| **HDL** | : Hardware Description Language |
| **HPF** | : High Pass Filter |
| **IF** | : Intermediate Frequency |
| **LPF** | : Low Pass Filter |
| **LVDS** | : Low Voltage Differential Signaling |
| **NRZ** | : Non-Return to Zero |
| **PA** | : Power Amplifier |
| **PAR** | : Place and Route |
| **PCB** | : Printed Circuit Board |
| **PM** | : Plus-Minus |
| **RF** | : Radio Frequency |
| **RZ** | : Return to Zero |
| **SDR** | : Software Defined Radio |
| **SFDR** | : Spurious Free Dynamic Range |
| **SPI** | : Serial Peripheral Interface |
| **TNS** | : Total Negative Slack |
| **WNS** | : Worst Negative Slack |

## LIST OF TABLES

**Page**

**LIST OF FIGURES**

**Page**

# DIGITAL INTERPOLATION AND MODULATION SYSTEM DESIGN FOR COMMUNICATION DACS

## SUMMARY

High speed digital to analog converters (DACs) are used in applications such as cellular base stations or digital TV broadcasting. In such systems, various digital signal processing blocks are also needed. Interpolation, modulation and channel equalization can be given as examples of such digital functionality.

Implementing digital signal processing blocks with the DAC on the same die has certain advantages. On chip oversampling and digital interpolation filtering allows receiving digital data at lower rates. Power hungry high-speed interfaces like low voltage differential signaling (LVDS) or current mode logic (CML) are no longer needed. Furthermore, trace count on the PCB can be reduced.

In this work, design and verification of a digital interpolation and modulation block used in high-speed communication DACs is explained in detail.

Interpolation and modulation features of the DACs used in the industry are examined based on their datasheet specifications.

For the design of the interpolation and modulation system, half-band finite impulse response (FIR) filter topology is used. The number of filter coefficients can be reduced and operational speed can be higher for the same performance level with respect to a conventional FIR filter.

Based on specifications in existing DAC datasheets, an interpolation and modulation system is designed with 85 dB spurious free dynamic range (SFDR) performance. Then, the design is implemented with LFoundry 0.15 μm CMOS technology. Functionality is tested with post-place-and-route (PAR) simulations. Such digital systems with 85 dB SFDR are used with existing DAC designs because the DAC, not the digital filter limits the SFDR performance. Power is traded-off with SFDR in digital filter designs.

As market demands DACs with higher SFDR performance, interpolation and modulation blocks with higher SFDR must support next generation DACs with better SFDR. To support a 90 dB DAC, it is necessary for the digital filter to have 96 dB or better SFDR.

A new interpolation and modulation block, which can support a DAC with 90 dB SFDR is designed. Filter coefficients are calculated with MATLAB's fdatool. 16-bit design is modeled in MATLAB and in Verilog. The design has a user selectable interpolation from 2x to 8x and 41 operation modes in total, including Hilbert transformers. With these interpolation and modulation modes, input signal can be pushed to any band in the output spectrum without distortion. Simulations show that the new block has 99 dB SFDR and no significant ripple or attenuation in the %80 of the Nyquist band and the signal to noise ratio (SNR) is 93.4 dB over full Nyquist band.

New design is synthesized and PAR’ed with TSMC 0.18 µm CMOS technology, since the LFoundry Fab. has moved from Germany to France with uncertain future. Area of the TSMC design is 1.2 mm x 3 mm and the clock speed is 1.2 GHz. The design consumes 1.826 W at 1.2 GHz for two channels.

Several digital additions like a serial peripheral interface (SPI) block, a control block, a RAMDAC and a clock divider block are included in the design. Thus, the whole digital sub-section of a communication DAC is completed.

For verification, the system is embedded to a Virtex 5 field programmable gate array (FPGA). Data is driven by a pattern generator and captured by a logic analyzer. A test methodology that matches the simulation inputs to pattern generator inputs is applied. A program written in C language then compares the outputs of the simulation to logic analyzer capture data. The bit error rate is found to be zero.

Finally, the complete digital system is mixed mode co-simulated with a DAC taken from a different work. Simulations are done with Cadence AMS simulator which supports analog and digital co-simulation. It is shown that the effect of the digital system on SFDR is insignificant with respect to the effects of the DAC, especially with high output frequencies.

# HABERLEŞME S/A DÖNÜŞTÜRÜCÜLERİ İÇİN SAYISAL ARA DEĞERLEME VE MODÜLASYON SİSTEMİ TASARIMI

## ÖZET

İşaret işlemenin sayısal ortamda yapılmasının daha avantajlı olması, işaret zincirlerine A/S ve S/A dönüştürücüleri eklemiştir. Veri haberleşmesi ve işlenmesi alanlarında önemli yer tutan S/A dönüştürücülerin bazı haberleşme uygulamaları için yüksek çözünürlükte ve yüksek hızda olmaları beklenmektedir.

Yüksek hızlı S/A dönüştürücüler, baz istasyonlarından sayısal televizyon yayın sistemlerine kadar pek çok alanda kullanılmaktadır. Bu sistemlerde kullanılan S/A dönüştürücülerle beraber çeşitli sayısal işaret işleme işlemleri de yapılmaktadır. Bunlara örnek olarak ara değerleme, modülasyon, kanal dengeleme gibi işlemler verilebilir.

Sayısal işaret işleme bloklarının S/A dönüştürücü ile aynı kırmık içerisinde üretilmesi, günümüzde endüstrinin yöneldiği bir yaklaşımdır. Ara değerleme işleminin kırmık içerisinde yapılması, kırmık içerisine daha düşük hızlarda veri alınmasını sağlar. Bu sayede kırmık girişlerinde LVDS ve CML gibi karmaşık ve yüksek güç tüketimli yapılar kullanılmasına gerek kalmaz. Ayrıca gerek kırmık içi, gerekse de PCB üzerindeki veri yolları daha esnek şekilde tasarlanabilir.

Bu çalışmada, yüksek hızlı haberleşme sistemlerindeki S/A dönüştürücü kırmıklarında kullanılan bir sayısal ara değerleme ve modülasyon sisteminin tasarım süreci işlenmiştir.

Çalışma kapsamında, endüstride kullanılan S/A dönüştürücü kırmıklarının sayısal ara değerleme ve modülasyon işlevleri katalog bilgileri üzerinden incelenmiştir.

Ara değerleme ve modülasyon sisteminin tasarımında yarım bant FIR süzgeçleri kullanılmıştır. Bu sayede, aynı seçicilik için gereken katsayı adedi yarıya düşerken sistemin büyük bir bölümü çıkıştaki hızın yarısı ile çalıştırılabilmektedir. Bu özellik sayesinde hem güç tüketimi azaltılmakta, hem de daha yüksek çalışma frekanslarına sahip sistemler üretilebilmektedir.

Elde edilen bilgiler ışığında tipik bir ara değerleme ve modülasyon sistemi tasarlanmıştır. Tasarım sürecinde ilk olarak MATLAB yardımıyla parametreler elde edilmiştir. Sistemin oluşturulan MATLAB modelinde 3 süzgeç yer almaktadır. Bu süzgeçlerin giriş çözünürlükleri 16-bit olarak seçilmiştir. Süzgeçler sırasıyla 15, 12 ve 13 bitlik 14, 6 ve 4 katsayı içermektedir.

Yapılan benzetimlerde süzgeçlerin 88, 88 ve 86 dB SFDR'a sahip oldukları görülmüştür. Süzgeçler birlikte kullanıldığında yapılan benzetimlerde ise 85 dB SFDR elde edilmiştir. Ayrıca, gerçeklenen 16 karmaşık modülasyon durumu ile giriş işaret bandının çıkışta farklı frekanslara ötelenmesi sağlanmıştır. Bu durumlar, spektrumda işaretin ötelenemeyeceği bir bölge kalmayacak şekilde seçilmiştir. Modülasyon durumlarının tamamında 8x ara değerleme yapılmaktadır.

Süzgeçlerin performansı anlaşıldıktan sonra LFoundry 0.15 μm CMOS teknolojisi kullanılarak sentez ve PAR işlemleri yapılmıştır. PAR sonrası yapılan benzetimlerle sistemin doğru çalıştığı kontrol edilmiştir.

SFDR performansı 85 dB olan böyle bir sistem, ancak kendisinden daha kötü performansa sahip bir S/A dönüştürücü ile çalıştığında anlamlı olmaktadır. Teknolojinin gelişimi ve pazarın istekleri ile birlikte daha yüksek performanslı S/A dönüştürücülerin üretilmesi yoluna gidilmektedir. Bu da daha yüksek SFDR performanslı ara değerleme ve modülasyon sistemlerine ihtiyaç duyulacağı anlamına gelmektedir. SFDR'ı 90 dB olan böyle bir S/A dönüştürücü ile çalışabilecek sayısal sistemin performansının da en az 95 dB olması gerektiği açıktır.

Gelecekte daha yüksek SFDR performanslı sistemlere ihtiyaç duyulacağından, çalışmanın sonraki kısımlarında 90 dB SFDR gibi daha yüksek performanslı bir S/A dönüştürücü ile beraber çalışabilecek bir tasarım yapılmıştır. Bu tasarım için gereken parametreler MATLAB'in fdatool aracı ile elde edilmiştir. 16 bitlik giriş ve çıkışlara sahip bu tasarım için MATLAB ve Verilog modelleri oluşturulmuştur. Ayrıca, çalışma durumları zenginleştirilerek seçilebilir 41 ara değerleme ve modülasyon durumu gerçeklenmiştir. Bu yeni durumlara örnek olarak 8x, 4x ve 2x ara değerleme, her bir ara değerleme durumuna karşı gelen modülasyon durumları verilebilir. Bir önceki tasarımda olduğu gibi bu tasarımda da giriş işareti, çıkışta istenen banda ötelenebilmektedir.

Yine 3 süzgecin bulunduğu sistemde süzgeçler sırasıyla 18, 16 ve 16 bitlik 16, 6 ve 6 adet katsayı içermektedirler. Süzgeçlerin ölçülen SFDR değerleri 98.3, 99.7 ve 99.7 dB'dir. Ayrıca ilgilenilen geçirme bandında (Nyquist bandının %80'i) zayıflamanın 0 dB olduğu görülmüştür. Süzgeçlerin geçirme bandı dalgalılıkları da önemsenmeyecek derecede düşük ölçülmüştür. Süzgeçlerin SNR değerleri ise sırasıyla 95.4, 94.6 ve 94.6 dB olarak hesaplanmıştır.

Tasarımın TSMC 0.18 μm CMOS teknolojisi ile sentezi ve PAR'ı yapılmıştır. PAR sonrası yapılan benzetimlerde 99 dB SFDR elde edilmiştir. Bu benzetimlerde ayrıca süzgeçlerin grup gecikmelerine de bakılmış, sırasıyla 18, 10 ve 12 saat işareti oldukları görülmüştür. Tasarımın kapladığı alan 1.2mm x 3mm olup 1.2GHz'lik saat işareti ile çalışabilmektedir. Bu hızla ortalama 1.826W güç harcamaktadır.

Çalışma kapsamında ayrıca normalde tümdevre için yapılan tasarımın donanım testleri de yapılmıştır. Bu donanım testlerinde tasarımı anlatılan sayısal ara değerleme ve modülasyon biriminin yanısıra bir haberleşme S/A dönüştürücüsü kırmığında bulunan sayısal arayüz, bellek döngüsü, saat işareti bölücüsü gibi çevresel birimlerin de bulunduğu komple bir sayısal sistem kullanılmıştır. Donanım testleri için sayısal sistem, Xilinx Virtex 5 FPGA'sına gömülmüştür. FPGA'nın sürülmesi Agilent 16822A sayısal veri üreteci ile yapılmış, çıkışları da Agilent 16802A lojik analizörü ile kaydedilmiştir. Testlerin güvenilir şekilde yapılabilmesi için benzetim ile sayısal veri üreteci girişlerinin tamamen aynı olmasını sağlayacak bir test metodu kullanılmıştır. Ayrıca benzetim sonuçları ile donanım testleri sonuçlarının aynılığını gösterebilmek için C diliyle yazılmış bir karşılaştırma programı kullanılmıştır. Program, benzetim ve donanım testi çıktılarını okuyabilmekte ve her bir andaki çıkışları tek tek karşılaştırabilmektedir. Bir farklılık olması durumunda hangi anda ve hangi çıkışlarda hata olduğunu söylemek de yine programın görevleri arasındandır.

Tezde son olarak, tasarlanan sayısal sistem, tasarımı devam eden yüksek performanslı bir S/A dönüştürücü ile beraber çalıştırılmıştır. Bunun için Cadence'ın AMS simülatörü kullanılmıştır. Bu simülatör; Verilog diliyle tanımlanmış sayısal bir sistemin, analog olarak tasarlanmış bir S/A dönüştürücüyle beraber çalıştırılmasını desteklemektedir. Benzetimde, lojik 0 ve 1 olarak verilen sayısal işaretlerin analoğa dönüştürülmesi ve tersi işlemlerinin gerçekleştirilmesi için, üretim teknolojisi ile uyumlu bir bağlantı kuralları dosyası kullanılmıştır. Elde edilen sonuçlarda, sayısal sistemin, S/A dönüştürücü performansını kötüleştirici yönde etkilemediği gözlenmiştir.

Çalışmanın bütününde işlenen süreç, yüksek hızlı S/A dönüştürücüler için sayısal işaret işleme sistemi tasarımı konusunda kaynak olarak kullanılabilecek zenginlikte anlatılmıştır. Çalışmanın, yüksek hızlı sayısal FIR süzgeçlerin kullanıldığı diğer uygulamalar için de yararlı olacağı düşünülmektedir.

# 1. INTRODUCTION

## 1.1. Ideal DAC Operation and Output

A Digital to Analog Converter (DAC) is an electronic device that converts a digital data sequence to a continuous time signal. From a signal processing perspective, a basic DAC operation can be split into two stages: a transcoding stage, where digital input is converted into an equivalent discrete time signal, and a reconstruction stage where analog signal is generated from discrete time signal [1]. Reconstruction stage also contains two subsections: a sample&hold and an optional reconstruction filter. Complete scheme of an ideal DAC is in Figure 1.1.

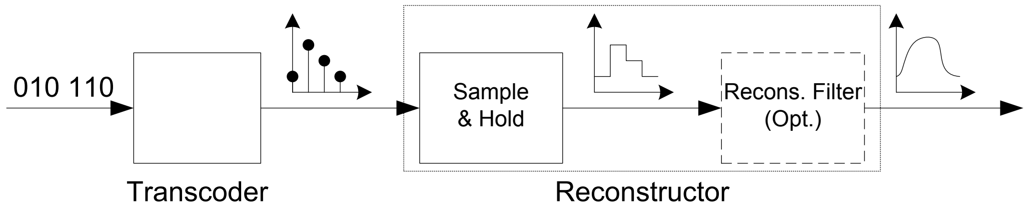
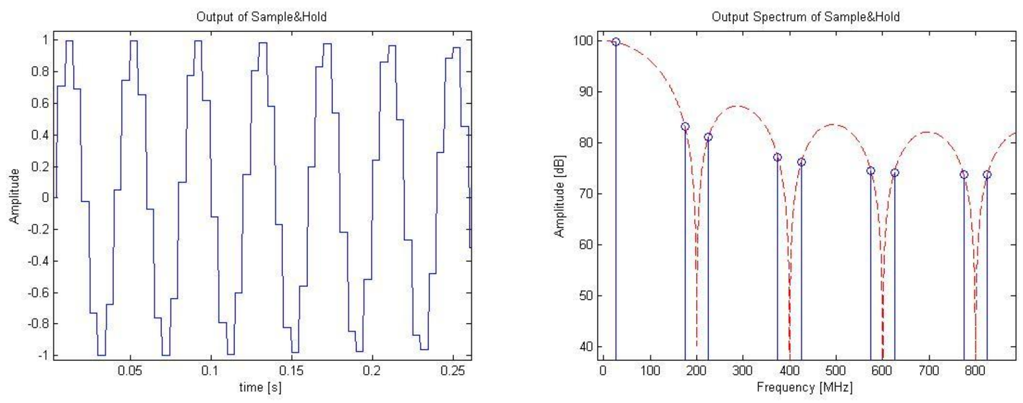


**Figure 1.1 :** Ideal DAC operation.

In the transcoder stage, bit sequences transform to corresponding discrete time values. During the reconstruction phase, sampled discrete time values are changed to continuous time via a sample-and-hold action and the output of the sample&hold can then be smoothed by an optional reconstruction filter. In practice, the reconstruction filter is implemented in analog. Output of an ideal DAC is in Figure 1.2.

Staircase behavior caused by sample&hold in Figure 1.2(a) results high frequency components seen in (b). Moreover, power of the high frequency components fade away like a sinc function, which is the frequency domain equivalent of the time domain sample&hold. It also can be seen from (b) that, higher sampling frequencies move undesired components away from the desired signal and relax the design specifications of the reconstruction filter. That effect will be discussed in Section 1.4.



(a) (b)

**Figure 1.2 :** Ideal DAC output in time and frequency domain. $f_s$=200MHz, $f_c$=25MHz.

## 1.2. DACs in Transmitter Design

DACs are widely used nowadays thanks to the rapid growth of the digital electronics. DACs are used in mp3 players, cell phones, tablet computers, digital TV broadcasts and many other areas. However, DACs addressed in this work are mostly used in cellular base stations, where high speed and signal processing capability are important.

Use of DACs in transmitters of the base stations is sprout with the idea of Software Defined Radio (SDR), which enables modifications on parameters of the communication system, like modulation frequency, modulation type, etc. An example of an ideal SDR system is given in Figure 1.3.

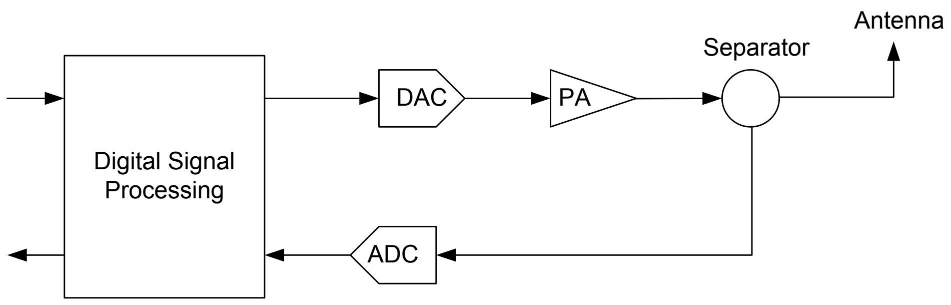


**Figure 1.3 :** Ideal SDR architecture [2].

In Figure 1.3, an ideal SDR block diagram is given with following sub-blocks: a DSP block that manages functionality of the system, a DAC, a Power Amplifier (PA), an ADC and a separator that separates incoming and outgoing signals and antenna.

In practice, ADCs and DACs do not operate fast enough to be directly connected to antenna for modern communication systems but operates on intermediate frequency and signal is modulated to radio frequencies with a final RF modulator. However, those intermediate frequencies are between several hundred megahertzes to gigahertzes and still cutting-edge DACs are used for these applications. Moreover, it is desired to put some of the DSP functionalities into DAC chips and create system on chips (SOCs) for cost efficiency.

## 1.3. Overview of DACs with Digital Interpolation and Modulation

An interpolating and modulating DAC is a DAC, which interpolates and modulates the input data digitally before converting it to an analog signal. Interpolating and modulating DACs offer two different improvements to the conventional DAC: pushed away images from fundamental frequency and ability of a mixer.

When Figure 1.2 (b) and Figure 1.4 are analyzed together, the effect of digital interpolation can be seen. For 4x interpolation, images around 200 MHz sampling frequency are translated as images around 800 MHz sampling frequency. This translation eases design specifications of an external reconstruction filter [3].

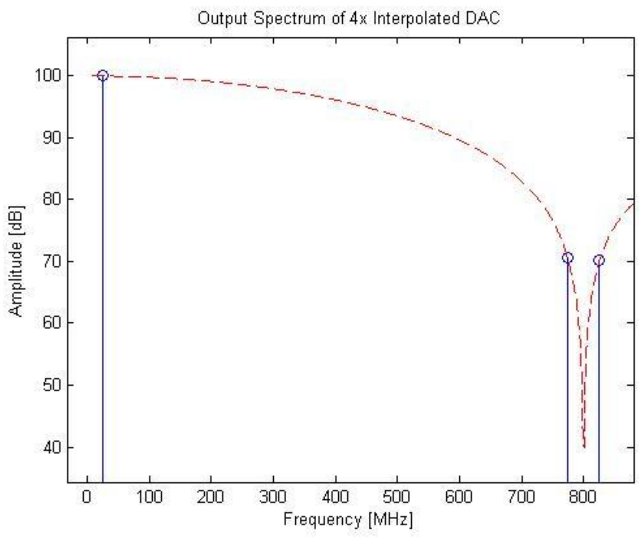
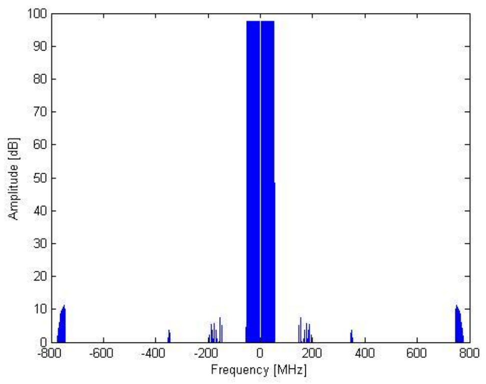
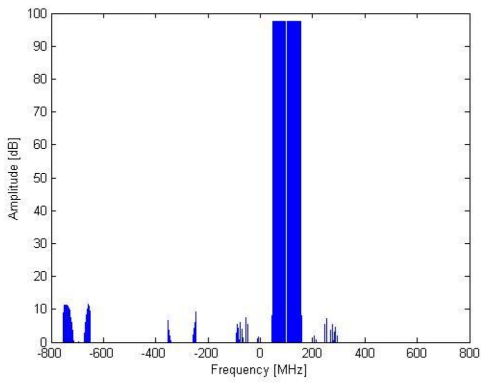


**Figure 1.4 :** 4x Interpolated DAC output spectrum.

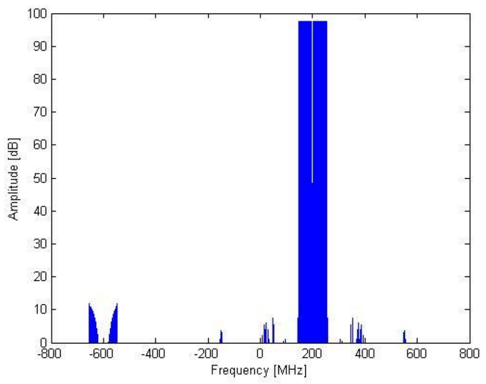
Other improvement of interpolation and modulation DACs is the ability of a mixer. With proper techniques, which will be discussed in later sections, it is possible to shift the input signal anywhere within the nyquist region of the DAC output sample rate. Example spectrum plots are in Figure 1.5 (a to h). By using on-chip digital modulation, users no longer need an analog mixer to modulate the baseband signal into an intermediate frequency (IF).



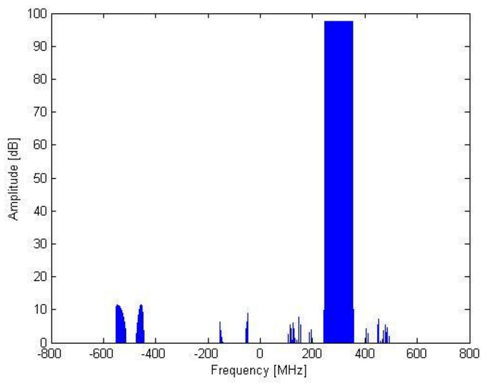
(a)



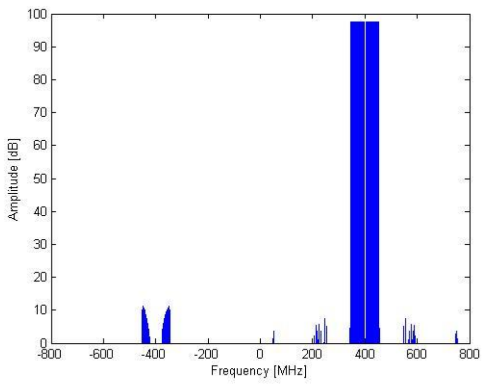
(b)



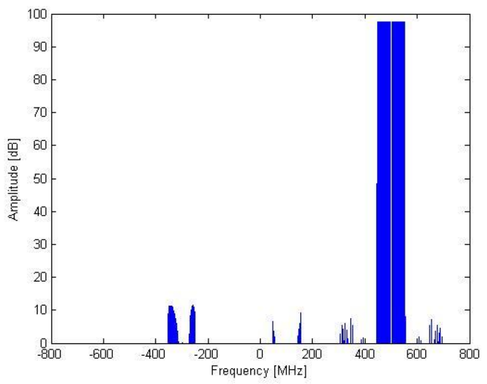
(c)



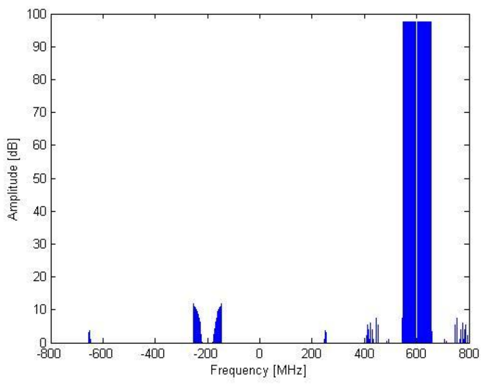
(d)



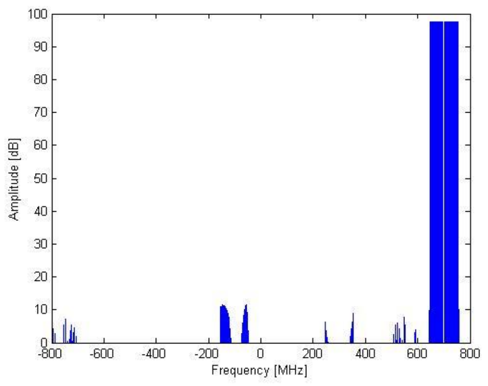
(e)



(f)



(g)



(h)

**Figure 1.5 :** Spectrum plots for digital modulation feature. (a): Baseband, (b): Shifted to 100MHz, (c): Shifted to 200MHz, (d): Shifted to 300MHz, (e): Shifted to 400MHz, (f): Shifted to 500MHz, (g): Shifted to 600MHz, (h): Shifted to 700MHz.

Moreover, an interpolating and modulating DAC allows a lower input data rate, which is easier to generate externally and less likely to generate noise within the system [3].

After mentioning the advantages of interpolation and modulation DACs, it is useful to list some commercial examples. Common products and specifications are listed in Table 1.1.

**Table 1.1 :** Interpolation and modulation DACs in the market.

| Brand | Part | Bit | MSPS | Max. Interpo-lation | SFDR [dB] | Advanced Modulation | Output |
|---|---|---|---|---|---|---|---|
| Analog Devices | AD9122 | 16 | 1230 | 8x | 82 | Yes | Complex |
| | AD9148 | 16 | 1000 | 8x | 80 | Yes | Complex |
| | AD9776/8/9A | 12/ 14/ 16 | 1000 | 8x | 80 | Yes | Complex |
| | AD9786 | 16 | 500 | 8x | 80 | Yes | Real |
| Texas Instruments | DAC3482 | 16 | 1250 | 16x | 82 | No | Complex |
| | DAC5689 | 16 | 800 | 8x | 80 | No | Complex |
| Maxim | MAX5898 | 16 | 500 | 8x | 88 | No | Complex |
| NXP | DAC1408D650 | 14 | 650 | 8x | 80 | No | Complex |

It is seen from Table 1.1 that most DACs have coarse modulation capability and only a few products offer fine modulation capability using a numerically controlled digital oscillator. SFDR values are around 80 dB and they are mostly determined by the output DAC (not by the interpolation filters). A particularly interesting DAC in the market is the ADI's AD9786, which converts complex data to real one with internal Hilbert transformer and outputs only real data.

## 1.4. Interpolation and Modulation: ASIC or FPGA?

Anti-imaging filters are a necessity for most high speed DACs. For a DAC operating at low clock speeds, analog filtering is difficult, because the images are close to the signal passband. In such cases, expensive high order analog filters must be used.

To avoid this problem, digital input sampling rates might be increased to push images away from the signal passband. If sampling rate is increased using FPGA based digital interpolation, on board data transmission speed will increase as a side effect. Thus, both the FPGA and the DAC will require high speed IOs and this will make reliable communication between the two ICs difficult. In such cases, Parallel Low Voltage Differential Signaling (LVDS) or parallel Common Mode Logic (CML) interfaces are needed and system power consumption goes up.

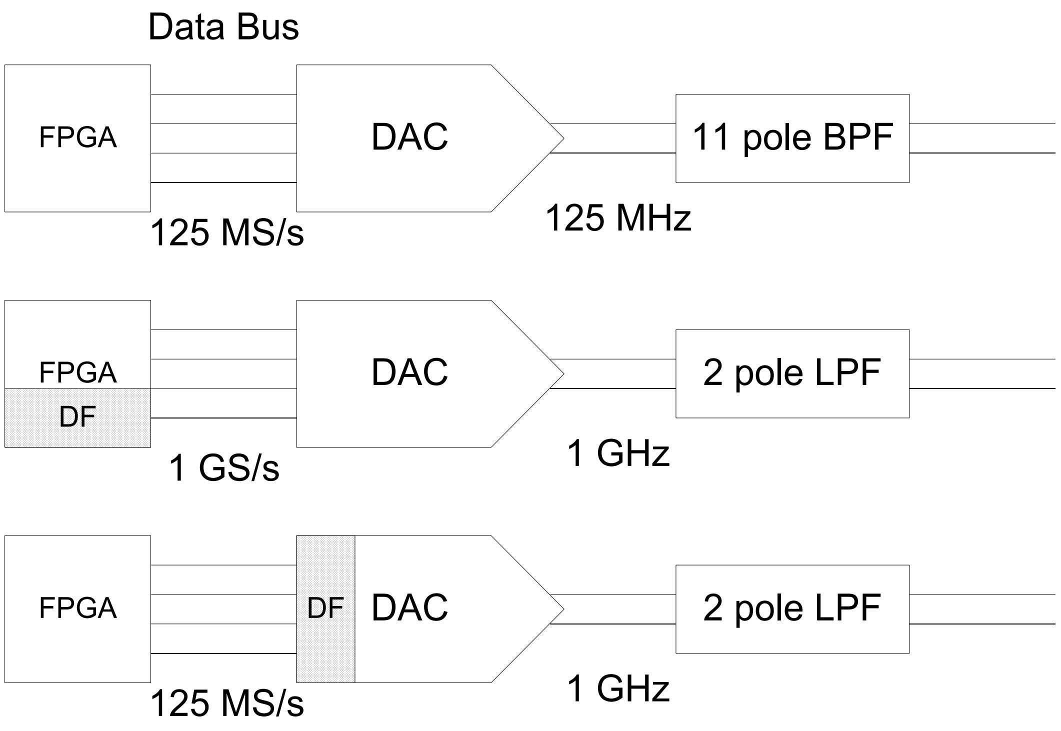


**Figure 1.6 :** Interpolation and modulation DAC architectures.

A good option is keeping DAC input speed lower and output sampling rate higher via on-chip digital interpolation filtering. Interpolation is done on the DAC chip instead of the FPGA. With this method, costly analog filters are avoided and high speed and high power digital interfaces are not needed. Figure 1.5 explains three methods above.

## 1.5. Implementation: System Architecture & Computer Arithmetic

Details of the architecture of the existing high speed DACs are still not clear because of the confidentiality. However, block diagrams given in datasheets of the products and recent papers or patents can give clues about the architectural details.

As a starting point, all examined DAC datasheets, that contain interpolation and modulation capability, have FIR filters in their signal path [4-13]. It can be said that filter architecture is FIR not IIR. That also makes sense as FIR filters offer linear phase. Other information that is given is; SFDR performance, filter coefficients, operation modes and even delay of the digital filters in clock pulses (like 32 clocks). With all these specifications, necessary data for a semi-custom digital design becomes available. On the other hand, if full-custom design methodology was assumed, some of the information would be missing. For example, in a DAC datasheet [13], a minimum clock speed specification is provided, which may imply that dynamic flip-flops might have been used in that design.

It is known that a typical digital interpolation system with modulation used in high-speed DACs contains FIR filters, is designed in semi-custom fashion (which leads to static flip-flops, standard library gates etc.) and is pipelined in order to operate as fast as possible for that process. SFDR requirement of the filters is also known to be 5 or 10 dB larger than DAC’s SFDR.

# 2. THEORY OF OPERATION

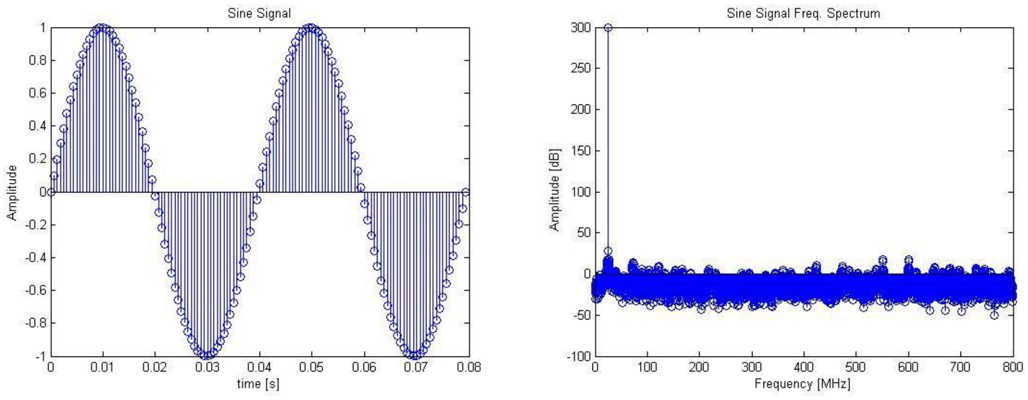
In this work, two main operations are supported within the DAC: interpolation and modulation. In this chapter, mathematical background of these operations and implementation methods are explained.

## 2.1. Process of Digital Filtering

In Figure 1.2 of the previous chapter, time and frequency domain representations of DAC output was given. In this section, process of digital to analog conversion and its effects to output signal will be explained step by step.

### 2.1.1. Images from sampling

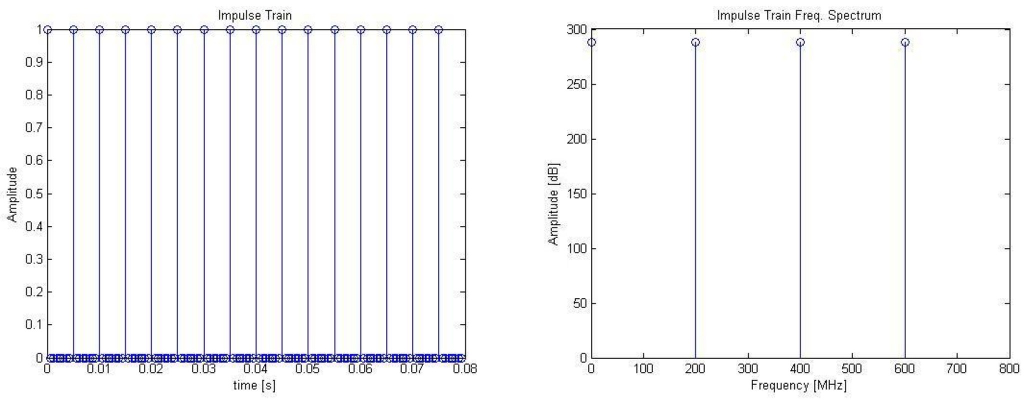
Sampling is the first source of non-idealities, in other words, images. To show the effect of sampling, an oversampled signal seen in Figure 2.1 can be considered as an analog signal. This is true when $f_s \rightarrow \infty$.



(a) (b)

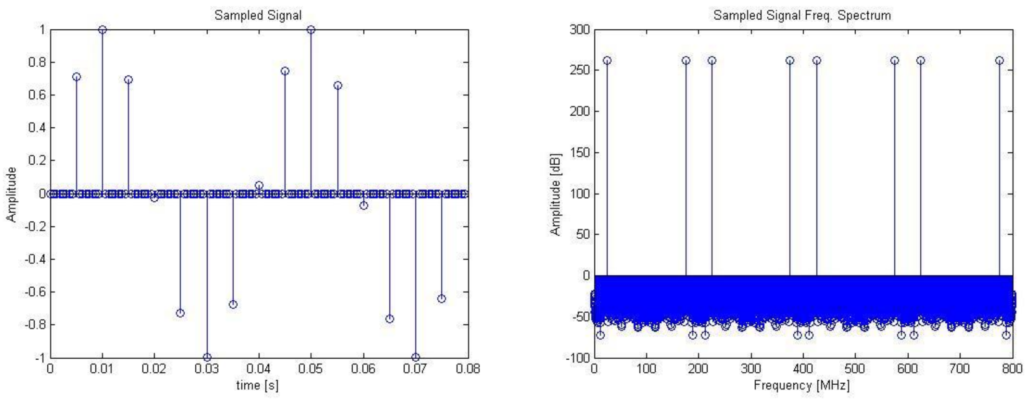
**Figure 2.1 :** Time (a) and frequency (b) plots of an analog sine wave in discrete time domain using the fast Fourier transform (FFT).

Sampling is equivalent to multiplying with an impulse train in time domain. Impulse train given in Figure 2.2 (a) is used for sampling the sine wave in Figure 2.1 (a). Fourier transform of an impulse train another impulse train as seen in Figure 2.2 (b). Multiplying with an impulse train in time domain is equivalent to convolving with another impulse train in the frequency domain. The effects of multiplying with an impulse train in time domain can be seen in Figure 2.3 (a), which generates Figure 2.3 (b) in frequency domain.



(a) (b)

**Figure 2.2 :** Time (a) and frequency (b) representations of an impulse train.



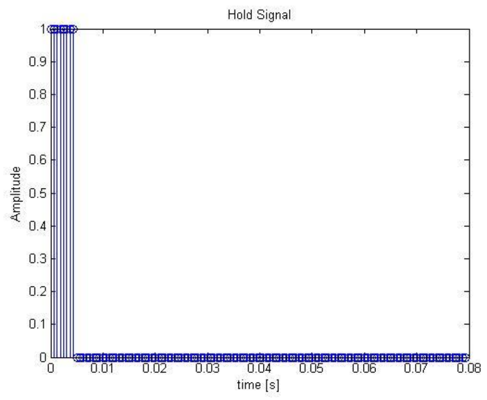
(a) (b)

**Figure 2.3 :** Time (a) and frequency (b) representation of sampled signal.

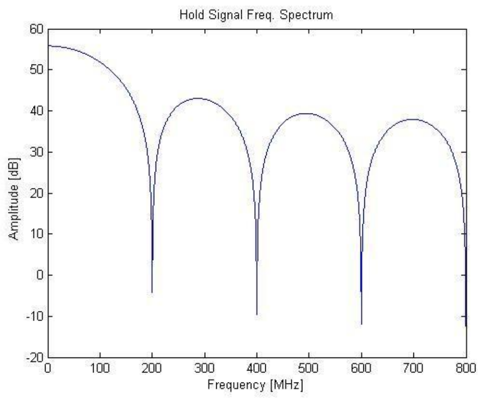
If Figure 2.3 is analyzed, it can be seen that in frequency domain, input signal replicates itself at every integer multiples of the sampling frequency. Time domain sampled signal only takes non-zero values at sampled times as expected. In frequency domain, it can also be said that frequency domain is corrupted by unwanted replicas of the input signal bandwidth.

### 2.1.2. Effect of hold

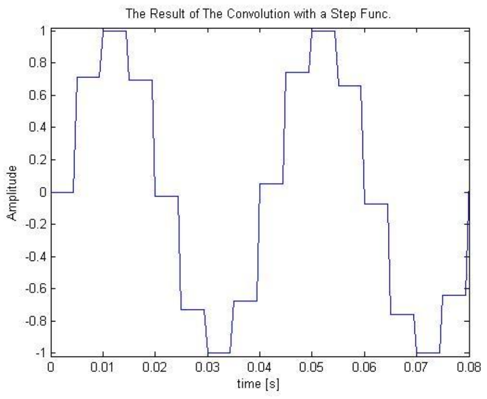
Another effect that is observed from Figure 1.2 was the effect of hold operation which will be added to model now. Hold operation makes the sampled value unchanged until next sampling time and creates a staircase behavior to time domain. In Figure 2.4 (a), hold signal that convolves with the input signal in time domain is shown. Its Fourier transform, which is a $\frac{\sin x}{x}$ function, is shown in Figure 2.4 (b). As seen in Figure 2.4, hold signal is a simple step signal that Fourier transforms into a sinc function. The results of the convolution can be seen in Figure 2.4 (c) and Figure 2.4 (d), in time and frequency domains respectively.



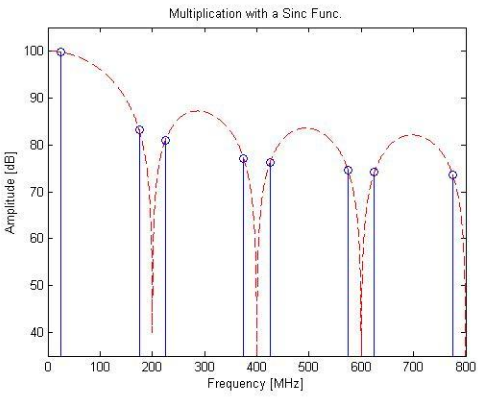
(a)



(b)



(c)



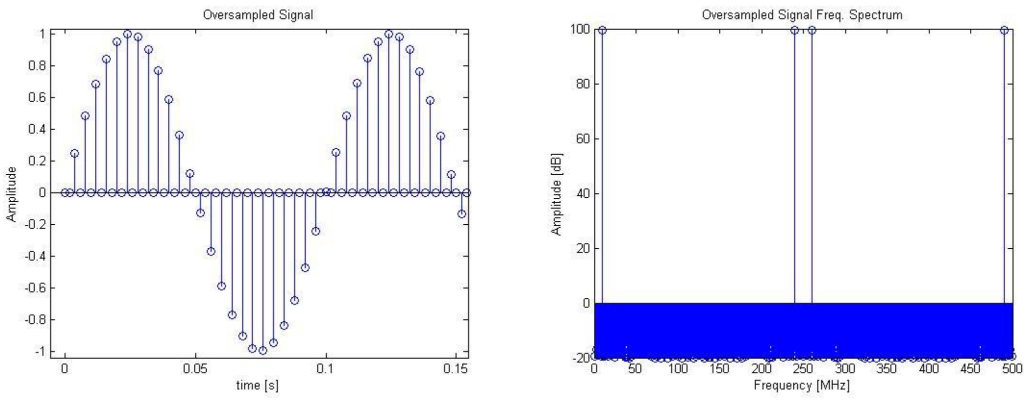
(d)

**Figure 2.4 :** Time (a) and frequency (b) representation of hold signal. (c): The result of the convolution with a step function. (d): Multiplication with a sinc function.

### 2.1.3. Effect of digital oversampling with zero filling

In addition to the sampling issue in analog domain, digital oversampling also causes images in the frequency domain. Plots of a 2x oversampled sine wave with zero filling are in Figure 2.5.

That effect may look similar to effect discussed in the analog domain; however, it actually occurs entirely in digital domain.

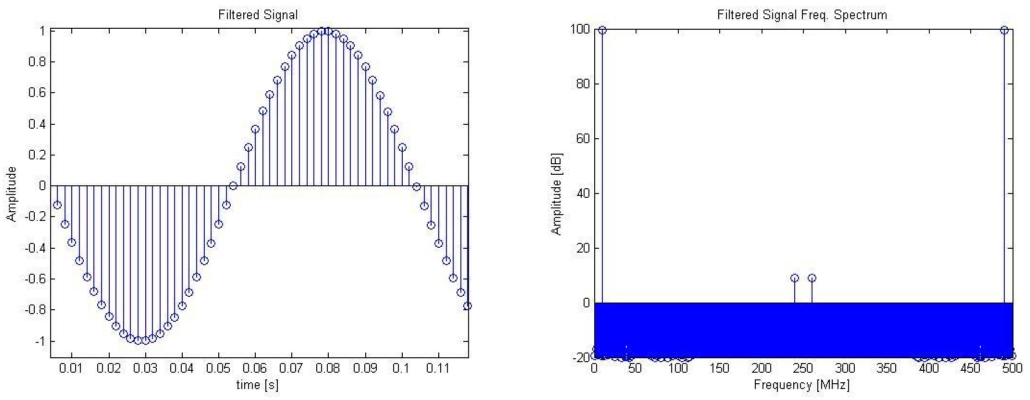


(a) (b)

**Figure 2.5 :** Time (a) and frequency (b) representation of 2x oversampled signal.

### 2.1.4. Effect of digital filtering after zero filling

It is easier to suppress digital images with a digital filter, as implementation of the digital filters is more robust than their analog counterparts. Thus, digital filtering must be performed after an oversampling operation, to reduce images. Filtered data is given in Figure 2.6.



(a) (b)

**Figure 2.6 :** Time (a) and frequency (b) representation of the filtered signal.

### 2.1.5. Digital filtering

As explained in Section 1.3, with digital filtering, unwanted digital images at lower frequencies are suppressed and analog images are translated to higher frequencies, which relaxes the specifications of the analog reconstruction filter. This effect is also shown in Figure 2.5-2.6. It is shown in Figure 2.2-2.3 that images occur at integer multiples of sampling frequency. In order to push them away, sampling frequency should be increased before digital to analog conversion. This is the reason that makes oversampled digital filtering useful.

## 2.2. Interpolating Filter Design

Interpolation is the combined process of up-sampling, zero padding and inserting newly calculated points between every two data points of the original digital input signal. With interpolation, sampling rate of the signal increases by the factor of interpolation rate. Interpolation operation is performed via an interpolation filter.

### 2.2.1. Half band filters as a polyphase interpolation filter

Half Band Filter is a special case of polyphase filters which can be used as an interpolation filter. Using a polyphase filter has many benefits like working with lower clock frequencies, using less multipliers and ability to perform frequency shifting operations (modulation).

#### 2.2.1.1. Basics of a polyphase interpolation filter [14]

Interpolation process can be shown as an up-sampler, followed by a low pass filter (LPF), as illustrated in Figure 2.7.

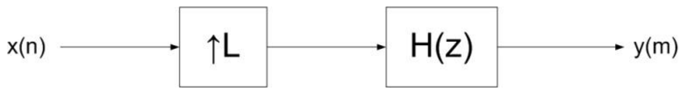


**Figure 2.7 :** Interpolation process.

In Figure 2.7, “↑L” means up-sample by a factor of L and H(z) represents LPF that is used for image rejection. Up-sampling increases the sampling rate by L and then, zero-pads by L-1 zeroes.

Although the structure above is functionally correct, it is not effective due to the large number of zeros at the filter input. At this point, polyphase interpolation becomes helpful.

A digital LPF can be shown with **(2.1)**.

$$H(z) = \sum_{n} h(n) z^{-n} \qquad \textbf{(2.1)}$$

In **(2.1)**, filter coefficients are h(n). That also equals to the representation in **(2.2)**.

$$H(z) = \sum_{k} \left( \sum_{p=0}^{L-1} h(kL+p) z^{-(kL+p)} \right) \qquad \textbf{(2.2)}$$

In **(2.2)**, $k = \left\lfloor \frac{n}{L} \right\rfloor$ (round towards zero) and $p = n\%L$ (n mod L). **(2.2)** states that, coefficients of a single filter can be grouped by L. If a delay element is taken out of the inner equation and the indices of summations are switched, **(2.3)** can be obtained.

$$H(z) = \sum_{p=0}^{L-1} \left( \left( \sum_{k} h_p(k) z^{-kL} \right) z^{-p} \right) \tag{2.3}$$

In **(2.3)**, $h_p(k) = h(kL + p)$. There arrangement means that L filters can be defined which is functionally equal to the main filter in **(2.1)**. In the end, z-domain representation of the final system is in **(2.4)** and Figure 2.8.

$$H(z) = \sum_{p=0}^{L-1} H_p(z^L) z^{-p} \tag{2.4}$$

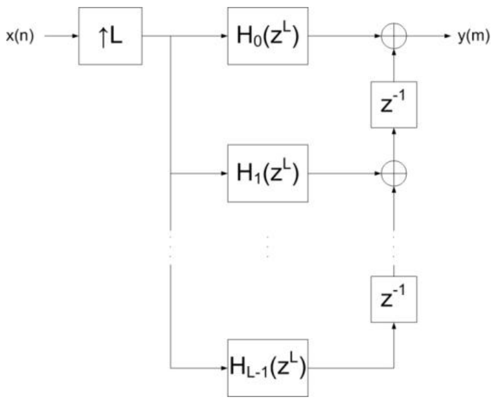


**Figure 2.8 :** Another representation of filter H(z).

In Figure 2.8, L filters are shown, which is functionally equal to the filter in Figure 2.7. At this point, up-sampler block can be put after every sub-filter without changing the functionality of the system. This is the noble identity for interpolation [14]. Resulting diagram is in Figure 2.9.

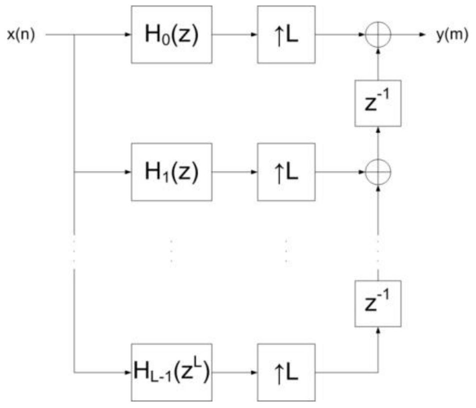


**Figure 2.9 :** Polyphase interpolation filter.

#### 2.2.1.2. Special case: half band filter (L=2) [14]

The Half Band Filter is a special case of polyphase filters with L = 2. It is both simple owing to less number of sub-filters and advantageous being a polyphase filter. Block diagram of Half Band Interpolation Filter is in Figure 2.10.

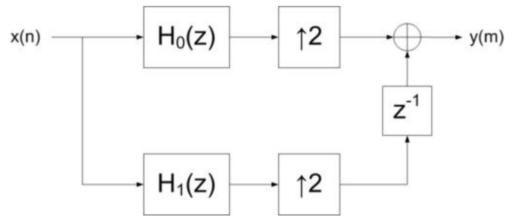
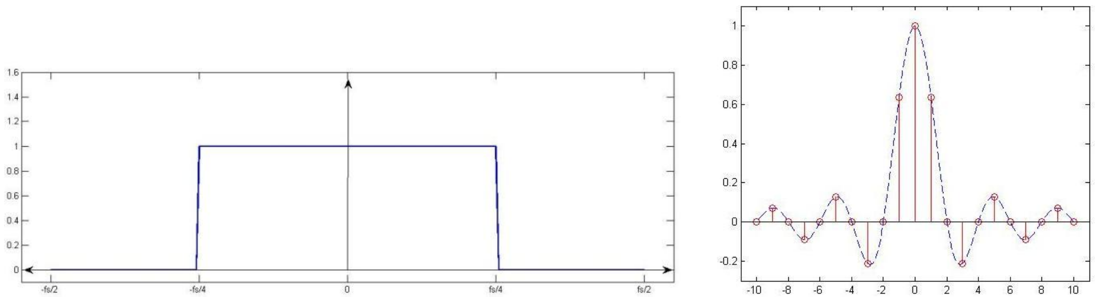


**Figure 2.10 :** Half Band Interpolation Filter.

To understand the structure of H(z)s, it is important to have a look at the impulse and frequency response of half band low pass filter given in Figure 2.11.



(a) (b)

**Figure 2.11 :** Frequency (a) and impulse (b) response of Half Band LPF.

As shown in Figure 2.11, almost half of the coefficients of Half Band Filter are zero. Moreover, zero coefficients are only even indexed coefficients which makes them easy to eliminate. Details and other advantages of Half Band Filters are in Section 2.1.2.

### 2.2.2. Advantages of half band interpolation filters

Main advantages of Half Band Interpolation Filters can be grouped under three headings: fewer coefficients, working at half output data rate and ease to transform.

#### 2.2.2.1. Fewer coefficients (and multipliers)

According to **(2.4)**, a Half Band Filter can be written as **(2.5)** below.

$$H(z) = H_0(z^2) + H_1(z^2)z^{-1} \quad \textbf{(2.5)}$$

And impulse response representation is in **(2.6)**.

$$H(z) = \sum_n h(2n)z^{-2n} + h(2n+1)z^{-(2n+1)} \qquad \textbf{(2.6)}$$

As seen from Figure 2.12, all except one even indexed coefficients of Half Band Filter are zero. So, resulting equation is in **(2.7)**.

$$H(z) = h(0) + \sum_n h(2n+1)z^{-(2n+1)} \qquad \textbf{(2.7)}$$

As indicated from previous equations, zero coefficients of the Half Band Filter can be reduced. Moreover, because implementing zero-coefficients is unnecessary, almost half of the multipliers are also removed. That feature is illustrated in Figure 2.12.

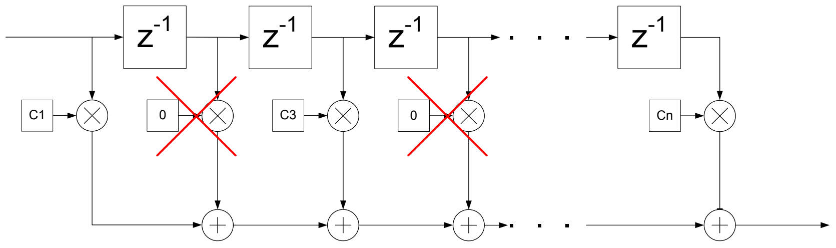


**Figure 2.12 :** Saved multipliers in HBFs due to zero-coefficients.

### 2.2.2.2. Working at half output data rate [14]

Input and output data rates of a conventional filter are equal as expected. On the other hand, an interpolation filter should increase the data rate by the interpolation factor L. At this point, architecture of the interpolation filter gains importance because of the clock speed of the filter. It is known that interpolation filters are mostly used in high speed DACs [4-13] which demands output data rates up to 1.25 GSPS[11]. While designing such a high speed digital circuit, it is desired to use lower speed clocks to reduce power consumption. In short, it would be a good idea to reduce clock speed of the filter without changing functionality and not worsen area and power characteristics. In Figure 2.13, Half Band Filter block diagrams are given before and after using noble identity.

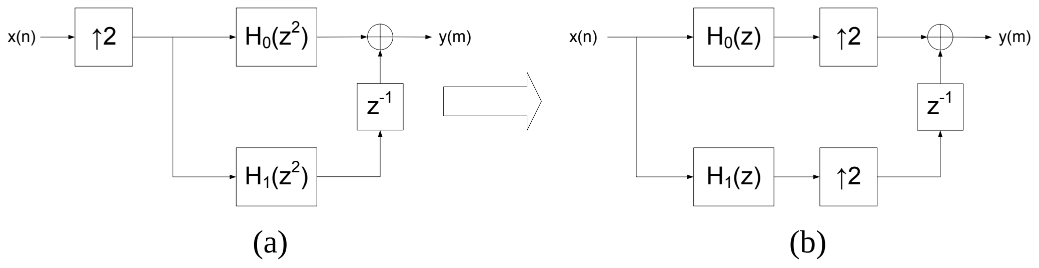


**Figure 2.13 :** HBF before (a) and after (b) applying noble identity.

In Figure 2.13 (a), it is seen that filtering is performed after up-sampling which means filters $H_1$ and $H_2$ should work at twice the data rate of input x(n) because of oversampling. However, after applying noble identity, $H_1$ and $H_2$ now come before the upsampler which makes them work at the same speed of input data rate. That feature makes HBFs not only smaller because of less multipliers, but also more power efficient due to being clocked at lower speeds.

### 2.2.2.3. Easy to transform from low-pass to high-pass [14]

In section 2.1.1.1, it is stated that for interpolation process, a LPF is required to suppress unwanted images. But in some cases, those images are needed to be kept, as they are modulated versions of the input signal. Although the modulation topic is explained in section 2.2, it is important to note now that changing pass-band of the filter may be desired. In this case, designers prefer configurable filters without a big area cost for implementation.

HBFs are one of the filters whose pass-band can be changed without additional cost. In Figure 2.14, high-pass version of the HBF can be seen. To make a HPF from its LP equivalent, LP impulse response is modulated by a cosine with half sample rate frequency. It is obvious that frequency domain equivalent of this operation pushes spectral center where pass-band is, to half the sample rate. Transform is in **(2.8)**.

$$h(n)_{HPF} = h(n)_{LPF} * \cos(\pi n) \qquad \textbf{(2.8)}$$

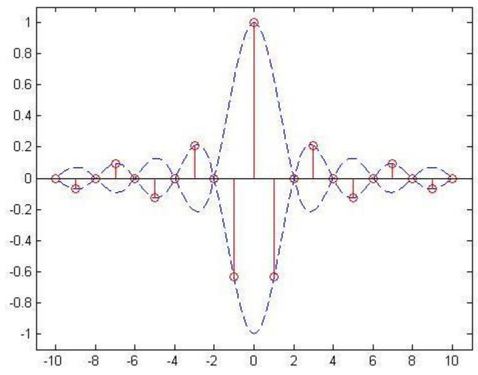


**Figure 2.14 :** Impulse response of Half Band HPF.

In **(2.6)** and **(2.7)**, it is given that all coefficients of Half Band Filter are odd indexed except the one in center. Besides, it can be said by looking Figure 2.10 that, even and odd indexed coefficients are implemented in separate filters $H_0$ and $H_1$ respectively. Based on these explanations, HP representation of HBF is as follows in Figure 2.15.

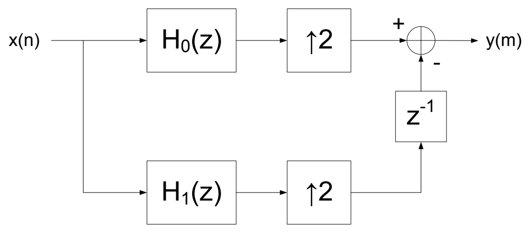


**Figure 2.15 :** Half Band High Pass Interpolation Filter**.**

As seen from Figure 2.15, only modification in block diagram is the subtractor. LP to HP transform is just an example of easy transformability of the HBFs. By using similar approaches, several complex BPFs can be realized without much area and power cost. The necessity of BPF transformation is explained in Section 2.3.4.

## 2.3. Coarse Modulator Design

Coarse modulation of input signal is commonly done by an analog mixer circuit after the DAC in an RF system [4-13]. On the other hand, due to recent trend about using less number of ICs and avoiding problems caused by analog mixers like jitter and other non-idealities, digital modulation before or within DAC circuits gains importance. That modulation could be a fine or a coarse modulation, depending on mixing requirements.

In order to implement a fine modulation, a much higher frequency resolution is required than a coarse modulation. Fine modulation is generally used to put data to a specific frequency channel. Although fine modulation is often critical for a communication system, the baseband digital circuits driving the DAC chip can easily implement it. The used method is performing a fine modulation before the DAC chip and coarse modulation within the DAC chip. For communications in multi-GHz frequencies, an analog modulator after DAC chip is necessary.

Coarse modulation however has only a few major frequency steps to shift the input data. Details of this operation and its implementation are explained in this section.

### 2.3.1. Standard digital modulation

It is well-known that modulation is simply performed by multiplying the input signal with a cosine with desired frequency. In order to apply that, a multiplier and a cosine generator (oscillator) circuit are required.

### 2.3.2. $2^n$ Ratios of sampling frequency

In a modulation system with a multiplier and oscillator, more complex block is the oscillator which calculates amplitude values of the cosine function. If characteristics of a cosine function like zero crossings and symmetry are used wisely, a numerical oscillator can be turned into a simple multiplexer with several constant values at the input. To take those advantages, frequency values of coarse modulator is chosen as $2^n$ ratios of the sampling frequency. Several examples are in Table 2.1.

**Table 2.1 :** Cosine values to multiply for several frequencies.

| Frequency | Values |
|---|---|
| $f_S/2$ | -1 1 |
| $f_S/4$ | -1 0 1 0 |
| $f_S/8$ | -1 $-\sqrt{2}/2$ 0 $\sqrt{2}/2$ 1 $\sqrt{2}/2$ 0 $-\sqrt{2}/2$ |

### 2.3.3. Eliminating multipliers and coefficients

At frequencies Interpolation and Modulation DACs operate, it is both area and power consuming to use a multiplier at the end of the interpolation filter where modulation is carried out. Thus, elimination the output multiplier is desired. In Section 2.1.2.3, performing modulation by $f_s/2$ is done by using a subtractor, which performs multiplication with -1s. As this only works for $f_s/2$ modulation, other coarse modulation frequencies are created by cascading several interpolation and modulation filter blocks with ascending sampling frequencies and LP, BP or HP modes.

### 2.3.4. Hilbert transformer [14]

Hilbert Transformer is a complex filter which passes positive frequencies and suppresses negative frequencies of the input signal. It is a positive shifted version of a LP Half Band Filter. Frequency response of a Hilbert transformer is in Figure 2.16.

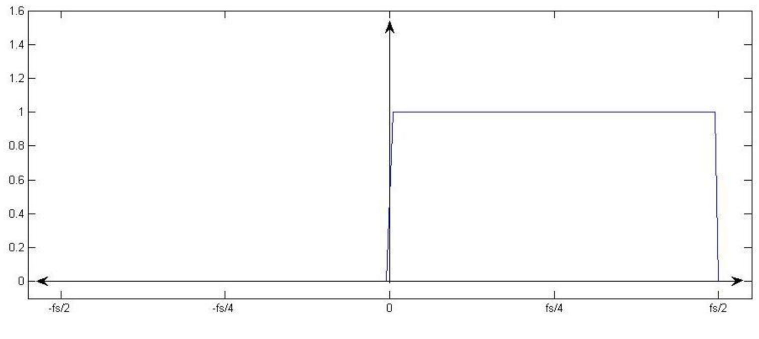


**Figure 2.16 :** Frequency response of Hilbert Transformer.

Operation of Hilbert Transformer is given in **(2.9)**

$$h(n)_{HT} = h(n)_{LPF} \times e^{jn\pi/2} \tag{2.9}$$

If $e^{jn\pi/2}$ is expanded with a sine and cosine function, **(2.10)** is obtained.

$$h(n)_{HT} = h(n)_{LPF} \times \cos(n\pi/2) + j\sin(n\pi/2) \tag{2.10}$$

Cosine function in **(2.10)** gives 1 and -1 when n is even and 0 when n is odd. However for odd n's, $h(n)_{LPF}$ gives non-zero and for even n's it gives zero except n=0 as it is a Half Band Filter. Eventually, for real part of the filter $h(n)_{HT}$, only one non-zero coefficient is present at n=0.

In the meantime, sine function gives -1 and 1 when n is odd and 0 otherwise. When multiplied by filter coefficients, sine function makes left half of the coefficients minus signed and right half positive signed. Resulting impulse responses are in Figure 2.17.

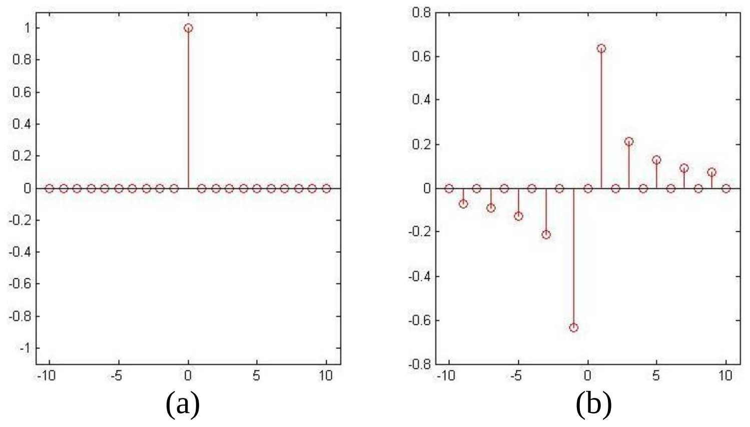


**Figure 2.17 :** Real (a) and imaginary (b) impulse responses of Hilbert Transformer.

As seen from Figure 2.17, only signs of coefficients are changed when a LPF is transformed into a Hilbert Transformer. That operation can easily be implemented by using subtractors instead of adders, which come after multipliers. Unlike LP and HP counterparts, Hilbert Transformer gives complex output (or two outputs: real and imaginary). Final visualization of Hilbert Transformer is in Figure 2.18.

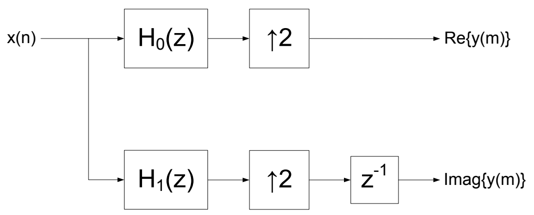


**Figure 2.18 :** Hilbert Transformer.

### 2.3.5. Limitations

As described in previous sections, an efficient coarse modulation can be performed via HBFs with small modifications. But due to the nature of the operation, modulation step size is limited to $f_s/2$. If more than one sampling rates and filter-interpolator couples are used, step size can be $f_s/2k$ where k is number of filter-interpolator couples. For example for k = 3, it is possible to shift base frequency of input data to following ratios of sampling frequency $f_s$ of the output: [-3/8 -2/8 -1/8 0 1/8 2/8 3/8 4/8].

When finer modulation is desired it is necessary to use a multiplier and oscillator circuits at the end of the interpolation filters which brings extra cost for higher operating frequencies.

## 2.4. Multi-Stage Interpolation with Half-Band Filters

So far, single-stage interpolation with HBFs, which doubles the sampling frequency was described. If more than 2x interpolation is desired for better improvement in image rejection or modulation with larger band is aimed, cascaded HBFs are used.

As every HBF stage doubles sampling frequency and data rate, using n-stage HBFs gives 2nx interpolation and pushes images 2nx away.

### 2.4.1. Baseband

Using LP modes of the half band filters in cascade form results in suppressing all the images and only the baseband frequencies are allowed to pass. This may be referred as baseband interpolation. A three-stage interpolation (8x) constructed by cascaded HBFs is represented in Figure 2.19.

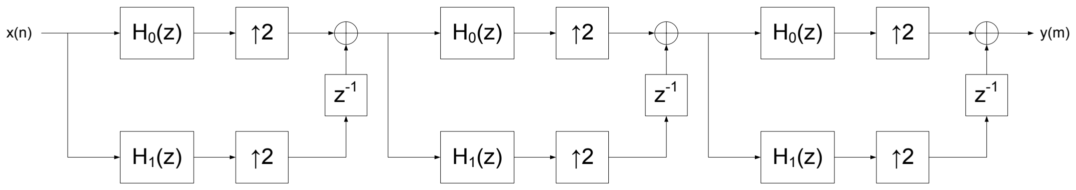


**Figure 2.19 :** Three-stage interpolation with cascaded HBFs.

In the interpolation system given in Figure 2.19, output y(m) is the 8x interpolated version of the input x(n). As all three HBFs work in LP mode, no modulation was performed.

## 2.4.2. Whole band

In Sections 2.2.2.3 and 2.3.4, it is mentioned that HBFs can easily be transformed from LP to HP and BP operation which may called as operation mode of the filter. By doing the transformations, sampling related images can be used for modulation purposes, as they are the modulated versions of the input data.

In order to demonstrate using different images for modulation purposes, let's examine an interpolation and modulation system with three-stage HBF cascades. Also let the operation modes of the filters be LP, HP and BP respectively. Then, the inputs and the outputs of the filters are given in Figure 2.20.

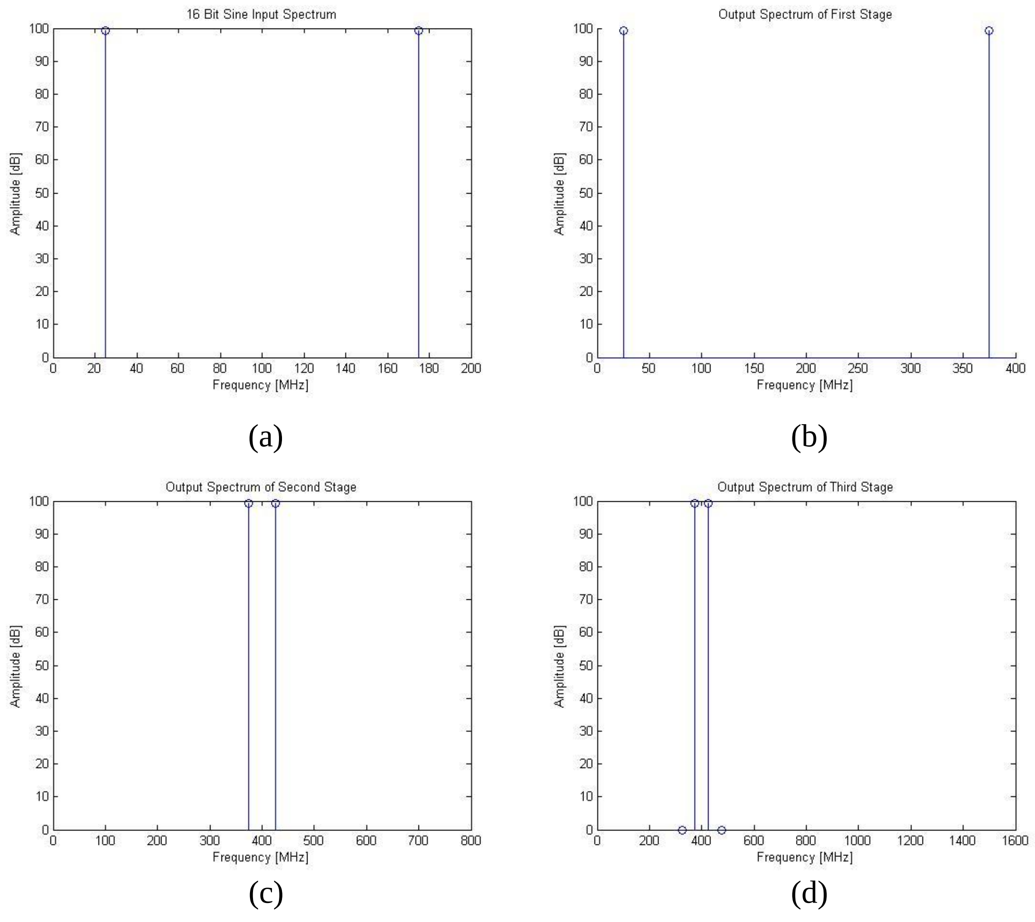


**Figure 2.20 :** Spectrum plots of the signal at filter inputs and outputs.

In Figure 2.20 (a), it is seen that 16 bit sine signal with 25 MHz is applied to system input at $f_S$ = 200 MHz. As first stage acts as a LPF, output of the first stage is similar to input with an increase in sampling rate (Figure 2.20 (b)). At second stage, HBF performs a HP behavior which passes frequencies around 400 MHz and suppresses the others, still increasing the sampling rate (Figure 2.20 (c)). Finally, third stage operates in Hilbert Transform mode, with a further increase in sampling rate. Thus, third stage passes positive frequency band (0-800 MHz) and suppresses the rest (800-1600 MHz) (Figure 2.20 (d)). In the end, both 8x interpolation and modulation to 400 MHz is performed with 3-stage HBFs.

Many other modulation steps are possible to achieve with the same approach, just by changing modes of the HBFs. Details of every mode and outputs will be given in Section 3.1.3.

# 3. IMPLEMENTATION OF THE ARCHITECTURE WITH 85 DB SFDR

This section covers the implementation of a standard architecture of Half Band interpolation filters with modulation capability. Standard architecture refers to a non-optimized architecture which is created by examining the datasheets of the existing DACs with interpolation and modulation capability, and theories published in the literature. Design process is examined under four headings: Modeling, HDL coding, synthesis, layout and simulation results.

## 3.1. Behavioral Modeling: MATLAB Models

This section explains the purpose of behavioral modeling and detailed description of the models.

### 3.1.1. Importance of modeling: why it is important?

Modeling is the first step of an implementation process. It gives a brief idea of what is to be expected. During behavioral modeling, ideal blocks are used to understand functionality and needs of the system better. Even after real blocks and physical effects are included to system, behavioral models are still used for comparison and detection of possible implementation errors.

Another advantage of modeling is visualizing the system and increasing comprehensibility.

### 3.1.2. Model description

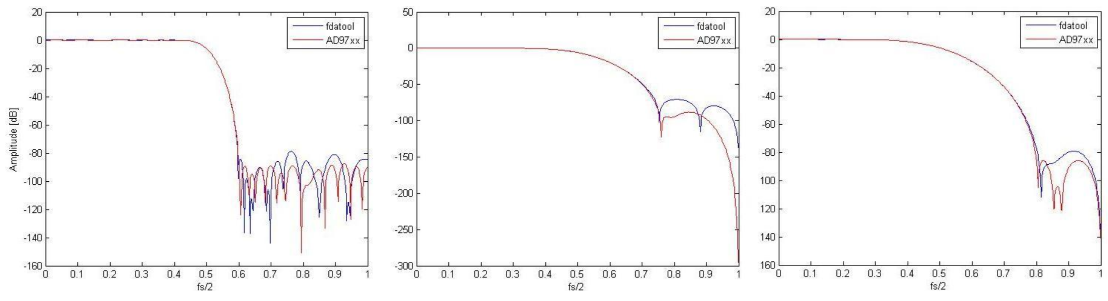
For behavioral modeling of the system, MATLAB is used first. Reasons of choosing MATLAB as a behavioral modeling and simulation medium are its libraries, ease of use, reliability and the convenience of co-simulation with Simulink.

Modeling filter functionality with Simulink models was another decision that was made. With its drag-and-drop interface, Simulink models offer visual representation of the system, which makes it easy to understand and convert to HDL models. It was seen that HDL design workload reduced dramatically as visual models of the system became available.

During the modeling process, MATLAB models of interpolation and modulation half band filters were simulated and the output data is analyzed to test functionality and performance. During code development, many MATLAB models are created, which start from basic models that are implemented and understood easily, to more complex models that are closer to reality. It is unnecessary to explain every model line by line. Instead, common parts of the models are explained in this section.

### 3.1.2.1. Obtaining filter coefficients

The most important part of designing a filter is calculating the filter coefficients. To make that process easier, MATLAB offers a filter design and analysis tool named ‘fdatool’. Fdatool is used to obtain Half Band Filter coefficients. Moreover, the differences between the filters obtained from the fdatool and the filters published in AD977x DAC datasheet are found. Frequency responses of the resulted filters are given in Figure 3.1.



(a) (b) (c)

**Figure 3.1 :** Comparison of fdatool generated filters and AD977x filters.

In order to generate above filters by fdatool, data from a previous work [15] was used. In that work, filter orders are chosen as 54, 22, and 14 respectively. Moreover, normalized pass-band frequencies are chosen as 0.4, 0.23 and 0.195 respectively. It is stated in the work [15] that using those numbers generated good performing filters.

As seen from Figure 3.1, there are significant differences between frequency responses of fdatool and AD977x results. SFDR values of the obtained filters are in Table 3.1.

**Table 3.1 :** SFDR differences of filters

| Filter | Source | SFDR [dB] | Delta [dB] |
|---|---|---|---|
| F1 | fdatool | 79 | 9 |
|  | AD977x | 88 |  |
| F2 | fdatool | 71 | 17 |
|  | AD977x | 88 |  |
| F3 | fdatool | 79 | 7 |
|  | AD977x | 86 |  |

Table 3.1, reveals that fdatool may not provide best result. We speculate that fdatool cannot model truncation of the coefficients during calculations but it truncates the coefficients after calculations. That may result in different performance. In order to avoid such problems, methods like "quantization using compensating zeros"[16] might be used. These methods take the effect of the truncated coefficients into account and better performance is obtained with respect to the truncating after the calculation.

During implementation, coefficients of AD977x DAC are used to get better performance, as a starting point for our first design.

### 3.1.2.2. Input signal generation

First parts of the MATLAB models are input signal generation. To perform FFT operation correctly, input signal frequencies are determined according to **(3.1)**.

$$f_{in} = \frac{x}{n_{FFT}} f_s \tag{3.1}$$

In **(3.1)**, $f_{in}$ is the input signal frequency, $n_{FFT}$ is number of points per FFT operation, $f_s$ is sampling frequency and x is the FFT bin number.

If this rule is followed, resulting FFT only has the desired frequency components, and spectral leakage is avoided. By changing $n_{FFT}$ and x, input signal frequency can be changed or multi-tone signals can be given as an input to the model.

Next simulation parameter is "n_of_bins" which is a misnomer that represents the multi-tone signal bandwidth. With that option, multi-tone signals can be generated as the input to the filter.

Finally, variables “hb1_mod”, “hb2_mod” and “hb3_mod” can be assigned to set desired filter modes. Variable “premod” is used to shift the input data by $f_s/2$. Details of filter modes and pre-modulation feature will be explained in Section 3.1.3.

#### 3.1.2.3. Filtering

Filtering is one of the main operations of the system. In order to analyze filtering, several different methods were used. Those methods can be grouped under two headings: filtering with “filter” function and filtering using Simulink blocks.

**With “filter” function**

First method of filtering is MATLAB’s “filter” function which takes data to be filtered and filter coefficients as input and outputs the filtered data. In that type of filtering, the method that is described below was used.

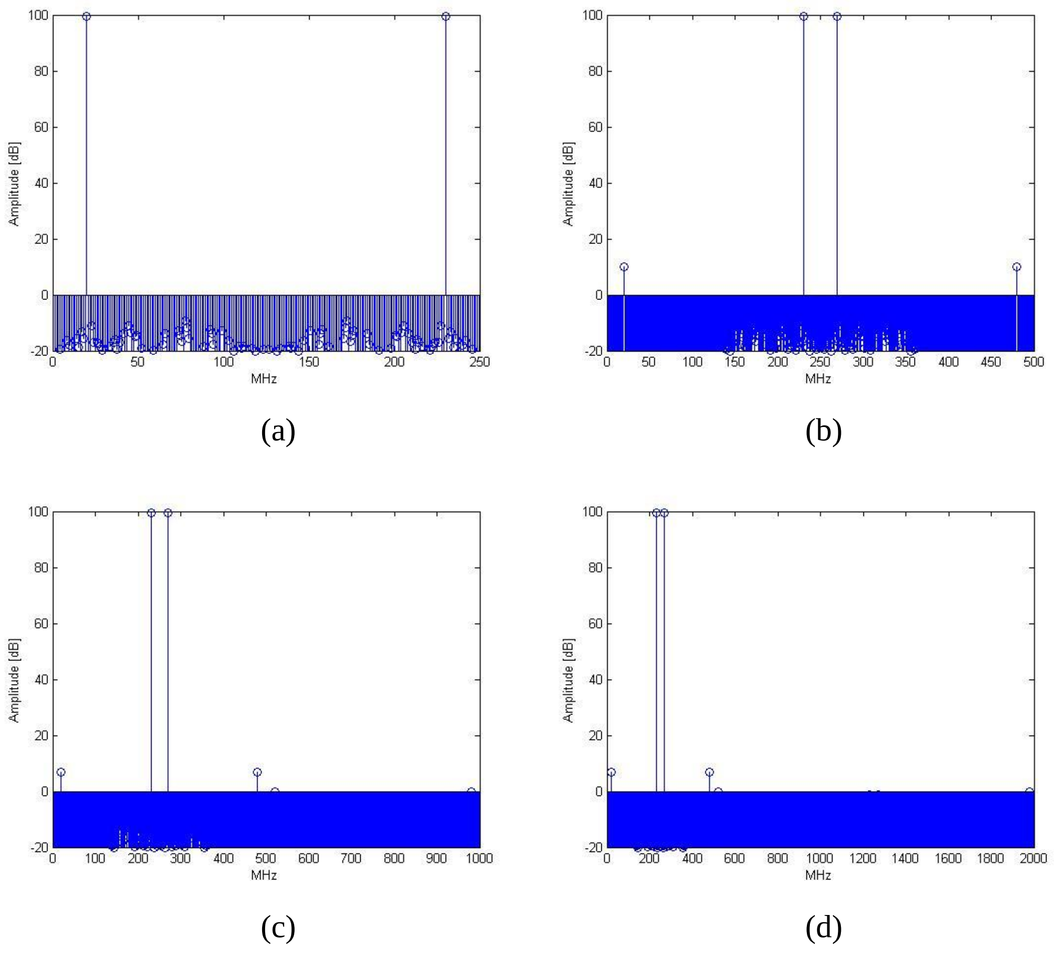


**Figure 3.2 :** Simulation outputs of MATLAB model. FFTs of (a): Input signal, (b): F1 output, (c): F2 output, (d): F3 output

After generating input, up-sampling is performed by zero padding between successive data points. That operation creates an image at $f_s$-$f_{in}$. Then, filtering is performed with “filter” function. Due to the number of filters in pipeline, zero padding and filtering might be performed again.

An example simulation outputs for MATLAB model of interpolation filter modulation are in Figure 3.2. $f_s$ = 250MHz, $f_{in} \approx$ 20MHz, $n_{FFT}$ = 1024, modulation = fs/8

In Figure 3.2 (a), frequency representation of the input signal takes place. After up-sampling and filtering by the first filter F1, images and unwanted components were suppressed. In Figure 3.2 (b), it is seen that F1 suppresses the low frequency signals and passes high frequencies, in other words performs HP filtering. Figure 3.2 (c) and (d) represent the outputs of second and third filters respectively. Frequencies around 250 MHz are passed and others are suppressed. As no image is shown at $f_s$-$f_{in}$, it should be noticed that complex filtering using a Hilbert transform was performed.

**With simulink model**

The goal of repeating the filtering action with Simulink models is better visualization. In order to change model to use Simulink models, several changes had been made.

First of all, Simulink model files (.mdl) are created for each filter. As decided, three interpolation filters with modulation feature are implemented. Building the model begins with writing “simulink” to command line. Then, Simulink library browser appears and necessary blocks are dragged and dropped from here to a new model file opened from “new model” button of the browser. Commonly used blocks are as follows.

- **Integer delay:** used to sample old values of the input
- **Goto & from:** name connection blocks to make model more readable
- **From workspace:** gets data from m-file
- **Gain & Fix:** used together to get 16 bit data
- **Pulse & switch:** used together as a commutator.
- **Sum & product:** performs the job that they are called
- **Subsystem:** Sub model that includes other blocks in it. Improves readability

- **Multiport switch:** Used as a multiplexer. Select input is the coefficient which determines the operating mode of the filter
- **To workspace:** exports data to m-file

After Simulink model files were done, previous MATLAB models were modified to call Simulink model files to use these files for filtering instead of the “filter” function of MATLAB. Function named “sim” is used to call and start a Simulink model file. Because three filters are present, “sim” function is used three times. After completion of the Simulink model simulation, the MATLAB code continues execution. Rest of the simulation process is the same as in former MATLAB models. Simulink models are shown in Appendix S.

### 3.1.3. Operation modes of the filters in 85 dB design

This section explains operation modes of the interpolation and modulation block of our 85 dB design. Since the 85 dB design is not our final design, all operation modes are not listed in Table 3.2, only 8x interpolation is shown for simplicity. $F_{DAC}$ is the DAC clock frequency (clock frequency at filter output), and it is assumed to be set at 2 GHz. F1, F2, and F3 represent three half -band filters, each up-sampling the input clock frequency by a factor of 2. F_center represents the frequency shift provided by the 8x interpolating filter. The modulation modes using 8x interpolation is shown in Table 3.2. Similar tables exist for 2x and 4x interpolation modes.

In Table 3.2, operating modes of the sub-filters F1, F2 and F2 are also given. F1 has 4 filtering modes. Mode 0 describes low-pass basic filtering. Mode 1 describes the first Hilbert transform mode that passes positive frequencies and suppresses the negatives. Mode 2 describes high-pass filtering. Mode 3 describes the second Hilbert transform mode that passes negative frequencies and suppresses the positives.

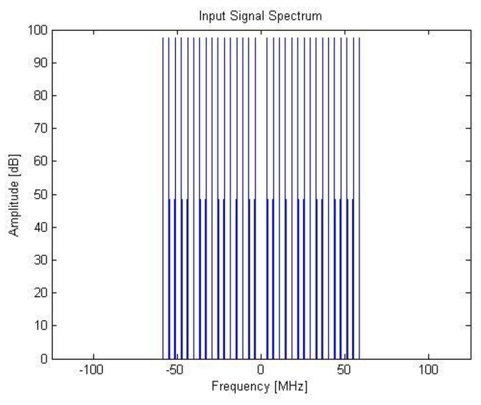
F2 and F3 have 8 modes of operation. Mode 0 describes low-pass basic filtering. Mode 1 describes $45^o$ pass-band shift. Mode 2 describes the first Hilbert transform mode that passes positive frequencies and suppresses the negatives. Mode 3 describes $135^o$ pass-band shift. Mode 4 describes high-pass filtering. Mode 5 describes $225^o$ pass-band shift. Mode 6 describes the second Hilbert transform mode that passes negative frequencies and suppresses the positives. Mode 7 describes $315^o$ pass-band shift.

**Table 3.2 :** Filter modulation modes at 8x over-sampling.

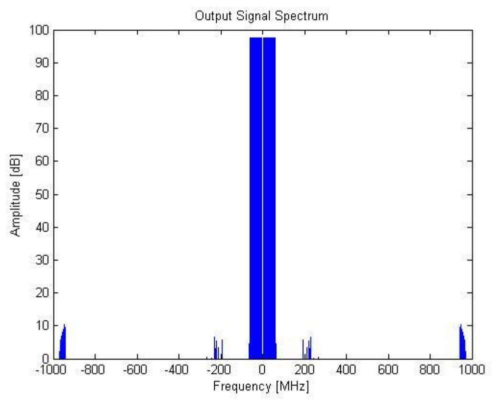
| pre_mod | F1 | F2 | F3 | f_center | f_center @ $f_{DAC}$ = 2GHz |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 (DC) | 0 (DC) |
| 1 | 1 | 1 | 0-1 | $f_{DAC}/16$ | 125 |
| 0 | 2 | 2 | 1 | $2f_{DAC}/16$ | 250 |
| 1 | 3 | 3 | 1-2 | $3f_{DAC}/16$ | 375 |
| 0 | 0 | 4 | 2 | $4f_{DAC}/16$ | 500 |
| 1 | 1 | 5 | 2-3 | $5f_{DAC}/16$ | 625 |
| 0 | 2 | 6 | 3 | $6f_{DAC}/16$ | 750 |
| 1 | 3 | 7 | 3-4 | $7f_{DAC}/16$ | 875 |
| 0 | 0 | 0 | 4 | $8f_{DAC}/16$ | 1000 |
| 1 | 1 | 1 | 4-5 | $9f_{DAC}/16$ | 1125 |
| 0 | 2 | 2 | 5 | $10f_{DAC}/16$ | 1250 |
| 1 | 3 | 3 | 5-6 | $11f_{DAC}/16$ | 1375 |
| 0 | 0 | 4 | 6 | $12f_{DAC}/16$ | 1500 |
| 1 | 1 | 5 | 6-7 | $13f_{DAC}/16$ | 1625 |
| 0 | 2 | 6 | 7 | $14f_{DAC}/16$ | 1750 |
| 1 | 3 | 7 | 7-0 | $15f_{DAC}/16$ | 1875 |

#### 3.1.3.1. Baseband

This is the first and fundamental mode of the system. In this mode of operation, all filters work at mode 0 which corresponds to no-modulation, just interpolation. To do that, variables “hb1_mod”, “hb2_mod” and “hb3_mod” are set to 0. In the end, input and corresponding output signal of this mode is obtained as in Figure 3.3.



(a)



(b)

**Figure 3.3 :** Input (a) and output (b) spectrum of the system in baseband mode.

As seen in Figure 3.3, output data rate is 8x faster than input data rate as 3-stage interpolation is used. Besides, some parts of filtered images already stand in spectrum and corrupt the output. That non-ideality shows the performance of the interpolation filters and represented with SFDR calculations. That will be discussed later sections.

#### 3.1.3.2. $F_{DAC}$/8 modulation

With that modulation mode, input signal both interpolated and modulated with the frequency $f_{DAC}$/8. In order to activate the mode, variables "hb1_mod", "hb2_mod" and "hb3_mod" are set to 2, 2 and 1 respectively. Resulting output spectrum when the same input is applied in Baseband mode, is in Figure 3.4.

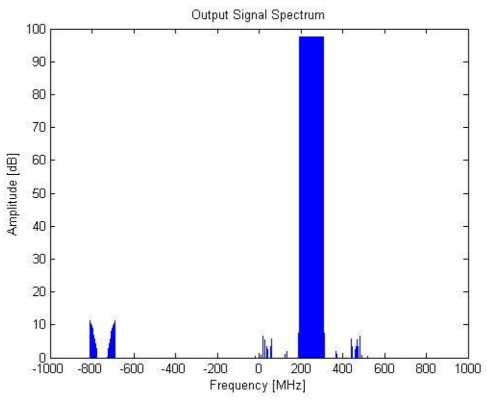


**Figure 3.4 :** Output spectrum of the system in $f_{DAC}$/8 modulation mode.

In Figure 3.4, frequency shift to $f_{DAC}$/8 which is 250 MHz for the example is shown. When compared, it is seen that the plot is just a shifted version of the baseband mode output spectrum.

#### 3.1.3.3. $F_{DAC}$/4 modulation

$F_{DAC}$/4 Modulation mode is the third mode of operation which shifts input data around frequency $F_{DAC}$/4. It can be selected by assigning 0, 4 and 2 to variables "hb1_mod", "hb2_mod" and "hb3_mod" respectively. Resulting spectrum plot is in Figure 3.5.

Again, plot in Figure 3.5 is a shifted version of the output in baseband mode.

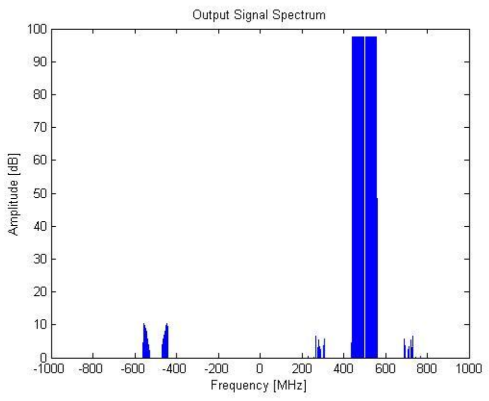


**Figure 3.5 :** Output spectrum of the system in $f_{DAC}/4$ modulation mode.

#### 3.1.3.4. $3F_{DAC}/8$ modulation

With this modulation mode, it is possible to modulate input signal with the frequency $3F_{DAC}/8$. Output spectrum plot is given in Figure 3.6.

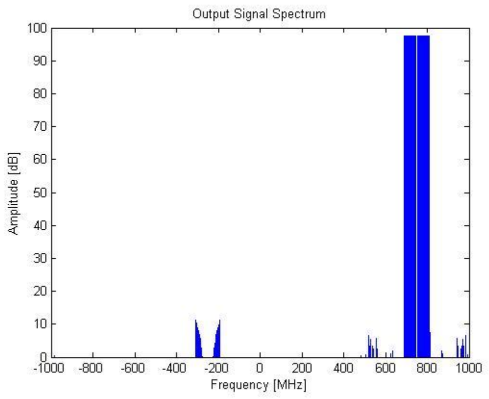


**Figure 3.6 :** Output spectrum of the system in $3f_{DAC}/8$ modulation mode.

To activate $3F_{DAC}/8$ modulation, variables “hb1_mod”, “hb2_mod” and “hb3_mod” are set to 2, 6 and 3 in the given order.

#### 3.1.3.5. $\pm F_{DAC}/2$ modulation

This modulation mode modulates the input signal with frequency $f_{DAC}/2$ which also equals to modulating with $-f_{DAC}/2$. To activate, “hb1_mod”, “hb2_mod” and “hb3_mod” are set to 0, 0 and 4 which work first two filters LP and last filter HP mode. Resulting output spectrum is given in Figure 3.7.

In Figure 3.7, it is seen that $f_{DAC}/2$ modulation is performed. From symmetry, it can be said that if the input signal is real, output also becomes real, which means a real modulation is performed. This modulation can be realized using a single channel.

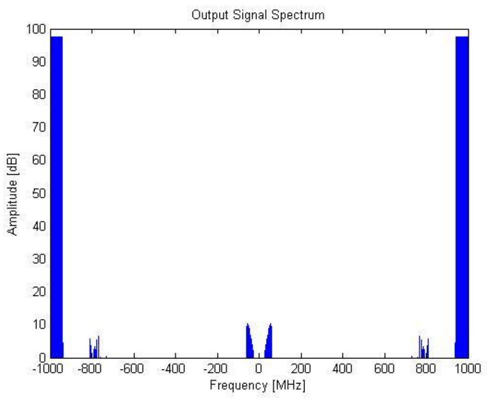


**Figure 3.7 :** Output spectrum of the system in ±$f_{DAC}/2$ modulation mode.

### 3.1.3.6. -$F_{DAC}/8$ modulation

This is the negative version of the $f_{DAC}/8$ modulation. With -$f_{DAC}/8$ modulation mode, which is activated by setting mode variables to 2, 6 and 7, input signal is shifted to around -$f_{DAC}/8$ frequency. Spectrum of the output signal is in Figure 3.8.

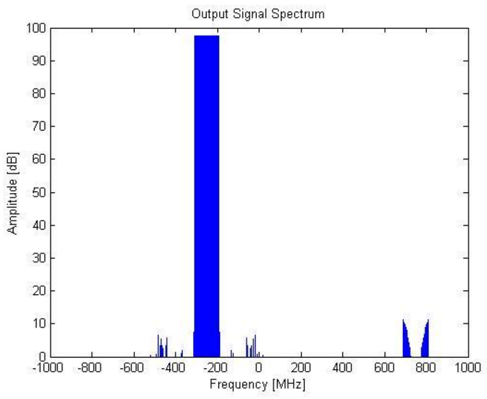


**Figure 3.8 :** Output spectrum of the system in -$f_{DAC}/8$ modulation mode.

As seen in Figure 3.8, spectrum plot is the mirror image of the output in $f_{DAC}/8$ Modulation mode in Figure 3.4.

### 3.1.3.7. -$F_{DAC}/4$ modulation

Like -$f_{DAC}/8$ Modulation mode, -$f_{DAC}/4$ modulation mode is the negative version of the $f_{DAC}/4$. .It can be selected by assigning 0, 4 and 6 to variables “hb1_mod”, “hb2_mod” and “hb3_mod” respectively. Resulting spectrum plot is in Figure 3.9.

Again, spectrum plot in Figure 3.9 is the mirror image of the output in $f_{DAC}/4$ modulation mode in Figure 3.5.

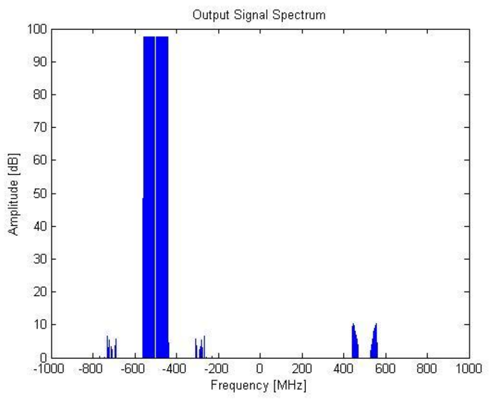


**Figure 3.9 :** Output spectrum of the system in -f$_{DAC}$/4 modulation mode.

#### 3.1.3.8. -3F$_{DAC}$/8 modulation

Within this mode of operation, it is possible to modulate input signal with the frequency -3F$_{DAC}$/8. Output spectrum plot is given in Figure 3.10.

To activate 3F$_{DAC}$/8 modulation, variables “hb1_mod”, “hb2_mod” and “hb3_mod” are set to 2, 2 and 5 in the given order.

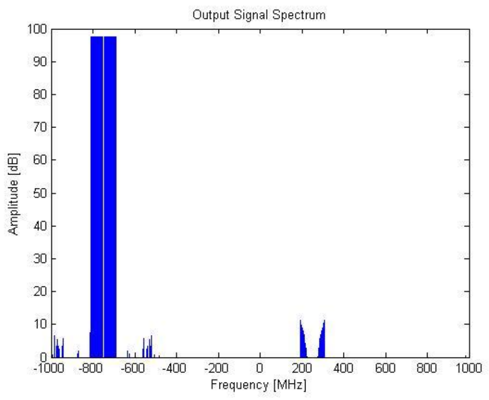


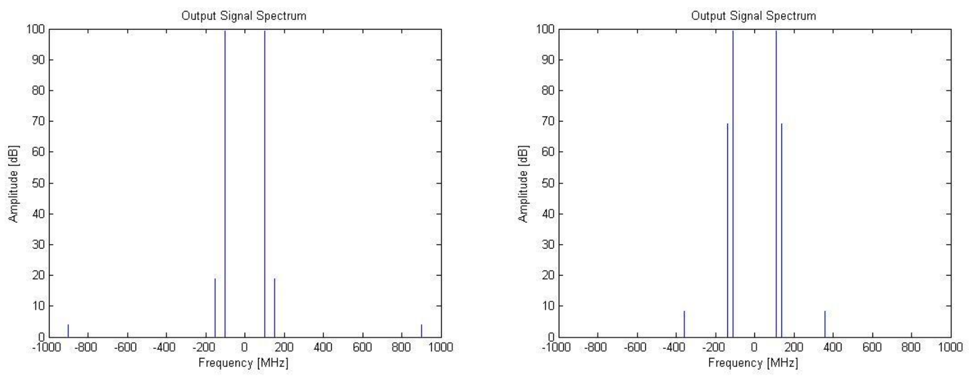
**Figure 3.10 :** Output spectrum of the system in -3f$_{DAC}$/8 modulation mode.

#### 3.1.3.9. F$_{DAC}$/16 modulations

Until now, all possible operation modes are explained and their results are presented. In this section, a possible weakness of the presented system and its solution is given.

#### A problem: using all the Nyquist band

During early times of design, it is desired to shift the input signal wherever we like in the output signal spectrum. Several operation modes were defined and explained to accomplish the objective. However, input signal frequencies between two modulation mode frequencies (ie. near F$_{DAC}$/16) fall into filter transition band and undesired levels of suppression are observed. That issue is visualized in Figure 3.11.



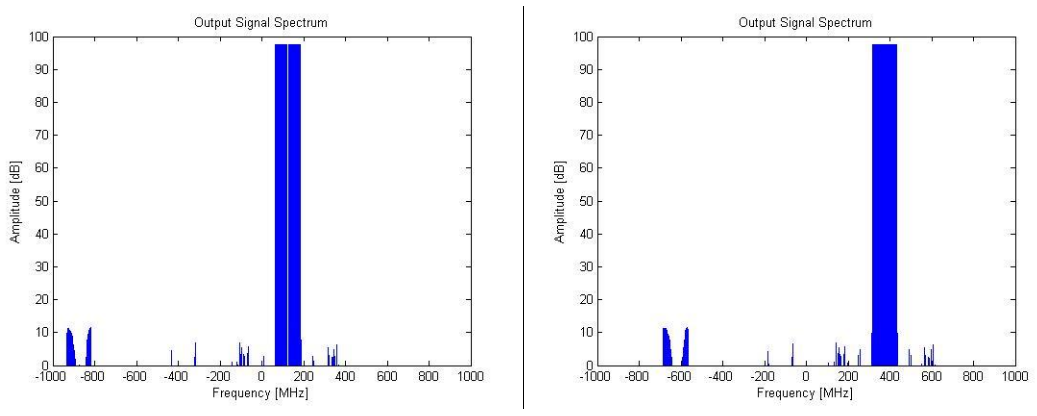
(a) (b)

**Figure 3.11 :** Attempts to use mid-mode frequency band. Fin = (a): 100 MHz, (b): 110 MHz.

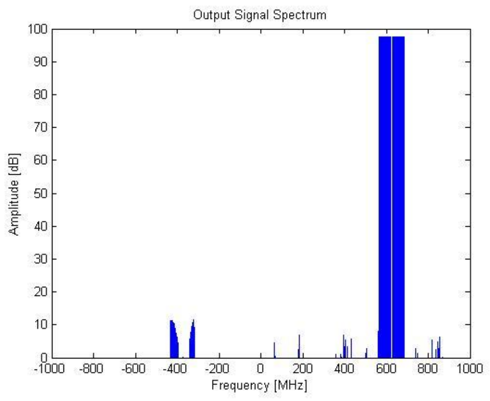
In Figure 3.11, it is desired to use a real valued mid-mode frequency band, which is chosen to be around 125 MHz. Frequency of the input signal is chosen as 100 and 110 MHz, for Figure 3.11 (a) and Figure 3.11 (b) respectively, and it is seen that images (at 150 MHz in (a) and 130 MHz in (b)) are no longer suppressed efficiently due to incapability of the filters.

## Solution: pre-modulation & mid-modes

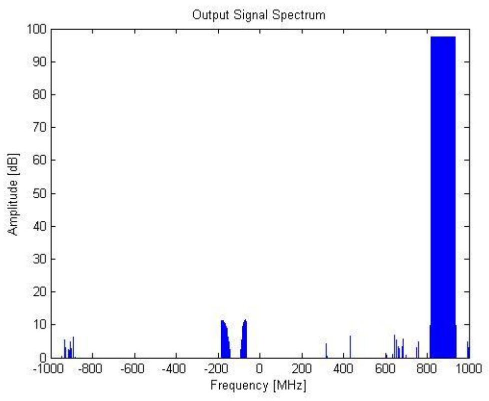
To solve the issue, 8 new operation modes are defined, which modulate the input signal to mid-mode frequencies without any SFDR penalty. A pre-modulation option is needed. When activated by setting "pre_mod" variable to 1, it modulates the input signal at baseband and prepares it for mid-mode filtering. As filter modes for that kind of modulations are already given in Table 3.2, only resulting spectrum plots for these operation modes are given in Figure 3.12.



(a) (b)



(c)



(d)

**Figure 3.12 :** Output Spectrums of mid-modes. (a): $F_{DAC}/16$, (b): $3F_{DAC}/16$, (c): $5F_{DAC}/16$, (d): $7F_{DAC}/16$.

Spectrum plots for mid-modes with negative frequencies are in Appendix P. It is seen in Figure 3.12(a) that input signal can be put around frequency $f_{DAC}/16$ without any image rejection problem. By the help of mid-modes, all output bandwidth can be used without penalty.

### 3.1.4. SFDR calculations

In previous sections, it is mentioned that designed interpolation and modulation system works as desired. However, proving functionality is not enough for such a system and giving performance metrics of the system is necessary. Therefore, SFDR calculations were made to evaluate signal distortion. SFDR for an operation mode is obtained from the worst SFDR value found during a full-scale frequency sweep, using a single tone.

#### 3.1.4.1. SFDR vs. operation mode

First analyzed metric is the change of SFDR wrt. operation mode. It is seen from the spectrum plots that modulated versions of the baseband mode is just shifted versions of the same pattern and no change in SFDR was expected from different modulation modes. To calculate SFDR values for every operation mode, a modified MATLAB model was generated which creates same input signal that covers all filter pass-band and changes operation modes from 1 to 16 in Table 3.2. It calculates SFDR values for every mode and in the end it plots them. Obtained graph is in Figure 3.13.

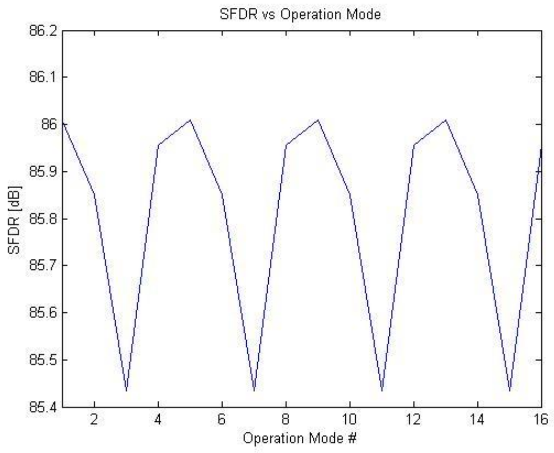


**Figure 3.13 :** SFDR values wrt. operation modes.

As seen in Figure 3.13, worst SFDR was calculated as 85.4 dB for 3rd, 7th, 11th and 13th modes. It is seen that characteristics of the first four modes replicate themselves and a symmetric pattern is obtained. In the beginning, a constant value was expected for all modes, but instead, a replicating pattern was found. The difference is subtle, 0.6 dB.

### 3.1.4.2. SFDR vs. signal amplitude

During simulations except this one, input signal amplitude is chosen as 0.9 times of the maximum input signal range. It is known that output SFDR value will change if amplitude of the input signal changes. To show that, amplitude of the input signal changed from 0.1 times to 0.9 times and SFDR values for baseband mode. Resulting graph is in Figure 3.14.

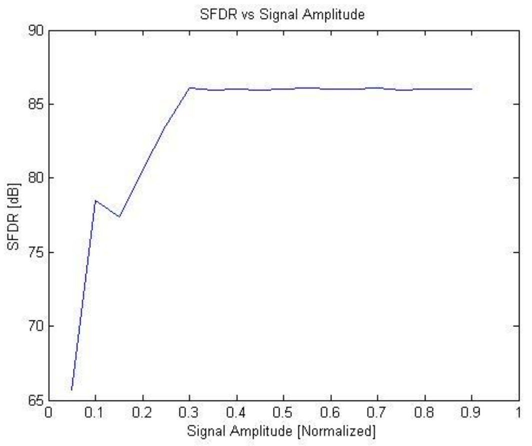


**Figure 3.14 :** SFDR values wrt. signal amplitude.

In Figure 3.13, it is seen that when signal amplitude drops under 0.3 times of its maximum, SFDR values also drops sharply. After 0.3 times, SFDR is almost constant and does not show a significant change. This is expected, since stop-band attenuation of the filter limits the SFDR.

### 3.1.4.3. SFDR vs. input frequency

In Section 3.1.3.9, difficulties about image rejection caused by increase in signal frequency was briefly mentioned. To analyze that effect more deeply, SFDR values for input signals with different frequencies are calculated. Results are shown in Figure 3.15.

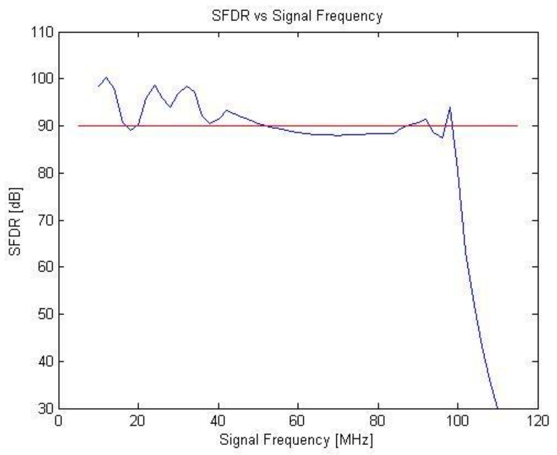


**Figure 3.15 :** SFDR values wrt. signal frequency.

Figure 3.15 suggests that SFDR values are larger than 90 dB when signal frequency is less than 50 MHz. Between 50-85 MHz, SFDR is almost constant at 86 dB. After 100 MHz, SFDR falls significantly. It is seen that for signals with lower frequency, SFDR of the system is larger; when input frequency increases, SFDR decreases slowly and later sharply. This is expected, as the SFDR follows the frequency response curve of the filter.

## 3.2. HDL Coding

Hardware Description Language (HDL) model of the system is generated to synthesize and finally implement the system.

Verilog HDL, which is a C-like HDL, is chosen to model the system. In following sections, sub-blocks of the system are presented and comparisons of the results with respect to Simulink models are given.

### 3.2.1. Sub-blocks

Interpolation and modulation system that is modeled consist three level interpolation filters with modulation feature. Verilog Codes of the sub-blocks are in Appendix V.

#### 3.2.1.1. Filter-1

Filter-1, in short F1, is the first interpolation filter of the system with modulation feature. Thanks to noble identity of Half Band Filters and other techniques; F1, which is a 55 tap filter, has only 14 different coefficients and multipliers. Those coefficients are in Table 3.3. Because of being the first filter, F1 should have the sharpest transition band with respect to other filters. The reason is that, the images occur closer than other filters. Verilog model of the F1, which is derived from the Simulink model, is now described.

**Table 3.3 :** Coefficients in F1

| Coefficients | Value |
|---|---|
| C0 | -4 |
| C1 | 13 |
| C2 | -34 |
| C3 | 72 |
| C4 | -138 |
| C5 | 245 |
| C6 | -408 |
| C7 | 650 |
| C8 | -1003 |
| C9 | 1521 |
| C10 | -2315 |
| C11 | 3671 |
| C12 | -6642 |
| C13 | 20755 |
| C14* | 32768* |

*:not implemented with multiplier

First *always* block contains a shift register chain that takes real and imaginary data inputs. It is 27 levels deep. Then "oys_register" comes, which is the short form of "last register of shift register". Normally, this is equal to associated shift register but for some modes of operation, the value is multiplied by -1 (or just sign is changed). After that, first adders arrive. As HBFs have symmetric coefficients, these adders sum up associated inputs that will be multiplied by same coefficients. By doing this, half of the multipliers are saved. For complex filtering modes, coefficients are not in their original form but their signs are changed. So, when these modes are active, subtraction is performed. After fist adders, multiplications with coefficients are done. Thereafter, within four levels, summations of the partial products are performed.

So far, no truncation or rounding is performed and outputs become 37-bit which is excessive. Therefore, truncation to 16-bit is done and final value is named as "kesik"(_r for real _i for imaginary). Next block is important for complex filtering again. That block switches real and imaginary filtered data to fulfill the operation of Hilbert transformer.

Last two *always* blocks act as a commutator, which connects the filtered data or shifted input data at "oys_register" to the output. At the end, 2x interpolation and optional modulation is performed during F1.

#### 3.2.1.2. Filter-2

Filter-2, comes after F1, operates much the same as F1 but with 4 more modes of operation. Verilog model of F2 generally is similar to F1 with a few differences. These differences are explained in this section.

First difference is the shift register length. Because of that F2 should suppress images far from F1's, it designed as a 23 tap filter, which can be implemented with only 6 different coefficients given in Table 3.4. Shift register length is chosen as 11 to hold all necessary old values of input. Moreover, because of the 4 new modes (mode 1, 3, 5 and 7) that F1 does not have, generating "oys_register" was changed. For the new modes; "oys_register" is equal to sum (or difference) of real and imaginary registered inputs times square root of 2. As that value becomes 32 bit long, it was also truncated to 16-bits as usual and named as "kesik_oys". For the new 4 modes, this value is assigned to "oys_register", otherwise directly the registered input value assigned as in F1.

There are also differences in the first adders as well. For the new 4 modes, not only the real values are added for real first adders, but also the associated imaginary values. That is also true for the imaginary first adders. For the first 4 modes (mode 0 2, 4 and 6), operation is the same as F1 (real first adders consists of real SR's values only and so as are the imaginary first adders).

**Table 3.4 :** Coefficients in F2

| Coefficients | Value |
|---|---|
| C0 | -2 |
| C1 | 17 |
| C2 | -75 |
| C3 | 238 |
| C4 | -660 |
| C5 | 2530 |
| C6* | 4096* |

*:not implemented with multiplier

Rest of the model is just like F1: multiplication with coefficients, adder trees, output truncation and commutator. With the modifications described above, F2 is made to have eight modes of operation, each corresponding to a different modulation frequency.

### 3.2.1.3. Filter-3

Filter-3 is the last interpolation and modulation filter of the system. F3 operates fastest and needs to suppress images that are farthest. F3 is almost the same as F2, but with lesser differences.

First difference is the SR length, which is 8 for F3. As F3 is the last interpolation and modulation filter, it has the easiest suppression specifications than the others. So 8-length SR, which corresponds to 14-tap filter, was found satisfactory for F3. Other difference is the coefficients which are presented in Table 3.5.

**Table 3.5 :** Coefficients in F3

| Coefficients | Value |
|---|---|
| C0 | -39 |
| C1 | 273 |
| C2 | -1102 |
| C3 | 4964 |
| C4* | 8192* |

*:not implemented with multiplier

### 3.2.2. Comparison with MATLAB models

After building a new model, it is simulated and compared with other known models. Therefore, Verilog models must be simulated. To do that, a verilog testbench is created to give inputs and collect outputs from the system. Details of that test bench and verification process are described below.

#### 3.2.2.1. Testbench: getting data from verilog to MATLAB

The Testbench has four clocks in order to simulate a 3-level interpolation and modulation system. These clocks are generated within the testbench, according to a period parameter. That period is for the input and other clocks use half of the previous clock's period. Besides, operation modes of the filters can be changed via changing appropriate mode bits in test bench. Moreover, test bench has a LUT including sine values for input.

When a Verilog simulation ends, an m-file is generated to write outputs of the simulation. Later on, it will be used for MATLAB comparisons. In that m-file, values of the output signal is written in a format which MATLAB code can read. Also for possible 'X' values, a precaution was taken and those values are assigned to NANs. Verilog code of testbench and its sample output are in Appendix V.

#### 3.2.2.2. Comparison

When the simulation is finished, an m-file including the output of the system is generated. In order to examine and compare it with the MATLAB models, another MATLAB code was written. That "Verilog output reader" code was created to be run after MATLAB model was simulated. It reads the output file of the Verilog model, and plots this data on top of the old data generated by the MATLAB model. It is seen that these two data are exactly the same. In order to be sure, Verilog output and MATLAB outputs are subtracted and all zeros are expected. It can be said that Verilog models are verified with no errors.

## 3.3. Synthesis, PAR and final layout

It is mentioned earlier that Verilog models are created to synthesize the circuit and to draw the layout to produce a chip. This section includes the process of generating the layout.

### 3.3.1. Flow & tools

Microelectronic design flow consists of modeling (MATLAB), hardware description (Verilog), synthesis, place and route (PAR) and generating final layout steps. From now on; synthesis, PAR and final layout generation will be described and physical design tools that are used will be explained.

For complex digital system design, using physical design tools becomes a must. Cadence design environment was used. Tools that are used for synthesis, PAR and final layout are: RTL compiler, Encounter and Virtuoso. Details of the design process are presented in following sections (Cadence tool called NCSim is previously used for verilog simulations).

### 3.3.2. Synthesis with RTL compiler

Synthesis is describing a behavioral HDL code with logic gates defined in standard gate library of the chosen process. That can be done either by hand (full-custom) or with CAD tools (semi-custom). Semi-custom design can also be divided to subsections on the basis of using automatic mapping to gates.

In the very beginning of the work, a certain number of semi-custom designs were generated and some of their full-custom equivalents are designed in order to understand the limits of the process and the capability of the CAD tools. During that time, various adder architectures [17-22] are tested. In the end, it is seen that full-custom designs give limited benefits on performance; moreover, they seriously lengthen the design and simulation times. In conclusion, it is decided to use a semi-custom design flow driven by the CAD tool.

Synthesis flow on RTL Compiler may be the easiest one, as flow is almost automatic and there are only few things to do. A tcl-script was written and run in the RTL Compiler. All work including desired specifications, file names, operations etc. is done by that script. But before moving to the script, opening the RTL Compiler will be explained.

#### 3.3.2.1. Synthesis flow

The synthesis flow contains information about using the synthesis tools, writing and using scripts to control them. Details of the written specifications and usage of the tools are grouped under Appendix I.

#### 3.3.2.2. Retiming / pipeline

To increase the performance of the circuit or to make it be able to operate with higher clock frequencies, an operation named “retiming” is performed. Retiming means repositioning combinational logic blocks to decrease longest path delay without changing functionality [23]. It can also be used as an automatic pipeliner, by putting registers after maximum combinational delay. It is seen that without automated retiming, user defined pipelined architectures had worse performance.

### 3.3.3. PAR with encounter

PAR (Place and Route) is next level of the design flow, where placement of the logic gates and generation of routs occur. PAR tool of Cadence is Encounter, which takes the output files of the RTL compiler as input, and generates a delay file .sdf for simulation, and layout file .gds, for production.

#### 3.3.3.1. Steps of PAR

During the PAR process at Encounter, commands defined in a script was used, which is a modified version of an Austrian Microsystems' PAR script. Commands of that script and PAR flow explained together in Appendix I.

#### 3.3.3.2. Outputs: GDS-II & SDF

When PAR is done, two files are generated in order to be used for production and simulation. These files are GDS-II and SDF files, respectively.

GDS-II file includes layer information which is used for the foundry to produce a chip. A plotted GDS-II file is in Figure 3.16.

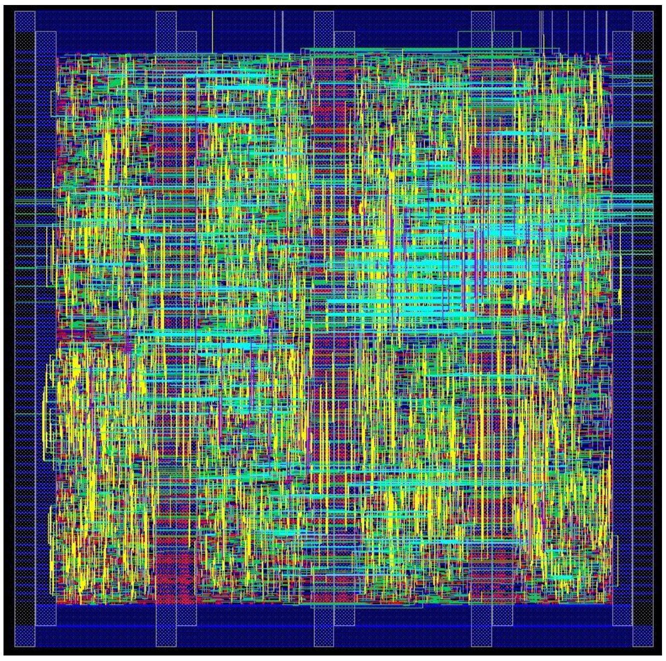


**Figure 3.16 :** Plotted GDS-II File of the 85 dB filter design in 0.15u LFoundry CMOS process

SDF is the acronym for standard delay format. An SDF file contains delay information of the final layout. When used with Verilog file that was generated, accurate post-PAR simulations can be made.

## 3.4. Post-PAR Simulations & Results with NCSim

In previous section, it is mentioned that by using Verilog and the SDF file, accurate post-PAR simulations can be performed. To do that, NCSim tool of Cadence is used for simulation and results are examined.

### 3.4.1. Comparison of results

In order to make post-PAR gate level Verilog simulations, modifications were made on Verilog test bench. First modification was including a gate library with typical conditions. By doing that, functionalities of the process-specific gates are defined to the simulator. Another modification was reading the SDF file that includes delays of both gates and the interconnects. Once simulation setup was finished, the simulation is performed. See Appendix V.

After simulation, a MATLAB m-file was generated as done before and read by MATLAB to compare with previous results obtained by behavioral models. It is found out that exactly the same functionality is achieved.

### 3.4.2. Maximum operating frequency

During simulation step, clock speed was increased to find its maximum value without harming functionality. It is seen that maximum clock in the system can be increased to 1.3 GHz with 2 levels of retiming for multipliers in L-Foundary 0.15 μm process.

### 3.4.3. SFDR of the output

As functionality is found to match with MATLAB models, SFDR of the final design exactly matches the MATLAB models. SFDR results are not repeated here. SFDR plots can be found in Section 3.1.4.

# 4. IMPROVEMENTS ON STANDARD ARCHITECTURE

This section covers the design improvements made on the standard architecture explained in Section 3. Improvements can be categorized into two headings: better SFDR performance and increased number of filter modes. Section also includes the new HDL representation, implementation and simulations of the new design, which is implemented not on the LFoundry process, but on TSMC 0.18 μm CMOS process.

## 4.1. Targeting Higher SFDR

In a DAC chip with digital signal processing, system performance should be determined by performance of the DAC. To satisfy that, performance of the digital blocks should be better than DAC's performance, within desired operating frequencies. Therefore, in order to be able to work with a DAC with 90 dB SFDR, improved digital filters should have better performance than 90 dB. This specification is chosen as 95 dB.

A digital filter's SFDR performance is given by two factors: filter order and coefficient resolution. Although increasing one item may only bring small benefits, it is necessary to increase both, if large improvements are required.

### 4.1.1. Higher order filters

Increasing filter order is the first issue adopted from analog filter design. However, it is necessary to consider the digital implementation when increasing the filter order. In a FIR filter, increasing the filter order also increases the final adder tree. Number of leaves of the adder tree is desired to be $2^n$. If this is not satisfied, area of the filter does not become optimum. Figure 4.1 shows an optimum and non-optimum cases for the adder tree.

During calculation of the new coefficients, number of leaves is targeted to be made $2^n$. This value is made non-optimum, only when it is worth doing so (that is, when SFDR performance improves significantly).

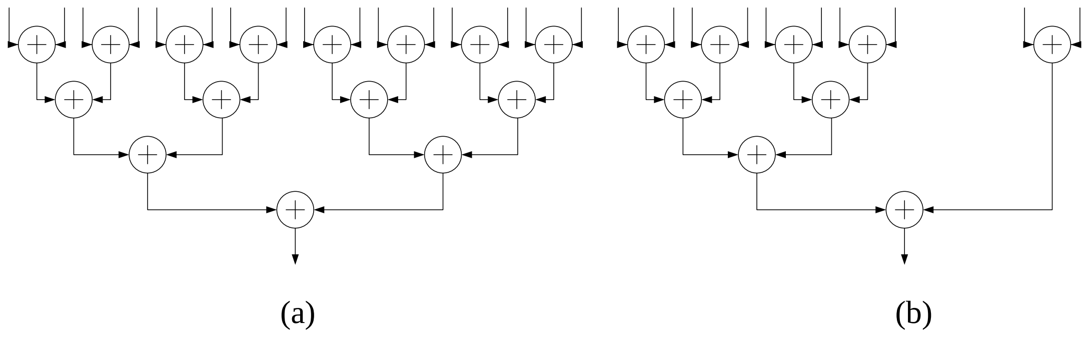


**Figure 4.1 :** Examples of optimum (a) and non-optimum (b) adder trees.

### 4.1.2. Larger bit coefficients

Coefficient resolution is another item affecting the SFDR directly. Using larger bit coefficients reduces the quantization error between real numbered coefficients calculated from solving the transfer function for given inputs and quantized coefficients. However, using coefficients with more resolution increases the power consumption, the hardware cost and reduces the operating frequency of the filter, which now requires higher resolution multipliers.

### 4.1.3. Obtaining filter coefficients

Calculation of Filter coefficients is done with the MATLAB fdatool. In order to get filters with higher SFDR, filter order and coefficient word length parameters are chosen to be larger than the previous architecture. Comparison of the filter fdatool parameters and resulting filter specifications are in Table 4.1

**Table 4.1 :** Comparison of fdatool parameters and resulting filter characteristics.

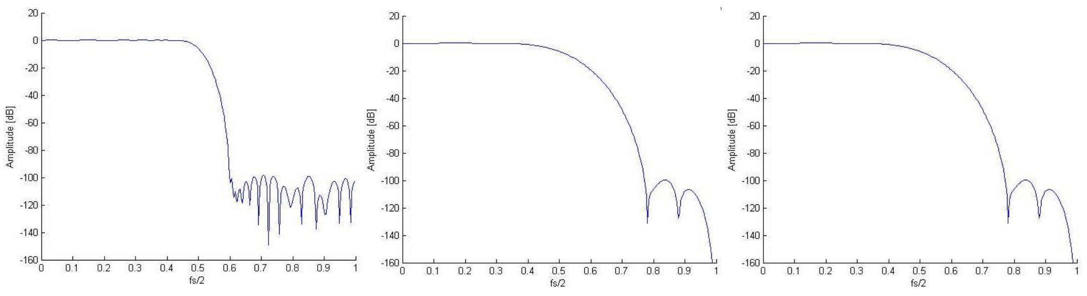
| Filter Name | Filter Order | Number of Coeff. | w pass | Word Length | Effective WL | SFDR [dB] |
|---|---|---|---|---|---|---|
| 1 old | 54 | 14 | 0,4 | 16 | 15 | 79 |
| 1 new | 62 | 16 | 0,40021 | 19 | 18 | 98,3 |
| 2 old | 22 | 6 | 0,23 | 13 | 12 | 71 |
| 2 new | 22 | 6 | 0,2003 | 17 | 16 | 99,5 |
| 3 old | 14 | 4 | 0,195 | 14 | 13 | 79 |
| 3 new | 22 | 6 | 0,225 | 17 | 16 | 99,7 |

After calculation, it is found out that Filter 2 and Filter 3 specifications are quite similar and Filter 3 can be used twice instead of Filter 2. By doing that, a small SFDR gain is obtained. In conclusion, Filters in Table 4.2 are used in the optimized design.

**Table 4.2 :** Filter characteristics used in the design.

| Filter Name | Filter Order | Number of Coeff. | w pass | Word Length | Effective WL | SFDR [dB] |
|---|---|---|---|---|---|---|
| 1 old | 54 | 14 | 0,4 | 16 | 15 | 79 |
| 1 new | 62 | 16 | 0,40021 | 19 | 18 | 98,3 |
| 2 old | 22 | 6 | 0,23 | 13 | 12 | 71 |
| 2 new | 22 | 6 | 0,225 | 17 | 16 | 99,7 |
| 3 old | 14 | 4 | 0,195 | 14 | 13 | 79 |
| 3 new | 22 | 6 | 0,225 | 17 | 16 | 99,7 |

Frequency responses of the Filters are given in Figure 4.2.



(a) (b) (c)

**Figure 4.2 :** Frequency responses of the filters. (a):Filter-1, (b):Filter-2, (c):Filter-3.

As mentioned before, main performance metric of the filters for the application is SFDR. However, in order to obtain other performance metrics of the designed filters, MATLAB simulations are performed. As a result, pass-band ripple of the filters are obtained as $2x10^{-4}$ dB (taken as 0) and SNR is calculated as 95.4 dB for F1 and 94.6 dB for F2 and F3. Combined SNR is 93.4 dB. The visible ripple in the plots is a graphics artifact. Zoomed plots show 0.0002 dB ripple.

## 4.2. SFDR Calculations of Improved Filters (with MATLAB)

After designing the filters, SFDR calculations with respect to the Operation mode, Signal amplitude and Input frequency are updated for new coefficient sets.

### 4.2.1. SFDR vs. operation mode

As done with the previous design, optimized filters are simulated for different operation modes and SFDR is calculated for each one. At last, Computed SFDR values are plotted in Figure 4.3.

Figure 4.3 shows that worst SFDR is 99 dB and it becomes 99.5 dB at most. It is seen that the characteristics of the first four modes replicate themselves and a symmetric pattern is obtained like the previous design.

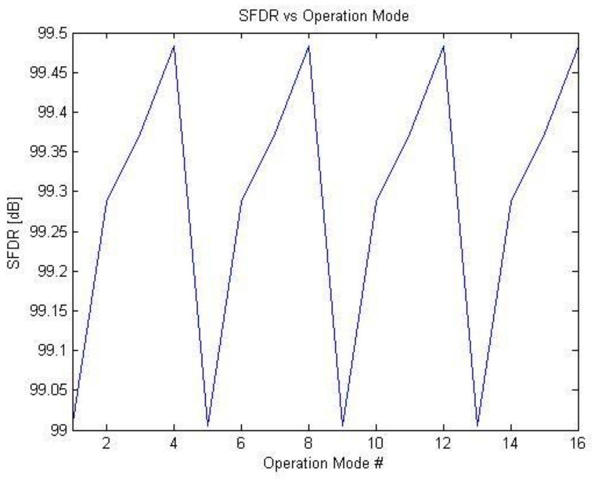


**Figure 4.3 :** SFDR values wrt. operation modes.

### 4.2.2. SFDR vs. signal amplitude

Simulation setup with changing signal amplitude is modified to support using the new coefficients for filtering. SFDR values for signal amplitudes that vary from 0.1 times to 0.9 times the maximum input range, are in Figure 4.4.

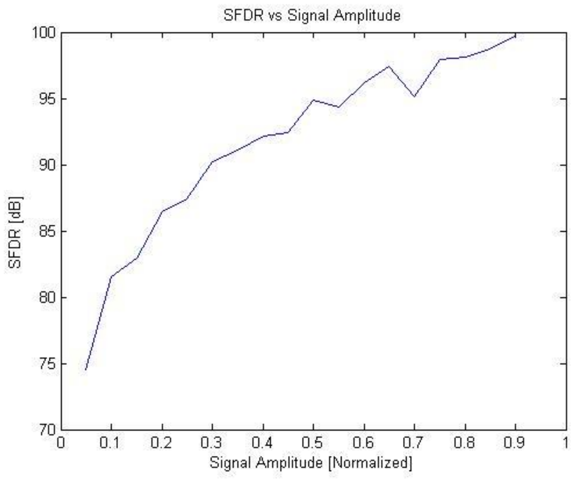


**Figure 4.4 :** SFDR values wrt. signal amplitude.

### 4.2.3. SFDR vs. input frequency

In order to calculate the SFDR variation with respect to input frequency, the frequency of the input signal swept from 0 to 110 MHz and SFDR values are calculated for each step. In the end, results are plotted in Figure 4.5.

Figure 4.5 shows that, for %80 of the input signal bandwidth (up to 100 MHz in this case), worst SFDR is 99 dB. That is an acceptable value to use with a DAC having 90 dB SFDR performance.

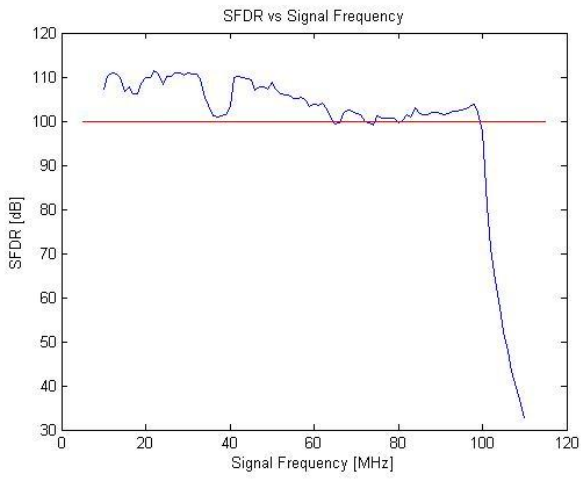


**Figure 4.5 :** SFDR values wrt. signal frequency.

### 4.3. New Modes and Peripherals

During design of the optimized filters, it is decided to add new modes and features to the current system.

#### 4.3.1. Selectable interpolation modes: 8x, 4x, 2x, no int.

In designed interpolation system, 3 filters with 2x interpolation are connected consecutively. That provides up to 8x interpolation. In the optimized design, interpolation feature is made selectable and new modes: 4x, 2x and no interpolation are added. That is done by disconnecting some filters in from the line and applying inputs to directly to output (no int.), to third filter (2x int.) or to second filter (4x int.) instead of applying them to first filter only. Associated block diagram of selectable interpolation cases is in Figure 4.6.

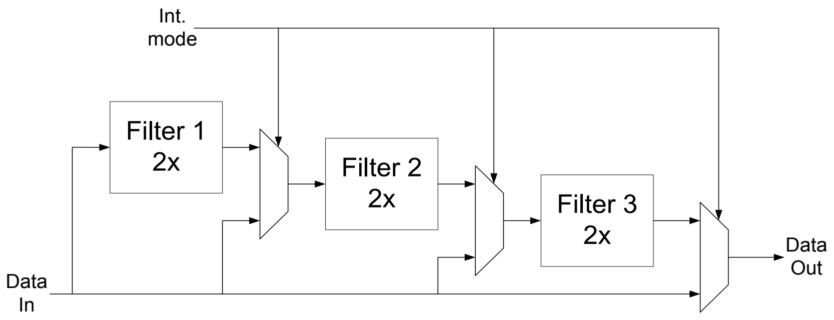


**Figure 4.6 :** Block diagram of selectable interpolation feature.

### 4.3.2. New modulation modes with filters

Having new interpolation modes also allows new modulation modes. In other words, modulation with using only one or two filters is now possible instead of using always three filters. The complete table including new modulation modes associated with 2x and 4x interpolation are given in Table 4.3

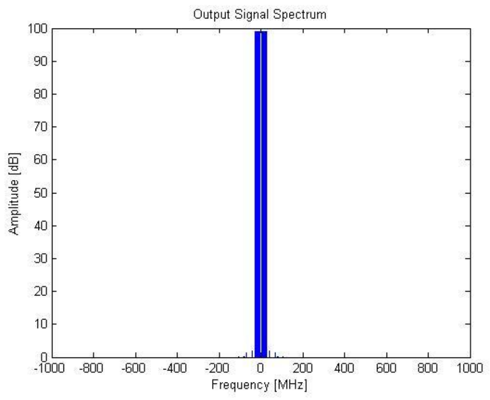
**Table 4.3 :** Complete table of operation modes.

| Code (10 bits) | Inter-polation | pre-mod | F1 | F2 | F3 | f_center | F_center @ $f_{DAC}$=2GHz | DAC Outputs |
|---|---|---|---|---|---|---|---|---|
| 0-X-X-X | No int. | Bypass | Bypass | Bypass | Bypass | N/A | N/A | Independent |
| 1-X-X-0 | 2x | Bypass | Bypass | Bypass | 0 | 0 (DC) | 0 (BB) | Independent |
| 1-X-X-1 | 2x | Bypass | Bypass | Bypass | 1 | fDAC/8 | 250 | |
| 1-X-X-2 | 2x | Bypass | Bypass | Bypass | 2 | 2fDAC/8 | 500 | |
| 1-X-X-3 | 2x | Bypass | Bypass | Bypass | 3 | 3fDAC/8 | 750 | |
| 1-X-X-4 | 2x | Bypass | Bypass | Bypass | 4 | 4fDAC/8 | 1000 | Independent |
| 1-X-X-5 | 2x | Bypass | Bypass | Bypass | 5 | 5fDAC/8 | 1250 | |
| 1-X-X-6 | 2x | Bypass | Bypass | Bypass | 6 | 6fDAC/8 | 1500 | |
| 1-X-X-7 | 2x | Bypass | Bypass | Bypass | 7 | 7fDAC/8 | 1750 | |
| 2-X-0-0 | 4x | Bypass | Bypass | 0 | 0 | 0 (DC) | 0 (BB) | Independent |
| 2-X-1-0/1 | 4x | Bypass | Bypass | 1 | 0-1 | fDAC/16 | 125 | |
| 2-X-2-1 | 4x | Bypass | Bypass | 2 | 1 | 2fDAC/16 | 250 | |
| 2-X-3-1/2 | 4x | Bypass | Bypass | 3 | 1-2 | 3fDAC/16 | 375 | |
| 2-X-4-2 | 4x | Bypass | Bypass | 4 | 2 | 4fDAC/16 | 500 | |
| 2-X-5-2/3 | 4x | Bypass | Bypass | 5 | 2-3 | 5fDAC/16 | 625 | |
| 2-X-6-3 | 4x | Bypass | Bypass | 6 | 3 | 6DAC/16 | 750 | |
| 2-X-7-3/4 | 4x | Bypass | Bypass | 7 | 3-4 | 7fDAC/16 | 875 | |
| 2-X-0-4 | 4x | Bypass | Bypass | 0 | 4 | 8fDAC/16 | 1000 | Independent |
| 2-X-1-4/5 | 4x | Bypass | Bypass | 1 | 4-5 | 9fDAC/16 | 1125 | |
| 2-X-2-5 | 4x | Bypass | Bypass | 2 | 5 | 10FDAC/16 | 1250 | |
| 2-X-3-5/6 | 4x | Bypass | Bypass | 3 | 5-6 | 11fDAC/16 | 1375 | |
| 2-X-4-6 | 4x | Bypass | Bypass | 4 | 6 | 12fDAC/16 | 1500 | |
| 2-X-5-6/7 | 4x | Bypass | Bypass | 5 | 6-7 | 13fDAC/16 | 1625 | |
| 2-X-6-7 | 4x | Bypass | Bypass | 6 | 7 | 14fDAC/16 | 1750 | |
| 2-X-7-7/0 | 4x | Bypass | Bypass | 7 | 7-0 | 15fDAC/16 | 1875 | |
| 3-0-0-0 | 8x | 0 | 0 | 0 | 0 | 0 (DC) | 0 (BB) | Independent |
| 3-1-1-0/1 | 8x | 1 | 1 | 1 | 0-1 | fDAC/16 | 125 | |
| 3-2-2-1 | 8x | 0 | 2 | 2 | 1 | 2fDAC/16 | 250 | |
| 3-3-3-1/2 | 8x | 1 | 3 | 3 | 1-2 | 3fDAC/16 | 375 | |
| 3-0-4-2 | 8x | 0 | 0 | 4 | 2 | 4fDAC/16 | 500 | |
| 3-1-5-2/3 | 8x | 1 | 1 | 5 | 2-3 | 5fDAC/16 | 625 | |
| 3-2-6-3 | 8x | 0 | 2 | 6 | 3 | 6DAC/16 | 750 | |
| 3-3-7-3/4 | 8x | 1 | 3 | 7 | 3-4 | 7fDAC/16 | 875 | |
| 3-0 -0-4 | 8x | 0 | 0 | 0 | 4 | 8fDAC/16 | 1000 | Independent |
| 3-1-1-4/5 | 8x | 1 | 1 | 1 | 4-5 | 9fDAC/16 | 1125 | |
| 3-2-2-5 | 8x | 0 | 2 | 2 | 5 | 10FDAC/16 | 1250 | |
| 3-3-3-5/6 | 8x | 1 | 3 | 3 | 5-6 | 11fDAC/16 | 1375 | |
| 3-0-4-6 | 8x | 0 | 0 | 4 | 6 | 12DAC/16 | 1500 | |
| 3-1-5-6/7 | 8x | 1 | 1 | 5 | 6-7 | 13fDAC/16 | 1625 | |
| 3-2-6-7 | 8x | 0 | 2 | 6 | 7 | 14fDAC/16 | 1750 | |
| 3-3-7-7/0 | 8x | 1 | 3 | 7 | 7-0 | 15fDAC/16 | 1875 | |

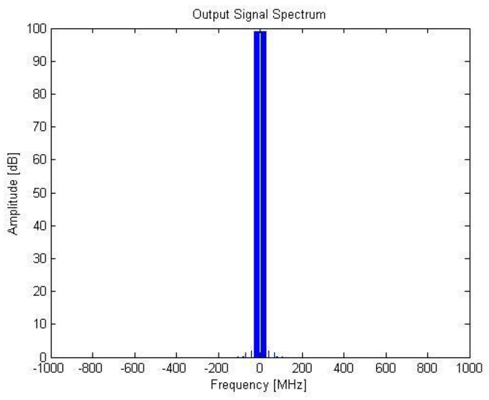
Modes of F1 (mode 0 to 3), F2 and F3 (mode 0 to 7) operate similar with the previous design. So it is not repeated here.

First column of Table 4.3 shows the 10-bit code that is to be written to Filter Register 0 to activate associated operation mode. Second column shows the interpolation rate of the mode. Next four columns represent modes of the filters and pre-modulation features. Other two columns show center frequency that the baseband signal modulated to. Last column gives the information of the DAC outputs are independent or not. Complex modulation results in dependent outputs where non-complex modulation results in independent outputs.

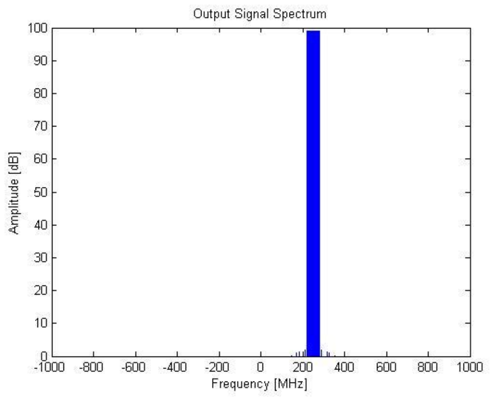
Frequency spectrum plots of the first four modes of operation are given in Figure 4.7. Rest of the plots is in Appendix P.



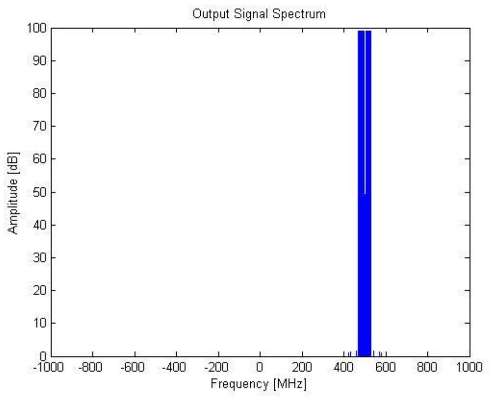
(a)



(b)



(c)



(d)

**Figure 4.7 :** Output spectrums of first four operation modes. (a): No int., (b): 2x int. baseband, (c): 2x int. $F_{DAC}/8$, (d): 2x int. $2F_{DAC}/8$

## 4.4. HDL Differences

HDL models of the new filters are mostly like the previous filters, but with small changes. Those changes in HDL level are increasing filter order, larger bit calculations and peripherals to insert new modes as mentioned above.

To increase filter order, length of the first shift register, number of multiplications, and leaves of the adder tree are increased. Increasing coefficient bit length and calculation sensitivity is done by increasing the bit lengths of the multipliers, adders and the registers which the results are written to. Finally, multiplexers which are given in Figure 4.6 are added to system to maintain the new modes.

## 4.5. Synthesis, PAR & Layout Results

After generation of the HDL models, the new design is synthesized, placed and routed. This section describes the implementation of the design to silicon.

### 4.5.1. Synthesis

Synthesis of the design is done by Cadence's RTL compiler tool. Artisan's 0.18 μm gate library and typical condition delay library is used for timing specification.

To synthesize the design to gates, RTL compiler is programmed via a TCL script. That script contains the addresses of the Verilog and library files read by the tool, constraints of design and commands that perform steps of the synthesis. File names and commands can be found from the script in Appendix R. The constraints and the reasons for selecting such constraints are explained in this section.

First set of constraints for the synthesis are related to the clock. That is given by "define_clock" command. Period of the slowest clock (1X) of the system is written. For other clocks (2x, 4x, 8x), the same period value and "divide_period" option is used. Fastest possible operating speed is desired for filters; therefore, period value is chosen as small as possible. After some iterations, smallest period value is found to be 5800ps, which implies 5800/8 = 725 ps for the fastest clock period.

Second group of constraints is "external_delay" which specifies delays for input signals according to a rising edge of the driving FF's clock and for output signals according to a rising edge of the load FF's clock. These values are chosen according to clock-to-Q delays of the FFs (300ps) for the inputs and setup time required (100ps) for the FFs for the outputs.

Next constraint group specifies the retiming process. All registers except the ones to be retimed are marked as “dont_retime true”. By doing that, unnecessary registers are not prepared for retiming by the tool and retiming process becomes more efficient as computation is done with less number of registers [23].

Rest of the script contains a command that makes the tool synthesize with high effort. That makes the tool to find a better solution for the design.

Summary of the timing and area reports of the synthesized design are in Table 4.4.

**Table 4.4 :** Summary of the synthesized design.

| Maximum Clock Frequency [GHz] | # of Cells Used | Area for Cells [μm x μm] | Area for Nets [μm x μm] | Total Area [μm x μm] |
|---|---|---|---|---|
| 1,38 | 35969 | 1124 x 1124 | 693 x 693 | 1321 x 1321 |

### 4.5.2. PAR

As mentioned in the previous design, the improved design is also placed and routed with the Encounter tool of Cadence. A modified version of the former TCL script (previously done for the Lfoundry 0.15 μm CMOS process) was used for PAR. Moreover, new strategies and methods are used for the PAR of the improved filter architecture designed for the TSMC 0.18 μm CMOS process.

During Place, floorplan of the design was more critical than before. As operating frequency of the system is nearly at the limits of the process and power consumption of the system is high, power distribution becomes the main issue. To better handle power distribution, power hungry blocks like Filter-3 are located near VDD and GND pins to reduce resistance of the power lines. Besides, less power hungry blocks like Filter-1 are located farther away. In the end, floorplan of the design becomes rectangular-shaped, with one of the long edge being directly connected to VDD and GND pins of the chip. Other tools to cope with power distribution are increasing the widths of VDD and GND lines between logic gates (Metal-1 layer) and adding horizontal and vertical stripes all around the floorplan.

During the route phase, it is found that, a large number of thick stripes make routing difficult and even impossible. So within some iteration, optimum stripe width and number are found and used. At the end of routing, optimizations are performed with "optDesign" command, which makes worst paths better and area smaller.

After PAR, power analyses were made to find VDD drop and power consumption of the design. With 1.2GHz operation speed, total dynamic power consumption of the system is found as 1,826W. IR-drop graph of the design is given in Figure 4.8.

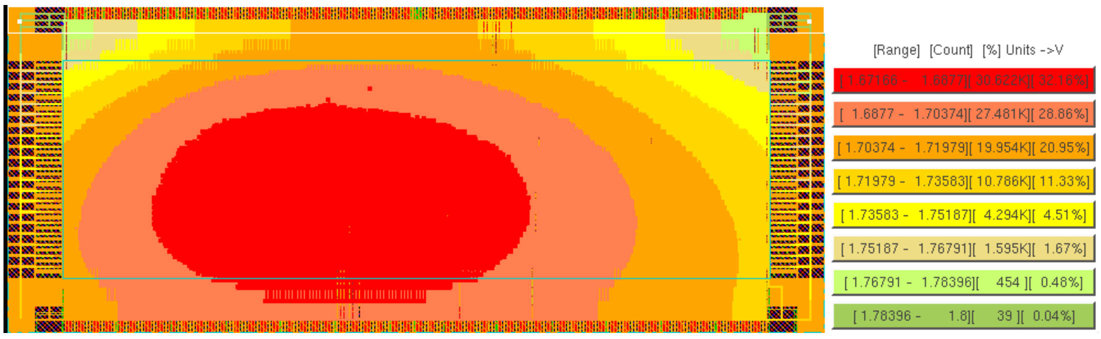


**Figure 4.8 :** IR-drop plot of the design.

As seen from Figure 4.8, worst IR-drop reduces VDD voltage from 1.8V to 1.67V. VDD and GND pins are connected to upper left and right corners of the layout. The TCL script used for PAR flow is in Appendix E.

### 4.5.3. Layout

After PAR, layout is generated and exported as a GDS-II file. Then, GDS-II file is read using Cadence's layout-XL editor and put together with other analog and digital layouts of the chip. Layout of the filters is obtained as in Figure 4.9.

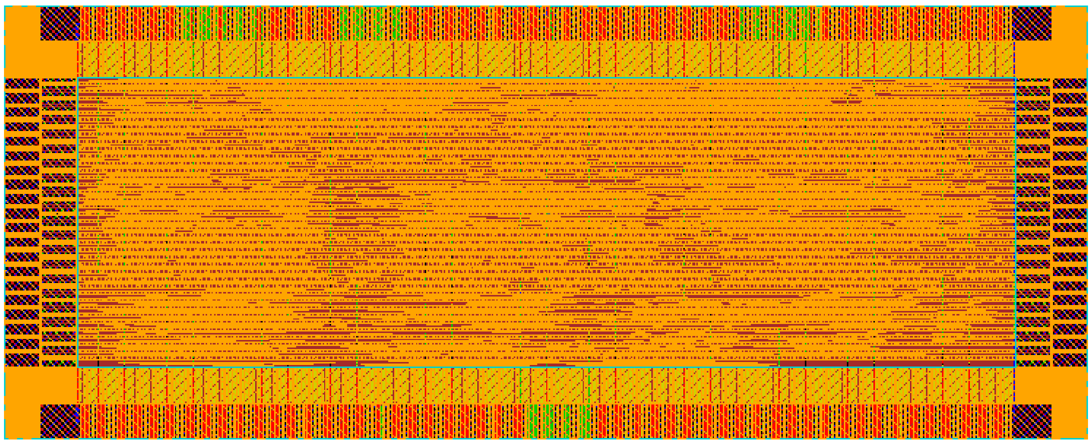


**Figure 4.9 :** Layout of the routed filters.

In Figure 4.8, cell area of the layout is 800 µm x 2600 µm and total area of the layout is 1200 µm x 3000 µm.

## 4.6. Post-PAR Simulations

After implementation, post-PAR simulations were performed to check the functionality and specifications of the routed design. Simulations are run in NCSim simulator of Cadence.

Difference of post-PAR simulations from behavioral simulations is that, post-PAR simulations contain a SDF file, which includes delays of the cells and interconnects in the simulations. Thus, simulations become more realistic. Acquired SDF file is given in Appendix E.

As mentioned before, key metrics of the system are SFDR performance and operating frequency. In addition, group delays of the filters are calculated at post-PAR simulations. Results are given in Table 4.5.

**Table 4.5 :** Group delays of filters.

| Group Delay | Filter-1 | Filter-2 | Filter-3 | Total |
|---|---|---|---|---|
| # of Filter-X clock cycles | 18 | 10 | 12 | |
| # of DAC clock cycles | 18 x 8 = 144 | 10 x 4 = 40 | 12 x 2 = 24 | 208 |

### 4.6.1. Max operating frequency

During PAR step, connection delays are added to models which reduce maximum operation frequency of the system. In order to find the new value, timing analyses are performed after the PAR step. Result of the timing analysis is given in Table 4.6

**Table 4.6 :** Result of the timing analysis at PAR step.

| Setup mode | all | Reg2reg | In2reg | Reg2out |
|---|---|---|---|---|
| WNS [ns] | -0.075 | -0.075 | 0 | -0.047 |
| TNS [ns] | -3.623 | -3.496 | 0 | -0.127 |

Values in Table 4.6 are taken from output file of the timing analysis. Acronym WNS stands for worst negative slack which gives maximum operating frequency of the system. TNS is the total negative slack which is a performance parameter and not directly effects the calculation of the operating speed. Reg2reg stands for the delay value between registers, In2reg is delay between input to register and Reg2out is the delay between register and output pin.

According to Table 4.5, worst delay path between the registers is increased by 75ps and delay between a register to an output is also increased by 47ps. In this situation, worst value is taken into consideration and 75ps is used for calculations. When this value is added to 725ps, which comes from cell delays (found at synthesis step), worst delay of the system becomes 800ps. That makes maximum operating frequency 1,25GHz under typical conditions.

### 4.6.2. SFDR of output

SFDR of the output signal is expected to be equal to the one found with MATLAB models. Reason of this expectation is that, no functional change happens during the implementation step but only physical effects like adding delays occur. To show if the filter operation is still as desired, a post-PAR simulation is made. Verilog testbench is written so that, signal at the filter output is written to file in a format readable for MATLAB. Then, file containing the post-PAR simulation data is imported to MATLAB and compared with the MATLAB model output. At the end, it is seen that both results exactly match. Thus, SFDR values given in Section 4.2 can be taken as the result of the post-PAR SFDR result. No more plots will be given here for that reason.

# 5. DESING VERIFICATION USING AN FPGA

Although only oversampling filters and modulators are being mentioned so far, the complete digital system also contains a clock divider, two RAMDACs, two binary to thermometer segment encoders with a circuit for DAC output mode selection (NRZ, RZ, PM) and a control block bound to a Serial Peripheral Interface (SPI) block. The details of all these blocks are given in Appendix C. Block diagram of the complete digital system is given in Figure 5.1.

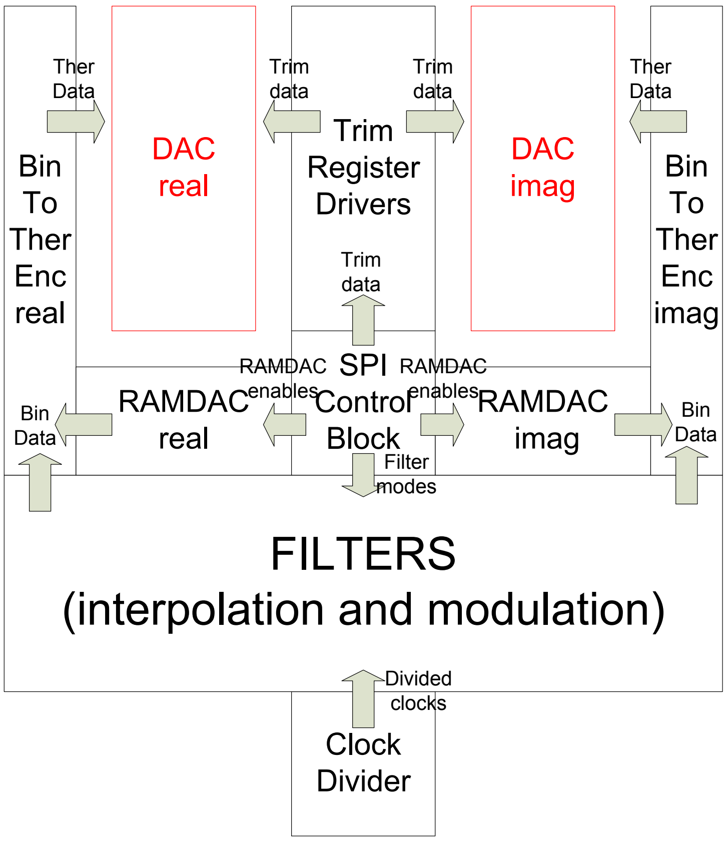


**Figure 5.1 :** Block diagram of the digital system.

In this section, the FPGA verification of the clock divider, digital filters and modulators, control block, RAMDAC and SPI is presented. SPI is the interface sets the operating modes of the filters and captures the data at filter outputs for serial read-back.

## 5.1. Test Cases

During test case preparation, it is aimed to start with a simplest case and continue with more complex ones. Moreover, attention is given to make the test flow as similar as possible to the test flow of a real DAC chip.

### 5.1.1. Read and write operations on registers

Most basic instruction that can be given to SPI interface is reading from and writing to a register. These operations are required to set the operating modes of the filters and to read filter outputs from the SPI interface. Besides, trimming current cells of the DAC and changing offset and gains of the analog blocks are possible using the SPI interface.

### 5.1.2. Fuse blowing

Using the SPI, it is possible to blow fuses of the chip. It is done by writing desired value to mask register of the fuse and then, by sending a fuse blow command. After blowing a fuse, it is also possible to check the value written to the fuse by reading the register that samples the fuse output via SPI.

### 5.1.3. Setting filter modes

Setting Filter modes is a simple write operation to a register. That register is named as “Filter Reg 0”. That is a 16-bit register that contains filter mode data for three interpolation filters and that sets 2x, 4x, 8x or no interpolation as interpolation modes. Also, it contains RAMDAC w enable, RAMDAC i enable and RAMDAC r enable registers which will be discussed in section 5.1.6. Explanative block diagram for “Filter Reg 0” is in Table 5.1.

**Table 5.1 :** Filter Reg 0 register

| Reserved | | | RAMDAC w enable | RAMDAC i enable | RAMDAC r enable | Int. Mode (1:0) | | Filter 1 (1:0) | | Filter 2 (2:0) | | | Filter 3 (2:0) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X | X | X | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

According to Table 5.1, the value of Int. Mode register is 3, which refers to 8x interpolation. 2 is for 4x and 1 is for 2x modes. If this value is 0, no interpolation is performed and the input data is directly sent to the DAC.

Filter modes are as they are explained in section 3.1.3. In the above example, Filter 1 is in mode 2, Filter 2 is in mode 2 also and Filter 3 is in mode 1.

After setting filter modes, parallel data to filters is applied and the input of the DAC is observed. If the DAC input is the same as expected, setting filter mode is successful.

### 5.1.4. Applying 16-bit parallel data

After setting different filter modes, 16-bit data is applied to the input and filters are allowed to work. At the end, some or all output data is compared with MATLAB simulations.

### 5.1.5. Reading filter outputs

For a real application, it is not possible to read the digital data that filters produce in a DAC chip. In order to test filter outputs, some registers controlled by the SPI is inserted to chip. By using these registers, data at filter outputs can be sampled and read back using the SPI. Those registers have the same name with the filter outputs.

To read data at filter outputs, first it is necessary to stop the filter clocks to prevent data from being changed. Then, a write command is sent to the appropriate register. The register samples the filter output. After that, a read command is sent and the sampled data is read with the SPI.

### 5.1.6. Filling and reading the RAMDAC

The term RAMDAC is used for the register matrix that is placed as a barrel shifter, which has 16-bit parallel data width and a selectable depth (before synthesis). That matrix is used to send very high speed data to the DAC internally. By doing this, it is desired to test the DAC’s analog performance at the limit of the sampling speed.

Filling the RAMDAC requires a write operation to registers “RAMDAC_in_r” and “RAMDAC_in_i” and a shift operation which makes RAMDAC sample the input given. But initially, “RAMDAC w enable” register should be set to enable RAMDAC registers. Then, one or both the “RAMDAC r enable” and “RAMDAC i enable” registers should be set to enable the real and imaginary inputs to the DAC.

After completion of the write operation, “RAMDAC w enable” should be reset, in order not to change the value during normal operation. If reading the RAMDAC data is desired, “RAMDAC_out_r” and “RAMDAC_out_i” registers can be read after giving a successful write command to them. If a shift and another write are applied, next data is ready to read at related RAMDAC_out register.

## 5.2. Behavioral Tests

### 5.2.1. Introduction & tools

Behavioral FPGA tests are done using the ISIM simulator of Xilinx. A testbench is written to test the functionality. Using the testbench, inputs of the systems are driven according to verify the test cases explained in 5.1.

Testbench has two tasks, which generate serial data that is used to drive the SPI. Testbench uses tasks described below to apply serial data with a high level interface.

Task SPI_CB_TASK has three inputs. First input is a string which defines the command type. That can be write, read, fuseblow or reserved. Second input has 11 bit width, which specifies the address associated with the command to be performed. Third input is an 8-bit data associated with the command. Purpose of this task is converting those inputs to serial and applying them to the SPI interface.

Task SPI_CB_TASK_24 has similar inputs with 24-bit data. This task is used when writing to and reading from the shift registers (trim registers) with 24-bit width. It calls SPI_CB_TASK 3 times with sending its data from MSB 8 bits to LSB 8 bits.

Testbench also writes states of inputs and outputs to a file in order to compare them with Logic Analyzer outputs to check errors.

### 5.2.2. Read and write on registers

In order to test read and write operations in behavioral domain, SPI_CB_TASK_24 task is used. Firstly, write operation is tested by the following line in testbench.

```
SPI_CB_TASK_24("write", 11'h001, 24'hACABA0);
```

Task call given above generates serial data that includes write command to address “1” and with 24-bit data “ACABA0”. After it is applied, the content of the shift register with address “1” is checked and it is seen that it is written correctly.

After writing, similar command is given for read operation. Associated task call is given below.

```
SPI_CB_TASK_24("read", 11'h001, 24'h000000);
```

Read task contains a command type and register address “1”. 24-bit data should also be applied but its content is not important. After reading is completed, bits read are examined and seen that they are correct. Moreover, content of the shift register is checked again and seen that it is not corrupted.

### 5.2.3. Fuse blow tests

Fuse blow test is a four step procedure which includes writing data to mask register, sending fuse blow command, sampling data at the output of the fuse and in the end, reading data at the output of the fuse. Associated task calls are given below.

```
SPI_CB_TASK_24("write", 11'h002, 24'h800000);
SPI_CB_TASK("fuseblow", 11'h003, 8'h00);
SPI_CB_TASK("write", 11'h003, 8'h00);
SPI_CB_TASK_24("read", 11'h003, 24'h000000);
```

First task is just like writing to an ordinary shift register. But this time, there is a fuse at the output. Second task is a special command that blows the fuse at a given address. In this example, the fuse address is “3”. Third task looks like an ordinary write command but with a difference: there is a parallel-input shift register with address “3”. When third task is called, Register at address “3” samples its 24-bit input data which is connected to output of the fuse. Bu doing that, the fuse data is prepared for being sent serially. The last task is an ordinary read task, which outputs 24-bit data serially.

After simulation, it is seen that the fuse is blown according to the applied bit-stream.

### 5.2.4. Filter mode tests

Setting the Filter modes is performed by writing appropriate mode data to “Filter Reg 0” register resides at address “301”. As operation is so simple and will be repeated in the following section, it is not explained here. When data is written to a register, filter modes are set correctly according to the behavioral simulation.

### 5.2.5. Applying 16-bit data

Applying 16-bit data to system is done by reading a 16-bit sine data from a file generated by MATLAB. Data read is applied to parallel data inputs of the system synchronous with the clock input, clkin. At the same time, filter outputs are written to a file which will be opened with MATLAB and its SFDR value will be checked.

After the simulation, output waveforms obtained from filter outputs are examined and it is seen that the filter operations are accurate. Moreover, data points are loaded to MATLAB to see if a difference is present or not. It is seen that same results with MATLAB models are obtained.

### 5.2.6. Reading filter outputs

Reading filter outputs is a two phase procedure which contains a sample command and a read command. Sample command makes shift register sample the associated filter output and read command performs a conventional read operation. An example commands for reading Filter-1 outputs are given below.

```
SPI_CB_TASK("write", 11'd378, 8'h00);

SPI_CB_TASK("write", 11'd379, 8'h00);

SPI_CB_TASK_16("read", 11'd378, 16'h0000);

SPI_CB_TASK_16("read", 11'd379, 16'h0000);
```

Register in address 378 samples real output of Filter-1 and register in address 379 samples imaginary output of Filter-1.

### 5.2.7. Filling and reading the RAMDAC

To start writing to the RAMDAC, associated enable bits in Filter Reg 0 should be set. Following command is sent.

```
SPI_CB_TASK_16("write", 11'd301, 16'h1F00);
```

With this command, writing to both real and imaginary RAMDACs is enabled. After that, writing to registers is performed. Following two commands writes data to “RAMDAC_in_r” and “RAMDAC_in_i” registers.

```
SPI_CB_TASK_16("write", 11'd374, 16'h0001);
SPI_CB_TASK_16("write", 11'd375, 16'hA000);
```

After that, a clock edge is given from clkin input to make RAMDAC registers shift the given data. Associated code part is below.

```
# 10 clkin = 0;
# 10 clkin = 1;
# 10 clkin = 0;
```

Now, the RAMDAC is ready to sample a new data. Above data writing process is repeated 3 more times with different data to fill RAMDAC. In the end, content of RAMDAC registers are checked and it is seen that data is written to correct locations.

After filling is successful, reading the RAMDAC is tested. First, write enable of RAMDAC is disabled by giving following command.

```
SPI_CB_TASK_16("write", 11'd301, 16'h0F00);
```

Then, RAMDAC data are sampled to “RAMDAC_out_r” and “RAMDAC_out_i” registers for reading. Finally, registers are read by applying read commands. Mentioned commands are given below.

```
SPI_CB_TASK("write", 11'd376, 8'h00);
SPI_CB_TASK("write", 11'd377, 8'h00);
SPI_CB_TASK_16("read", 11'd376, 16'h0000);
SPI_CB_TASK_16("read", 11'd377, 16'h0000);
```

Within first read, first data that is written to RAMDAC is taken out. Therefore, it can be said that RAMDAC has FIFO architecture. To read second data, RAMDAC data should be shifted. This is done with the following code.

```
# 10 clkin = 0;
# 10 clkin = 1;
# 10 clkin = 0;
```

To read second data and the rest; sampling, reading and shifting is repeated. During tests, it is seen that data is read correctly and register contents are not corrupted.

## 5.3. Testing with Logic Analyzer

Once behavioral tests succeed, Verilog code of the whole system is implemented using an FPGA.

### 5.3.1. Test setup

Hardware test setup contains a Logic Analyzer that also has pattern generator functionality and an FPGA that implements the system under test. Block diagram of the hardware test setup is in Figure 5.2.

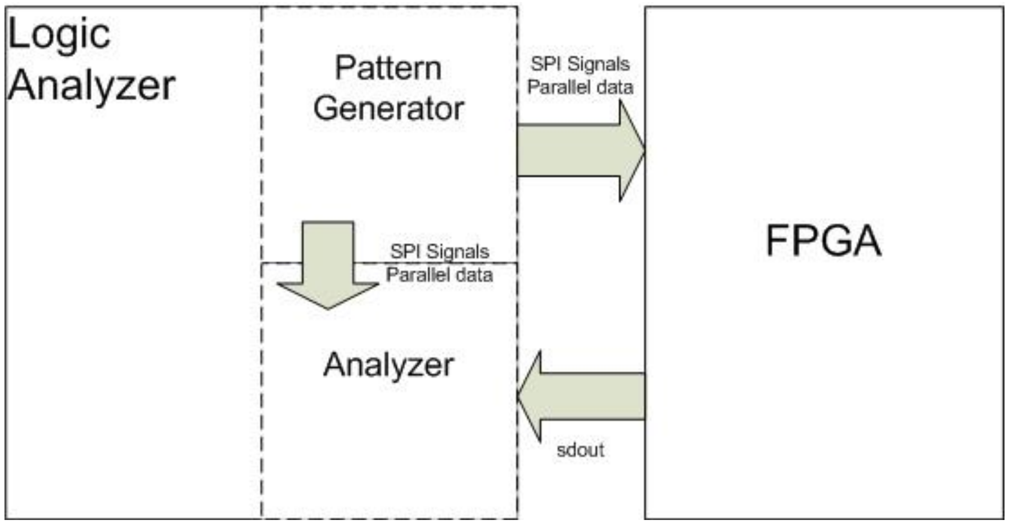


**Figure 5.2 :** Hardware test setup.

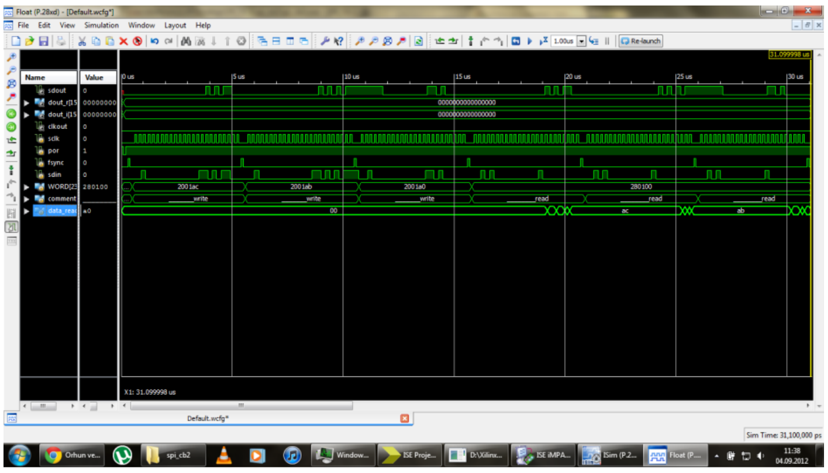
For consistency, it is important to use the same input stream for both behavioral and hardware tests. In order to do that, input stream for FPGA tests are also generated by behavioral testbench. That is done by the following code part.

```
always@(por, fsync, sclk, sdin, clkin)begin
        $fwrite(file3,"%h\n",{3'h0, clkin, sdin, sclk, fsync, por});
end
```

Above code writes inputs of the system to a file when one of them is changed during behavioral tests. Then, file is read by pattern generator to generate the same inputs for Hardware tests. By this test setup, it is possible to take same outputs from both behavioral and hardware tests.

### 5.3.2. Basic comparison

For basic functionalities like writing to/reading from a register, comparing behavioral simulation and Logic Analyzer output waveforms is a fast way of detecting equivalency. In Figure 5.3, behavioral simulation and Logic Analyzer output waveforms containing one write and one read command are given.



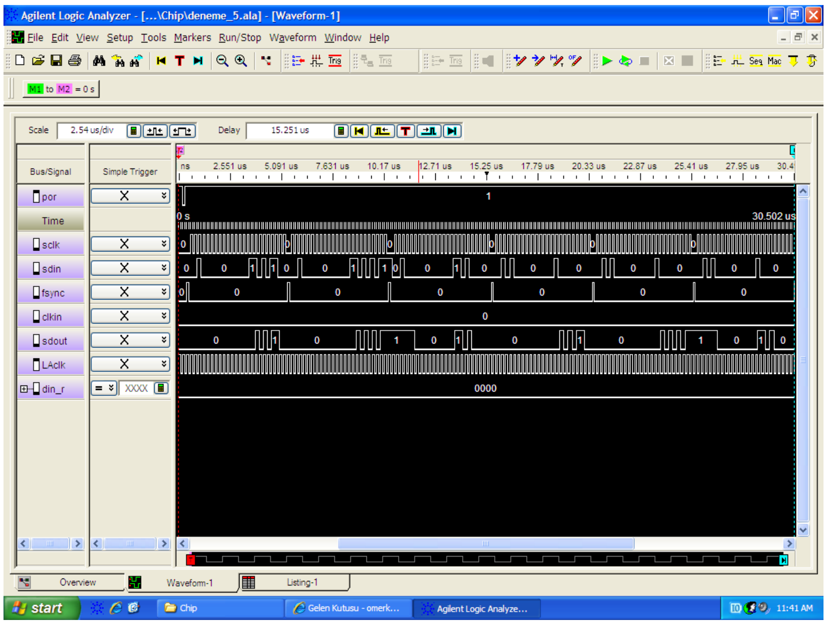


**Figure 5.3 :** Behavioral simulation (Upper) and Logic Analyzer output (Lower) waveforms.

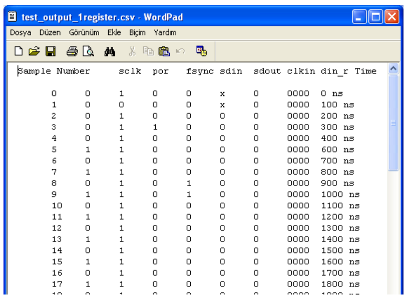
According to Figure 5.3, it can be said that waveforms of output signal “sdout” are the same and the hardware and the behavioral models have the same functionality.

### 5.3.3. Automatic comparison

Comparing waveforms of behavioral simulation and Logic Analyzer outputs is not difficult for short simulations with a small amount of data. However, when complexity increases and data to be compared become huge, it is indispensible to make the comparison process automatic. Therefore, a C program is used to compare outputs of the behavioral simulation and the Logic Analyzer.

First step for comparison is generating the data to be compared with a proper format. It is decided to use a format similar to Logic Analyzer export file.

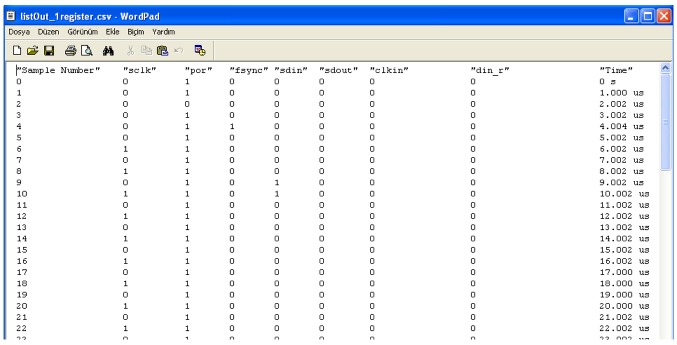
Outputs of the Logic Analyzer are written to a file by selecting export option from file tab. Outputs of behavioral simulation and Logic Analyzer are given in Figure 5.4.



```
Sample Number       sclk  por    fsync sdin   sdout clkin din_r Time

        0     0     1     0     0     x     0     0000  0 ns
        1     0     0     0     0     x     0     0000  100 ns
        2     0     1     0     0     0     0     0000  200 ns
        3     0     1     1     0     0     0     0000  300 ns
        4     0     1     0     0     0     0     0000  400 ns
        5     1     1     0     0     0     0     0000  600 ns
        6     0     1     0     0     0     0     0000  700 ns
        7     1     1     0     0     0     0     0000  800 ns
        8     0     1     0     1     0     0     0000  900 ns
        9     1     1     0     1     0     0     0000  1000 ns
        10    0     1     0     0     0     0     0000  1100 ns
        11    1     1     0     0     0     0     0000  1200 ns
        12    0     1     0     0     0     0     0000  1300 ns
        13    1     1     0     0     0     0     0000  1400 ns
        14    0     1     0     0     0     0     0000  1500 ns
        15    1     1     0     0     0     0     0000  1600 ns
        16    0     1     0     0     0     0     0000  1700 ns
        17    1     1     0     0     0     0     0000  1800 ns
```

(a)

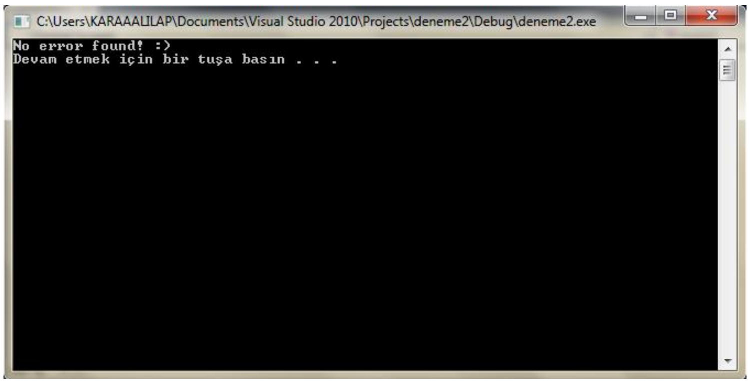


```
"Sample Number"    "sclk"    "por"   "fsync" "sdin"  "sdout" "clkin"           "din_r"           "Time"
0                  0         1       0       0       0       0                 0                 0 s
1                  0         1       0       0       0       0                 0                 1.000 us
2                  0         0       0       0       0       0                 0                 2.002 us
3                  0         1       0       0       0       0                 0                 3.002 us
4                  0         1       1       0       0       0                 0                 4.004 us
5                  0         1       0       0       0       0                 0                 5.002 us
6                  1         1       0       0       0       0                 0                 6.002 us
7                  0         1       0       0       0       0                 0                 7.002 us
8                  1         1       0       0       0       0                 0                 8.002 us
9                  0         1       0       1       0       0                 0                 9.002 us
10                 1         1       0       1       0       0                 0                 10.002 us
11                 0         1       0       0       0       0                 0                 11.002 us
12                 1         1       0       0       0       0                 0                 12.002 us
13                 0         1       0       0       0       0                 0                 13.002 us
14                 1         1       0       0       0       0                 0                 14.002 us
15                 0         1       0       0       0       0                 0                 15.002 us
16                 1         1       0       0       0       0                 0                 16.002 us
17                 0         1       0       0       0       0                 0                 17.000 us
18                 1         1       0       0       0       0                 0                 18.000 us
19                 0         1       0       0       0       0                 0                 19.000 us
20                 1         1       0       0       0       0                 0                 20.000 us
21                 0         1       0       0       0       0                 0                 21.002 us
22                 1         1       0       0       0       0                 0                 22.002 us
```

(b)

**Figure 5.4 :** Output files of simulation (a) and Logic Analyzer (b) for comparison

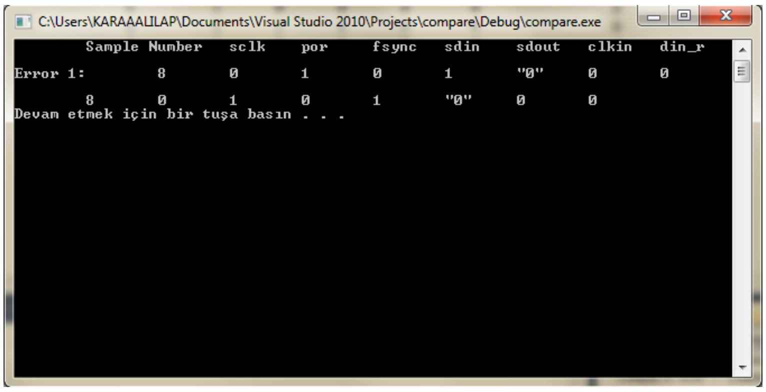
Files in Figure 5.4 are loaded to a C program which reads, compares and determines the equality. After every simulation, outputs are compared and C program gives the output seen in Figure 5.5.



```
No error found! :)
Devam etmek için bir tuşa basın . . .
```

**Figure 5.5 :** Output of C program when input files are equivalent.

C program also shows non-equal lines of input files which eases debugging. An example of C program output when the input files are not equivalent is in Figure 5.6.



```
         Sample Number   sclk     por      fsync    sdin     sdout    clkin    din_r
Error 1:         8        0        1        0        1        "0"      0        0

         8       0        1        0        1        "0"      0        0
Devam etmek için bir tuşa basın . . .
```

**Figure 5.6 :** Output of C program when input files are not equivalent.

# 6. CO-SIMULATION WITH AN ANALOG DAC

The digital interpolation system with modulation feature that forms the content of this work is designed to work with a DAC. Thus, this section is devoted to use the digital system with a DAC (designed in a different study). The design automation environment that supports the simulation of a digital system and an analog DAC is also included in this section.

## 6.1. Presentation of the DAC Taken from a Different Work

The DAC is a high speed, 16 bit DAC with segmented (31 unary scaled MSB, 11 binary scaled LSB) current cells. It is designed to optimize the SFDR performance of a standard architecture with a PMOS switch and double cascode current source. The DAC draws 20 mA full scale current from a 3.3 V supply and is expected to operate well above 1.28 GSPS. With a 12.5 Ω load to ground, this DAC generates a 500 mV peak to peak differential signal. The current steering architecture of the DAC is given in Figure 6.1.

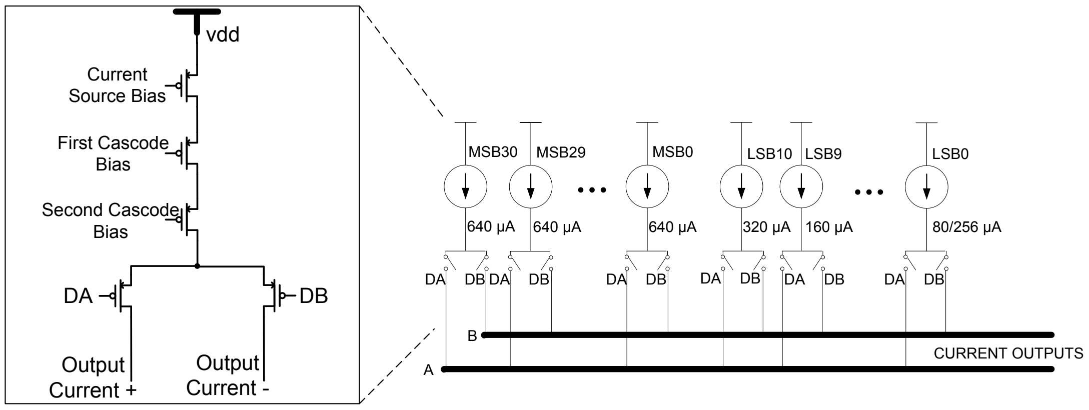
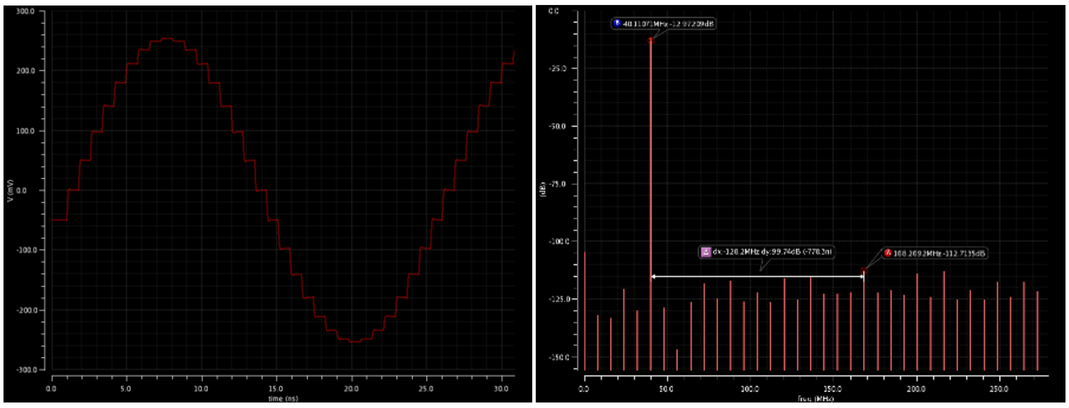


**Figure 6.1 :** Current steering architecture used in the DAC.

The unit cell of the DAC is also shown in Figure 6.1. This unit cell draws 80 uA current from a 3.3 V supply. The unary cells include eight unit cells.

At 1.28 GHz clock speed, using 0.5V peak to peak voltage swing on a 25 Ω differential load, the DAC SFDR is 99 dB and 78 dB at 40 MHz and 240 MHz output frequencies respectively. 99 dB SFDR (DAC only) plot is given in Figure 6.2.



(a) (b)

**Figure 6.2 :** Output of the DAC in time (a) and frequency (b) domain.

## 6.2. Co-simulation of the Filter and the DAC

In order to simulate the digital system with the DAC, a simulation schematic is generated based on the block diagram in Figure 6.3.

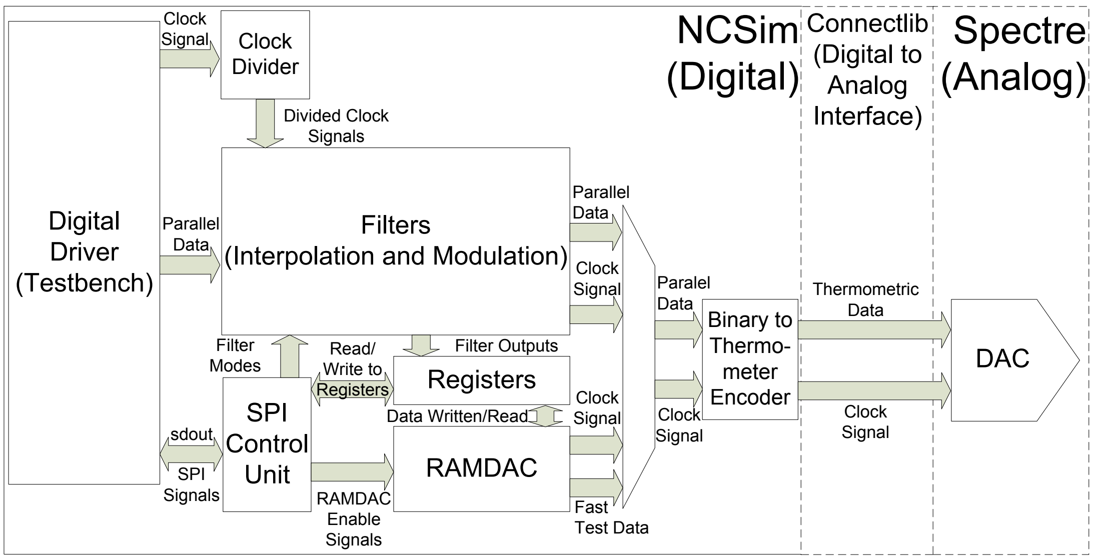
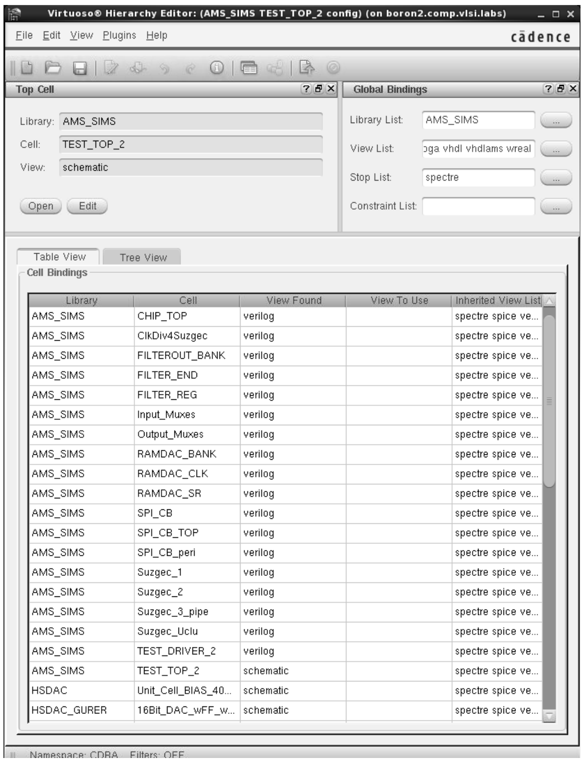


**Figure 6.3 :** Block diagram of simulated blocks and simulation tools.

### 6.2.1. AMS simulator

This simulator can simulate a system which contains analog and digital blocks. AMS requires a configuration file which shows calculation engines for the blocks to be simulated. An example of an AMS configuration file is given in Figure 6.4.

As seen in Figure 6.4, “verilog” views are present for digital blocks whereas “spectre” views are present for analog blocks. Apart from these two, “schematic” views are the top level models for both views.


Virtuoso® Hierarchy Editor: (AMS_SIMS TEST_TOP_2 config) (on boron2.comp.vlsi.labs)

File Edit View Plugins Help

**Top Cell**

Library: AMS_SIMS
Cell: TEST_TOP_2
View: schematic

**Global Bindings**

Library List: AMS_SIMS
View List: ›ga vhdl vhdlams wreal
Stop List: spectre
Constraint List:

Table View | Tree View

**Cell Bindings**

| Library | Cell | View Found | View To Use | Inherited View List |
|---|---|---|---|---|
| AMS_SIMS | CHIP_TOP | verilog | | spectre spice ve... |
| AMS_SIMS | ClkDiv4Suzgec | verilog | | spectre spice ve... |
| AMS_SIMS | FILTEROUT_BANK | verilog | | spectre spice ve... |
| AMS_SIMS | FILTER_END | verilog | | spectre spice ve... |
| AMS_SIMS | FILTER_REG | verilog | | spectre spice ve... |
| AMS_SIMS | Input_Muxes | verilog | | spectre spice ve... |
| AMS_SIMS | Output_Muxes | verilog | | spectre spice ve... |
| AMS_SIMS | RAMDAC_BANK | verilog | | spectre spice ve... |
| AMS_SIMS | RAMDAC_CLK | verilog | | spectre spice ve... |
| AMS_SIMS | RAMDAC_SR | verilog | | spectre spice ve... |
| AMS_SIMS | SPI_CB | verilog | | spectre spice ve... |
| AMS_SIMS | SPI_CB_TOP | verilog | | spectre spice ve... |
| AMS_SIMS | SPI_CB_peri | verilog | | spectre spice ve... |
| AMS_SIMS | Suzgec_1 | verilog | | spectre spice ve... |
| AMS_SIMS | Suzgec_2 | verilog | | spectre spice ve... |
| AMS_SIMS | Suzgec_3_pipe | verilog | | spectre spice ve... |
| AMS_SIMS | Suzgec_Uclu | verilog | | spectre spice ve... |
| AMS_SIMS | TEST_DRIVER_2 | verilog | | spectre spice ve... |
| AMS_SIMS | TEST_TOP_2 | schematic | | spectre spice ve... |
| HSDAC | Unit_Cell_BIAS_40... | schematic | | spectre spice ve... |
| HSDAC_GURER | 16Bit_DAC_wFF_w... | schematic | | spectre spice ve... |

Namespace: CDBA Filters: OFF

**Figure 6.4 :** Configuration file for AMS.

Setting views for blocks makes AMS to choose the appropriate simulation engine for these views. AMS automatically selects NCSim for simulating Verilog models (of digital blocks) and Spectre for simulating Spectre models (of analog blocks).

### 6.2.2. Applying inputs and setting modes

Applying digital inputs to digital blocks is done by a testbench-like driver written in Verilog. This block generates signals like the SPI signals, the parallel data and the clock signal. Then, these signals are applied to the digital system to be simulated.

First phase of the simulation is setting the filter modes. This is done by writing appropriate bits to “Filter reg 0” via SPI. For this simulation, 8x interpolation with no modulation is chosen. After that, 16-bit parallel data and clock is applied.

Outputs of the digital system are connected to the DAC. But, as outputs of the digital system consists of logic 1s and 0s, they should be converted to analog levels. Rules of this conversation procedure are defined in a rules file, in library “Connectlib”.

### 6.2.3. Conversion interface: connectlib

The interface that converts logical signals to analog signals is managed by rules in Connectlib. For that conversion, it is necessary to define voltage levels for logic 1 and 0. Besides; transition time, on and off resistances of interface should be set. Values used in this simulation are given in Table 6.1.

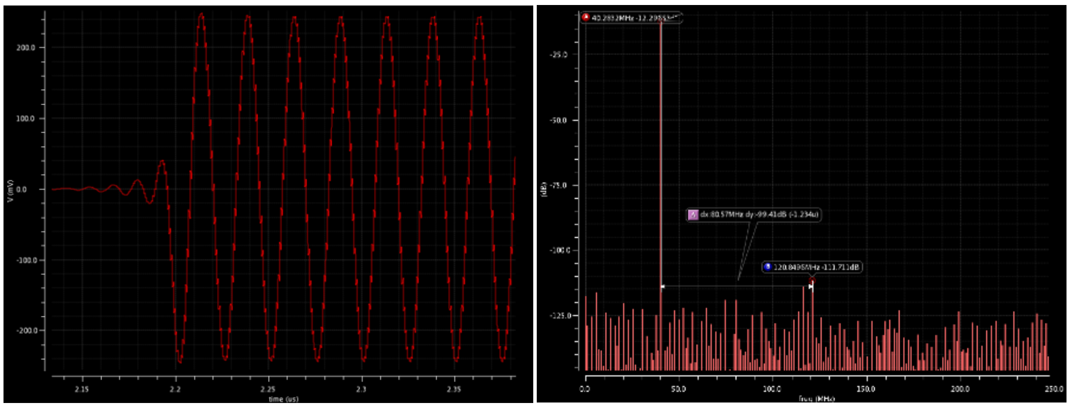
**Table 6.1 :** Conversion parameters for the interface.

| Vsup | Vthi | Vtlo | Tr | Rlo | Rhi | Rx | Rz |
|---|---|---|---|---|---|---|---|
| 1.8 | 1.2 | 0.6 | 40p | 200 | 200 | 40 | 10M |

In Table 6.1, Vsup stands for supply voltage. It is the value equivalent to logic 1. Vthi and Vtlo are used when analog values are converted to digital. So, they are left at typical values given in Connectlib. Tr is rise time, chosen as 40 ps as in digital simulations. Rlo and Rhi are resistance values when output is logic 0 and 1 respectively. These values are left unchanged. Rx and Rz are resistances when output is logic-X or high-Z. As this simulation does not contain these values at normal operation, these values are kept as suggested values for TSMC 0.18 μm technology.

## 6.3. Simulation Results

After simulation, voltage difference between two differential DAC outputs is plotted in both time and in frequency domains. Resulting plots are in Figure 6.5.



(a) (b)

**Figure 6.5 :** Time (a) and frequency (b) plot of signal at DAC output.

In Figure 6.5, it is seen that 99 dB SFDR is obtained from DAC outputs. This value is less than the SFDR value of the filter (102 dB) for this frequency (40 MHz). As a result, it can be argued that digital filtering system does not worsen SFDR performance of the DAC and SFDR of the output is determined by the DAC, not by the digital system.

Details of the AMS simulations are given in Appendix A.

# 7. CONCLUSIONS AND RECOMMENDATIONS

## 7.1. Results and Conclusions

In this study, a digital interpolation and modulation system is designed for a two channel high performance communications DAC. A MATLAB model of an interpolation and modulation system is created and simulations are performed. After desired specifications are met in MATLAB environment, a digital behavioral design is done in Verilog language to reflect the MATLAB model. The verilog code is synthesized using TSMC 0.18 μm standard gate library (Artisan) and place and route (PAR) operations are performed. Delays are extracted and post-PAR simulations are done. Extracted delay information is also used in mixed mode co-simulations with a DAC.

Performance Summary of the digital filters is given in Table 7.1.

**Table 7.1 :** Summary of system specifications.

| | F1 | F2 | F3 | Total |
|---|---|---|---|---|
| Filter order | 62 | 22 | 22 | - |
| Number of Coefficients | 16 | 6 | 6 | - |
| Bit length of Coeff.s | 18 | 16 | 16 | - |
| Number of Modes | 4 | 8 | 8 | 41 |
| Pass-band Ripple [dB] | $2x10^{-4}$ | $2x10^{-4}$ | $2x10^{-4}$ | $6x10^{-4}$ |
| Group Delay [clk] | 18 (144) | 10 (40) | 12 (24) | 208 |
| SFDR [dB] | 98.3 | 99.7 | 99.7 | 99 |
| SNR [dB] | 95.4 | 94.6 | 94.6 | 93.4 |
| Area [$mm^2$] (dual channel) | - | - | - | 1.2 x 3 |
| Power [W] (dual channel) | - | - | - | 1.826 |
| Speed (dual channel) | - | - | - | 1.2 GSPS |
| Gate Count (at synthesis) | - | - | - | 35969 |

Comparison of the performance of our design with existing designs can be seen in Table 7.2.

**Table 7.2 :** Comparison of the work with best products.

| Company | Part name | SFDR [dB] | Max. data rate [MSPS] | Power in datasheet | Power for Comparison |
|---|---|---|---|---|---|
| Analog Devices | AD9122[4] | 85 (@ %80 bandwidth) | 1230 | 1.55 W (@1.2 GSPS) | 1.55 W (@1.2 GSPS) |
| MAXIM | MAX5898[8] | 95 (unknown bandwidth) | 500 | 702 mW (@500 MSPS) | 1.685 W (@1.2 GSPS) |
| TI | DAC5689[9] | 80 (@ %80 bandwidth) | 800 | 774 mW (@500 MSPS) | 1.858 W (@1.2 GSPS) |
| This Work | Improved Design | 99 (@ %80 bandwidth) | 1200 | 1.826W (@1.2 GSPS) | 1.826 W (@1.2 GSPS) |

In addition to the interpolation and modulation system, digital blocks like the decoders, the SPI, the control block, the RAMDAC and the register blocks are added to complete the digital system. The digital system also sets the output mode (NRZ, RZ, and PM) of the DAC. The complete system is verified in an FPGA environment.

## 7.2. Recommendations and Future Work

Future work can include the tape out and the testing of the interpolating and modulating dual channel DAC chip. The power and SFDR trade-off of the digital block can be characterized in various CMOS technologies. Scan technology can be added and its effects on speed can be analyzed. Techniques can be explored to reduce the power consumption and to balance the self-heating between real DAC and imaginary DAC.

## REFERENCES

[1] **Maloberti, F.** (2007). *Data Converters.* Dordrecht: Springer.

[2] **Simić, I.S.** (2007). Evolution of Mobile Base Station Architectures. *Microwave Review*, vol. 13, no.1, June 2007 pp.29-34.

[3] **Kester, W.** (2009). Oversampling Interpolating DACs. Data retrieved: 14.09.2011 URL: http://www.analog.com/static/imported-files/tutorials/MT-017.pdf

[4] **Analog Devices.** (2009). AD9122: Dual, 16-Bit, 1230 MSPS, TxDAC+ Digital-to-Analog Converter. Product Datasheet. Norwood, MA.

[5] **Analog Devices.** (2010). AD9148: Quad, 16-Bit, 1 GSPS, TxDAC+ Digital-to-Analog Converter. Product Datasheet. Norwood, MA.

[6] **Analog Devices.** (2007). AD9776-78-79: Dual, 12-/14-/16-Bit, 1 GSPS Digital-to-Analog Converters. Product Datasheet. Norwood, MA.

[7] **Analog Devices.** (2005). AD9786: 16-Bit, 200 MSPS/500 MSPS TxDAC+ with 2x/4x/8x Interpolation and Signal Processing. Product Datasheet. Norwood, MA.

[8] **Maxim.** (2010). MAX5898: 16-Bit, 500 Msps, Interpoalting and Modulating Dual DAC with Interleaved LVDS Inputs. Product Datasheet. Sunnyvale, CA.

[9] **Texas Instruments.** (2009). DAC5689: 16-Bit, 800 MSPS, 2x-8x Interpolating Dual-Channel Digital-to-Analog Converter. Product Datasheet. Dallas, Texas.

[10] **NXP Semiconductors.** (2010). DAC1408D650: Dual 14-Bit DAC; up to 650 MSPS; 2x, 4x or 8x interpolating with JESD204A interface. Product Datasheet. URL: http://www.nxp.com/documents/data_sheet/DAC1408D650.pdf

[11] **Texas Instruments.** (2011). DAC3482: Dual-Channel, 16-Bit, 1.25 GSPS, Digital-to-Analog Converter (DAC). Product Datasheet. Dallas, Texas.

[12] **Texas Instruments.** (2011). DAC5682Z: 16-Bit, 1.0 GSPS 2x-4x Interpolating Dual-Channel Digital-to-Analog Converter (DAC). Product Datasheet. Dallas, Texas.

[13] **Analog Devices.** (2009). AD9739: 16-Bit, 2.5 GSPS, RF Digital-to-Analog Converter. Product Datasheet. Norwood, MA.

[14] **Schniter, P.** (n.d.). Polyphase Interpolation. Data retrieved: 17.06.2011 URL: http://cnx.org/content/m10431/2.11/

[15] **Sokullu, H.** (2011). *Yarım Bant Sayısal Filtre Tasarımı*, BSc. Thesis, ITU, Istanbul.

[16] **Kotteri, K.A., Bell, A.E. and Carletta J.E.** (2003). Quantized FIR Filter Design Using Compensating Zeros. *IEEE Signal Processing Magazine*, vol. 20, no.6, November 2003 pp.60-67.

[17] **Sklansky, J.** (1960). Conditional-Sum Addition Logic. *IRE Transactions on Electronic Computers.* vol. EC-9, is. 2, pp.226-231.

[18] **Kogge, P.M. and Stone, H.S.** (1973). A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations. *IEEE Transactions on Computers.* vol. C-22, is. 8, pp.786-793.

[19] **Brent, R.P. and Kung, HT.** (1982). A Regular Layout for Parallel Adders. *IEEE Transactions on Computers.* vol. C-31, is. 3, pp.260-264.

[20] **Han, T. and Carlson, D.A.** (1987). Fast Area-Efficient VLSI Adders. *IEEE 8th Symposium on Computer Arithmetic (ARITH).* pp.49-56.

[21] **Knowles, S.** (2001). A Family of Adders. *IEEE 15th Symposium on Computer Arithmetic (ARITH).* pp.277-281.

[22] **Roma, N., Dias, T. and Sousa, L.** (2003). Fast Adder Architectures: Modeling and Experimental Evaluation. *XVII Conference on Design of Circuits and Integrated Systems – DSIC'03.* pp.367-372.

[23] **Cadence.** (2007). Using Encounter® RTL Compiler, *Product Manual,* San Jose, CA.

## APPENDICES

## APPENDIX A

AMS simulation is like an ordinary spectre simulation in Cadence with some minor changes. In this section, these changes and details of AMS simulations made in this work are given.

### A.1. Importing CORELIB

Post-PAR verilog file contains logic gates defined in Artisan library. For simulation, verilog models of these gates should be defined to simulator.

A new library named CORELIB is created with Cadence's library manager. Then, in icfb window import-> verilog is chosen. After that following lines are filled in the opened window: Target Library Name: CORELIB, Verilog Files to Import: tsmc18.v, Import Module as: schematic and functional, Verilog Cell Modules: Import.

A new library named CORELIB is filled with gate models defined in tsmc18.v file.

### A.2. Importing Verilog Model File and SDF in Library Editor

To import the post-PAR verilog file, it should be renamed first. Name should be verilog.v. Moreover, verilog file should call SDF file with "$sdf_annotate" command. Syntax is:

```
initial $sdf_annotate("address_of_sdf_file");
```

After that, a new cellview is creted with File -> new -> cellview. At last, in icfb window, import-> verilog is chosen. After that following lines are filled in the opened window: Target Library Name: SIM_LIB, Reference Libraries: CORELIB, Verilog Files to Import: verilog.v, Import Module as: schematic and functional, Verilog Cell Modules: Import.

### A.3. Creating Test Schematic

After importin verilog files, a test schematic is created with file -> new ->cellview. Test schematic contains a driver circuit. That works like a testbench in digital simulations. Next, digital system is added and connected to driver circuit. At last, DAC is added and connected to digital system. With DAC; bias circuit, VDD voltage supply etc. should also be added. Logic outputs of digital system are directly connected to analog inputs of the DAC in this schematic. Conversion is performed with configuration files of AMS.

### A.4. Configuration File

Configuration file is generated with File –> new -> cellview. Cellname is chosen as same as name of test schematic, view name and type should be chosen as config. Application should be opened with hierarchy editor.

In the newly window, Use template button is hit and AMS is selected. Then, OK is hit. Template becomes selected now. Then, view is changed to schematic, Library list should be SIM_LIB and CORELIB. Then, OK is hit.

In configuration window, views should be checked. It they are OK (verilog for digitals, schematic & spectre for analogs), it is closed. Configuration file is now ready to use.

### A.5. Transient Simulation (AMS and APS)

For a transient simulation, newly created “config” view is selected and both yes’es are selected in the opened window. After that, ADE L is opened with simulation –> ADE L. Outputs to be plotted, transient simulation parameters etc. are selected.

In Setup -> simulator/directory/host, “ams” is selected. Then in Setup -> high performance options, APS is selected.

### A.6. ConnectLib

Last thing to select is connection rules. As it is necessary to convert digital signals to analog, Setup –> Connect Rules is selected. After that, Rules name is changed to connectLib.ConnRules_18Vfull_fast. If more modifications like changing rise and fall times is desired, copy of this file can be used and changed. It is necessary to select the modified file in this screen for that case.

### A.7. Run

When all things are set as told, run is hit in ADE L screen and simulation is begun.

## APPENDIX C

### C.1. Clock Divider

Clock divider contains shift registers that divide input clock signal by 2, 4, 8 and 16 to create clock signals to be used by the filters. Input clock is connected to clock input of the flip-flops and clock is divided with respect to the length of the shift register. Verilog code of the clock divider circuit is in Appendix V.

### C.2. Filters

Interpolation filters with modulation capability is the main topic of this work. They are already explained in Section 4 in detail.

### C.3. Registers

In the complete design, many memory cells are used in order to write/read data to/from internal circuits of the chip. Every memory cell has a unique address.

#### C.3.1. Register map

Addresses of the memory cells are put together in a register map given in Table C.1.

#### C.3.2. Trim registers (0-255)

Trim registers are used to trim current cells of the DACs. They are programmed by SPI to correct mismatch errors due to process variations. Every address has a memory cell with 24-bit length.

#### C.3.3. Filter registers (301-302)

Filter registers are used to set the operation mode of the chip. Description of Filter Reg 0 was given in Section 5.1.3.

Filter Reg 1 controls the modes of the DAC output. When modulation mode bits are set to 00 or 01, DAC outputs are non-return to zero (NZR) form. When it is 10, DAC outputs has return-to-zero(RZ) behavior. If it is set as 11, Outputs of the DAC becomes plus-minus (PM) of the input signal. For the last two cases, DAC clock frequency becomes twice of the fastest clock used in Filters. Bits of the Filter Reg 1 are given in Table C.2.

**Table C.1 :** Register map.

| Address | Name | Functionality |
|---|---|---|
| 0<br><br>255 | 24-Bit trim word for current cells | 2x512 current cells requires 256 registers with 6 bits |
| 256 | Offset adjust | Reserved for Future Use |
| 257 | Gain Adjust | |
| 258 | $I_{out}$ Level Adjust | |
| 259 | Clock Current Adjust | |
| 260 | Clock Offset Adjust | |
| 261<br><br>276 | Bandgap offset & Drift Adjust | |
| 277<br><br>284 | LVDS Receiver current adjust | |
| 285<br><br>300 | LVDS Receiver offset adjust | |
| 301 | Filter Reg 0 | int. & filter modes |
| 302 | Filter Reg 1 | mod. modes |
| 303<br><br>373 | Reserved addresses | Reserved for Future Use |
| 374 | RAMDAC_in_r | regs to fill RAMDAC |
| 375 | RAMDAC_in_i | |
| 376 | RAMDAC_out_r | regs to read RAMDAC |
| 377 | RAMDAC_out_i | |
| 378 | dout_1_r | regs to read Filter Outputs |
| 379 | dout_1_i | |
| 380 | dout_2_r | |
| 381 | dout_2_i | |
| 382 | dout_3_r | |
| 383 | dout_3_i | |

**Table C.2 :** Filter Reg 1 register.

| Reserved | | | | | | | | | | | | | | Mod. Mode | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | x | x | x | x | x | x | x | x | x | x | x | x | x | 0 | 0 |

### C.3.4. RAMDAC registers (374-377)

RAMDAC registers are used to write data to the RAMDAC and check its content. To write data to RAMDAC, data is written to RAMDAC_in registers first. Then, a clock signal is applied to make RAMDAC sample the data. To check the data that was written to RAMDAC before, RAMDAC_out registers are read. All read and write operations are done with SPI.

### C.3.5. Filter output registers (378-383)

Filter output registers contains the data that is at the output of the associated filter. For example; dout_2_i contains the output of the 2nd filter's imaginary data. Reading is done by SPI. These registers are added to system for accessing the internal outputs of the filter circuits.

## C.4. SPI with Control Block

SPI with control block provides accessibility to internal registers of the chip. It is responsible for the communication between user and internal circuits. I/O signals of the block are given in Table C.3. Verilog code of the SPI with Control Block "SPI_CB" is in Appendix V.

**Table C.3 :** I/O signals.

| Name | Direction | Width | Function |
|---|---|---|---|
| por | Input | 1 | Reset input |
| sclk | Input | 1 | Clock signal |
| fsync | Input | 1 | synchronization signal |
| sdin | Input | 1 | Serial data input <- User |
| dataR | Input | 512 | Parallel data input <- Registers |
| sdout | output | 1 | Serial data output -> User |
| resetRegOut | output | 1 | Reset signal -> Registers |
| sdoutEn | output | 1 | Enable signal -> sdout pin |
| BlowFuse | output | 1 | Blow signal -> Fuses |
| write | output | 1 | Write signal -> Registers |
| dataW | output | 1 | Serial data output -> Registers |
| CE | output | 512 | Clock Enable -> Registers & Fuses |

Using SPI, commands can be given to system like reading data, writing data, blowing a fuse etc. In following sections, format of a typical SPI command and their applications are presented.

### C.4.1. Reading a command

Synchronization of the system is provided by fsync input. All operations are made when fsync is “0”. After a successful reset from por, system waits for a serial data which is 24-bits long. When fsync is “0”, with every rising edge of the clock, 24-bit word is sampled from sdin input in “MSB comes first and LSB comes last” fashion. After 24 successive clocks, no more operation is performed (even if more clock edges come) and system starts to wait for fsync to be made “1”. When it becomes “1” then, transmission is completed.

If fsync becomes “1” before 24 clock cycles, current operation is halted.

After a successful reading, system determines what to do. Current functionality of the system is explained in Table C.4 below.

**Table C.4 :** Instruction definition.

| D23 | D22 | D21 | D20 | D19 | D[18:8] | D[7:0] | Functionn |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | X | X | Noop (Use for shift out) |
| 0 | 0 | 0 | 0 | 1 | X | X | Reset registers |
| 0 | 0 | 0 | 1 | 0 | X | X | Enable sdout pin |
| 0 | 0 | 0 | 1 | 1 | X | X | Disable sdout pin |
| 0 | 0 | 1 | 0 | 0 | ADDR | DATA | Write DATA to ADDR |
| 0 | 0 | 1 | 0 | 1 | ADDR | DATA | Read data from ADDR |
| 0 | 0 | 1 | 1 | 0 | ADDR | X | Blow Fuse at ADDR |
| 0 | 0 | 1 | 1 | 1 | X | X | Reserved |
| 0 | 1 | X | X | X | X | X | Reserved |
| 1 | X | X | X | X | X | X | Reserved |

As seen from Table C.4, some commands have associated DATA word either sent by user or SPI_CB according to the command word type.

### C.4.2. Command without DATA

For instructions without DATA, operation is performed instantly with 16th successful clock edge. “Reset registers” and “enable sdout” are two examples of that type. After that type of instruction, system returns to initial condition (see reading a command).

### C.4.3. Command with DATA

For command words with DATA, which are either read or write operation, a different path is followed.

#### C.4.3.1. Write command

For write case, writing starts with 17th clock edge and finishes with 24th. With 24th clock edge, 8 bit DATA is written to ADDR. After that, no more operation is performed (even if more clock edges come) and system starts to wait for fsync to be made "1". When it becomes "1" then, transmission is completed.

If fsync becomes "1" before 24 clock cycles, current operation halted. Register which was written might be corrupted and needs to be written again. Signal diagram of a write command is given in Figure C.1.

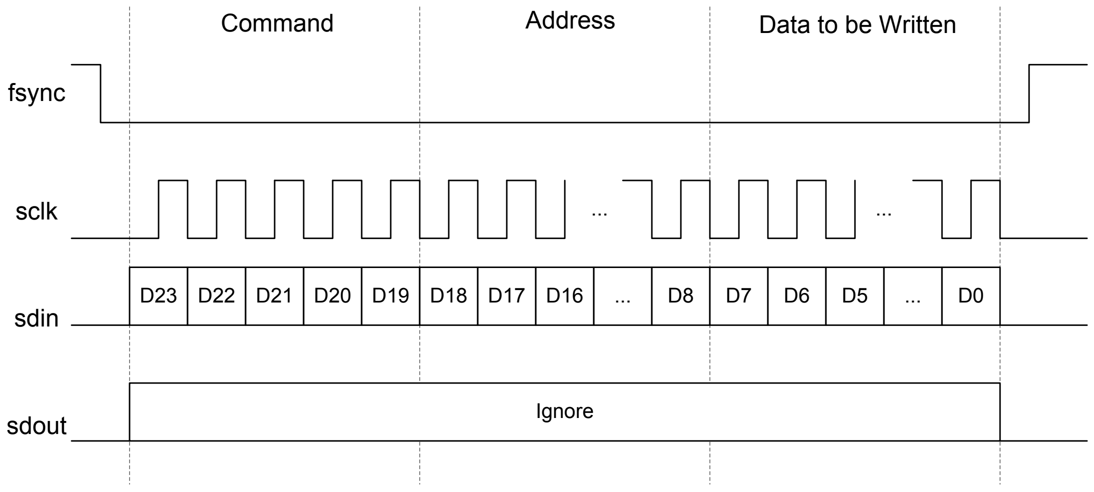


**Figure C.1 :** Signal diagram for write command.

#### C.4.3.2. Read command

For read case, reading starts with 17th clock edge and finishes with 24th. With 24th clock edge, 8 bit DATA is read from ADDR and given from sdout. After that, no more operation is performed (even if more clock edges come) and the system starts to wait for fsync to be made "1". When it becomes "1", the transmission is complete.

If fsync becomes "1" before 24 clock cycles, current operation halted. Register which was read might be corrupted and needs to be written again. An example signal diagram for read operation is given in Figure C.2.

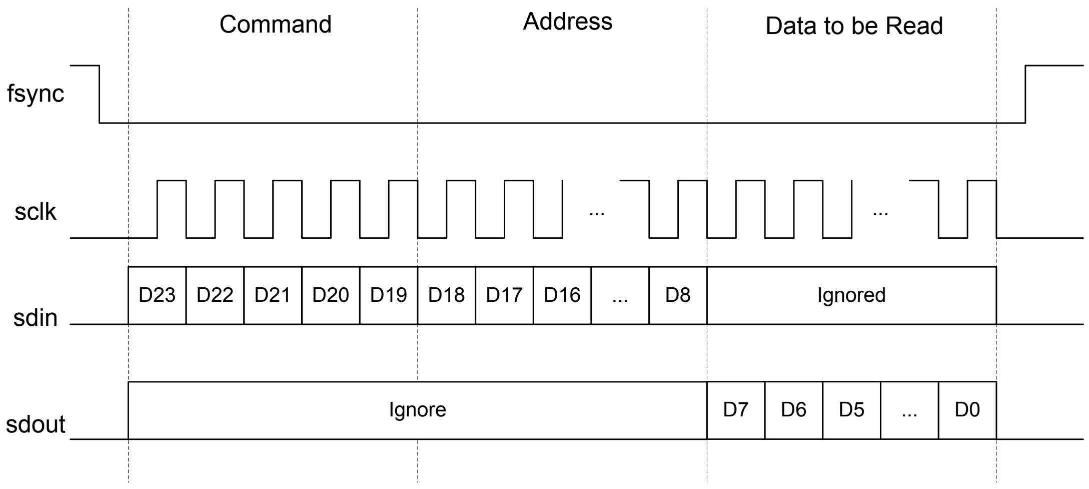


**Figure C.2 :** Signal diagram for read command.

### C.4.4. Blowing a fuse

Every Fuse is located between two register lines: first for programming and second for reading the fuse. If an address of the fuse is ADDR, address of the mask programming register is "ADDR-1" and address of the read register is "ADDR" which is same with Fuse.

Blowing a fuse requires a 4-step flow which includes both programming and testing. First of all, mask programming register "ADDR-1" is programmed with appropriate bit stream via a "Write" command. Then, "Blow Fuse" command is sent with fuse ADDR. After blowing, a single "Write" command is sent to ADDR. Writing to a fuse read register does not require a DATA because it samples associated fuse data. At the end, a "Read" command is sent to ADDR and content of the read register that sampled fuse data is read. An example is given in Table C.5.

**Table C.5 :** Blowing a fuse routine.

| Instruction Word (CMD,ADDR,DATA) | Functionality |
|---|---|
| {5'b00100, 11'h003, 8'b10000000} | Write 1 to MSB of address 3 MSB of fuse 4 will be blown |
| {5'b00110, 11'h004, 8'b00000000} | Blow fuse at address 4 wrt mask at address (4-1=3) |
| {5'b00100, 11'h004, 8'b00000000} | "Write" command to address 4 for sampling fused data |
| {5'b00101, 11'h004, 8'b00000000} | "Read" comment to address 4 for reading fused data |

## C.5. RAMDAC

RAMDAC is a memory loop which is used to apply high speed data to DACs internally. Data is written to the RAMDAC by enabling it and changing the mode to write. After writing is completed, RAMDAC is operated in enabled mode only. Within every clock pulse after that, RAMDAC data is applied to DACs and data is stored in the beginning of the RAMDAC. With that loop fashion, DACs can be tested at high speed and without the requirement of the external data applied from LVDS inputs of the chip. Two RAMDACs are used for the design as there are two DACs (one for real and one for imaginary output) in the chip. Verilog code of the RAMDAC is given in Appendix V.

## C.6. Binary to Thermometer Encoder with DAC Mode Selection

As mentioned earlier, the Filters and the RAMDAC operate with 16-bit binary data. However, the DAC accepts thermometric data as input. This conversion process is performed with “Bin2TherWmod” block. Block takes 16-bit binary encoded data and converts in to 11-bit binary LSB data and 31-bit thermometric MSB data. “Bin2TherWmod” also manipulates the data according to DAC mode. DAC modes and associated data patterns are explained in “Filter Registers” Section above.

There are two Binary to Thermometer Encoders with DAC mode selection in the design. Verilog Code of the “Bin2TherWmod” is given in Appendix V.

## APPENDIX E

## E.1. Tcl Script for Standard Architecture

```
set topcellname "Suzgec_Uclu_syn"
#set dbdir "DB"
#set WorkDir [format "/PROJ/LF150CGEN/DIGITAL/gurerozbek/%s/PAR" $topcellname]
set filename "Suzgec_Uclu_syn"
set LibDir "/work/kits/lf/1.8.0/LF150C/digital/liberty"

global filename
global WorkDir
global LibDir

proc LFMakeChip {} {
   ##--- Load configuration file
   #LFDBSetup

   ##--- Set User Grid
   LFUserGrid

   ##--- make global connections
   LFGlobalConnect
}
proc LFUserGrid {} {
   ##--- Set user grids
   setPreference ConstraintUserXGrid 0.1
   setPreference ConstraintUserXOffset 0.1
   setPreference ConstraintUserYGrid 0.1
   setPreference ConstraintUserYOffset 0.1
   setPreference SnapAllCorners 1
   setPreference BlockSnapRule 2

   snapFPlanIO -usergrid
}
proc LFGlobalConnect {} {
##--- Define global Power nets - make global connections
            clearGlobalNets
            globalNetConnect VDD -type pgpin -pin VDD -inst * -module {}
            globalNetConnect VSS -type pgpin -pin VSS  -inst * -module {}
            globalNetConnect VDD -type tiehi
            globalNetConnect VSS -type tielo
}
proc LFOpCond cond {

    switch $cond {
      "typ" {
                                               setOpCond -min typical -max typical
         }
      "minmax" {
                                               setOpCond -min best -max worst

      }
      "min" {
                                               setOpCond -min best -max best
      }
      "max" {
                                               setOpCond -min worst -max worst
      }
   }
}
proc LFSave postfix {
   global topcellname
   global dbdir
   set filename [format "%s.enc" $topcellname]
   saveDesign $filename
}
proc LFWrite postfix {
   global topcellname
   ##-- Save Design
   LFSave $postfix
```

```
  ##-- Write GDS2
  set filename [format "%s_%s.gds" $topcellname $postfix]
  set mapdir "/work/kits/lf/1.8.0/PDK_LF150i_V1_8_0/libraries/techfiles"

  streamOut $filename -mapFile $mapdir/encounter_layer.map -libName DesignLib -structureName $topcellname \
       -attachInstanceName -attachNetName -stripes 1 -units 1000 -mode ALL

  ##-- Verilog Netlist
  set filename [format "%s_%s.v" $topcellname $postfix]
  saveNetlist $filename

  ##-- Extract detail parasitics
  setXCapThresholds -totalCThreshold 5.0 -relativeCThreshold 0.01
  extractRC
  set filename [format "%s_%s.spef" $topcellname $postfix]

  ##-- run QX extraction
  #runqx
  #set filename [format "%s_%s_qx.spef" $topcellname $postfix]
  #rcOut -spef $filename
}

proc LFWriteSDF {} {
  global topcellname
  ##-- Parasitic Extraction
  #runQX

  ##-- typical SDF
  LFOpCond typ
  set filename_t [format "%s_typ.sdf" $topcellname]
  #delayCal -sdf $filename_t
  write_sdf -version 2.1 -prec 3 -edges check_edge -average_typ_delays  \
    -remashold -splitrecrem -splitsetuphold -force_calculation \
    $filename_t
  ##-- best case SDF
  LFOpCond min
  setAnalysisMode -hold
  set filename_b [format "%s_best.sdf" $topcellname]
  #delayCal -sdf $filename_b
  write_sdf -early -version 2.1 -prec 3 -edges check_edge -average_typ_delays  \
    -remashold -splitrecrem -splitsetuphold -force_calculation \
    $filename_b
  ##-- worst case SDF
  LFOpCond max
  setAnalysisMode -setup
  set filename_w [format "%s_worst.sdf" $topcellname]
  #delayCal -sdf $filename_w
  write_sdf -late -version 2.1 -prec 3 -edges check_edge -average_typ_delays  \
    -remashold -splitrecrem -splitsetuphold -force_calculation \
    $filename_w
  ##-- Combine all SDFs
  set filename [format "%s_all.sdf" $topcellname]
  sdfCombine -file $filename_b $filename_t $filename_w -output $filename
  print "### Combined SDF File for best/typ/worst written!!"
}
proc LFLoadCons {} {
  global filename
  ## load timing constraints
  unloadTimingCon
  set filepath [format "%s.sdc" $filename]
  if {[file exists $filepath]} {
      loadTimingCon $filepath
  } else {
    print "-E-# $filepath not found - no constraints loaded"
  }
}
proc LFFloorplan {type util iodist {ratio 1.0}} {
 ##-- Initialize floorplan
 switch $type {
   "core" {
          floorPlan -r $ratio $util $iodist $iodist $iodist $iodist
         }
   "peri" {
          floorPlan -r $ratio $util $iodist $iodist $iodist $iodist
          ##--- Load corner io file to add corner cells (if necessary)
#                                          loadIoFile corners.io

          ##-- Snap IO cells to user grid
          snapFPlanIO -usergrid
         }
  }
   fit
}
proc LFPowerRoute {{pownetsList {}}} {
   set offset 0.8
   # foreach power net in the specified list
   # route a ring
   foreach pownet $pownetsList {
      set name [lindex $pownet 0]
      set width [lindex $pownet 1]
      print "----$name $width $offset----"
      addRing \
          -width_left   $width -spacing_left   0.8 -offset_left   $offset -layer_left   METAL2 \
                      -width_top    $width -spacing_top    0.8 -offset_top    $offset -layer_top    METAL1 \
                      -width_right  $width -spacing_right  0.8 -offset_right  $offset -layer_right  METAL2 \
                      -width_bottom $width -spacing_bottom 0.8 -offset_bottom $offset -layer_bottom METAL1 \
                      -stacked_via_top_layer METAL2 \
                      -stacked_via_bottom_layer METAL1 \
                      -around core \
                      -jog_distance 0.7 \
                      -threshold 0.7 \
                      -nets $name
      set offset [ expr $offset + 0.8 + $width]
               global $width
   }
   addStripe \
                                    -spacing 0.8 -width $width -nets {VSS VDD} -layer METAL2 \
                                    -xleft_offset 120 -xright_offset 120 -number_of_sets 3 \
                                    -block_ring_top_layer_limit METAL3 \
                                    -block_ring_bottom_layer_limit METAL1 \
                                    -padcore_ring_top_layer_limit METAL3 \
                                    -padcore_ring_bottom_layer_limit METAL1 \
                                    -stacked_via_top_layer METAL_F \
                                    -stacked_via_bottom_layer METAL1 \
                                    -max_same_layer_jog_length 1.16 \
                                    -merge_stripes_value 0.61
   # do followpin routing
   sroute  -allowJogging true
}
proc LFPlace how {
   ##-- Placement
   switch $how {
      "ntd" {
            setPlaceMode -timingDriven false -reorderScan false -congEffort medium -doCongOpt false -modulePlan false
            placeDesign -noPrePlaceOpt
           }
      "td" {
            setPlaceMode -timingDriven true -reorderScan false -congEffort medium \
                    -doCongOpt false -modulePlan false
            placeDesign -noPrePlaceOpt
           }
      "opt"   {
            setPlaceMode -timingDriven true -reorderScan false -congEffort high \
                    -doCongOpt true -modulePlan false
            placeDesign -inPlaceOpt -noPrePlaceOpt
           }
   }
   LFSave placed
}
proc LFSave postfix {
   global topcellname
   global dbdir
   set filename [format "%s_%s.enc" $topcellname $postfix]
   saveDesign $filename
}
proc LFCts {} {
   global topcellname
   set filename [format "Clock.ctstch"]
   ##-- Specify Clock tree
   specifyClockTree -file $filename
  ##-- delete existing buffers
  #deleteClockTree -clk  <clockroot>

  ##-- Run CTS
  set filename1 [format "%s_cts.guide" $topcellname]
  set filename2 [format "%s_cts.ctsrpt" $topcellname]
  ckSynthesis -rguide $filename1 -report $filename2

  LFSave clkplaced
}
proc LFTa {state consList} {

   global topcellname
   foreach cons $consList {
     clearClockDomains
     setClockDomains -all
     LFLoadCons ## $cons
     set filename [format "%s_%s" $cons $state]
     switch $state {
       "prePlace" {timeDesign -prePlace -idealClock -pathReports -drvReports -slackReports -numPaths 50 \
                      -prefix $filename -outDir timingReports }
       "preCTS" {timeDesign -preCTS -idealClock -pathReports -drvReports -slackReports -numPaths 50 \
                      -prefix $filename -outDir timingReports }
       "postCTS" {timeDesign -postCTS -pathReports -drvReports -slackReports -numPaths 50 \
                      -prefix $filename -outDir timingReports
              clearClockDomains
              ## setClockDomains -all
              timeDesign -postCTS -hold -pathReports -slackReports -numPaths 50 \
                      -prefix $filename -outDir timingReports
              }
       "postRoute" {timeDesign -postRoute -pathReports -drvReports -slackReports -numPaths 50 \
                      -prefix $filename -outDir timingReports
              clearClockDomains
              setClockDomains -all
              timeDesign -postRoute -hold -pathReports -slackReports -numPaths 50 \
                      -prefix $filename -outDir timingReports
              }
       "signOff" {timeDesign -signOff -pathReports -drvReports -slackReports -numPaths 50 \
                      -prefix $filename -outDir timingReports
              clearClockDomains
              setClockDomains -all
              timeDesign -signOff -hold -pathReports -slackReports -numPaths 50 \
                      -prefix $filename -outDir timingReports
              }
     }
  }

}
proc LFOpt {state what cons} {
  unloadTimingCon
  LFLoadCons
  setOptMode -yieldEffort none
  setOptMode -effort high
  setOptMode -maxDensity 0.95
  setOptMode -drcMargin 0.0
  setOptMode -holdTargetSlack 200.0 -setupTargetSlack 200.0
  setOptMode -simplifyNetlist false
  clearClockDomains
  setClockDomains -all
  setOptMode -usefulSkew false
  optDesign -$state -$what
}
proc LFFillcore {} {
  ##-- Add Core Filler cells
            addFiller -cell FILLCELL_X1 FILLCELL_X2 FILLCELL_X4 FILLCELL_X8 FILLCELL_X16 FILLCELL_X32
FILLCELL_X64 -prefix FILLER
}
proc LFRoute {{router wroute}} {
   switch $router {
     "nano" {
           ##-- Run Routing
           ##-- Nano-Route
           getNanoRouteMode -quiet
           getNanoRouteMode -quiet envSuperThreading
```

```
            setNanoRouteMode -quiet -drouteFixAntenna true
            setNanoRouteMode -quiet -routeInsertAntennaDiode false
            setNanoRouteMode -quiet -timingEngine CTE
            setNanoRouteMode -quiet -routeWithTimingDriven false
            setNanoRouteMode -quiet -routeWithEco false
            setNanoRouteMode -quiet -routeWithSiDriven false
            setNanoRouteMode -quiet -routeTdrEffort 2
            setNanoRouteMode -quiet -routeSiEffort normal
            setNanoRouteMode -quiet -routeWithSiPostRouteFix false
            setNanoRouteMode -quiet -drouteAutoStop true
            setNanoRouteMode -quiet -routeSelectedNetOnly false
            setNanoRouteMode -quiet -drouteStartIteration default
            setNanoRouteMode -quiet -envNumberProcessor 1
            setNanoRouteMode -quiet -drouteEndIteration default
            globalDetailRoute
           }
   "wroute" {
            ##-- WROUTE
           wroute
           }
    }
}
```

## E.2. Tcl Script for Improved Architecture

```
set topcellname "Suzgec_Uclu_syn"
#set dbdir "DB"
#set WorkDir [format "/PROJ/TSMC150CGEN/DIGITAL/gurerozbek/%s/PAR" $topcellname]
set filename "Suzgec_Uclu_syn"
set filename_2 "Suzgec_Uclu"
set LibDir "/work/kits/tsmc/lib/180/stdcel_cl018g_2004q3v1/aci/sc/synopsys"

global filename
global WorkDir
global LibDir

proc TSMCHelp {} {
   global consList
   print "#### TSMCAuto Command Functionality"
   print "#### "
   print "---# TSMCAuto start end"
   print "---#    1 - TSMCConfig"
   print "---#    2 - TSMCMakeChip"
   print "---#    3 - TSMCFloorplan"
   print "---#    4 - TSMCLoadIOs"
   print "---#    5 - TSMCPowerRoute"
   print "---#    6 - TSMCPlace td"
   print "---#    7 - TSMCCts"
   print "---#    8 - TSMCPlace opt"
   print "---#    9 - Optimize postCTS"
   print "---#    10 - Optimize postRoute"
   print "---#    11 - editDelete -type Signal"
   print "---#    12 - TSMCRoute nano"
   print "---#    13 - Optimize postRoute"
}
proc TSMCAuto {start {end -1}} {
 if {$end == -1} { set end $start }
 for {set i $start} {$i<=$end} {incr i} {
   print "---# ---- Step $i -----"
   set step [format "s%d" $i]
   switch -exact $step {
     "s1"  { TSMCConfig }
     "s2"  { TSMCMakeChip }
     "s3"  { TSMCFloorplan }
     "s4"  { TSMCLoadIOs }
     "s5"  { TSMCPowerRoute }
     "s6"  { TSMCPlace td }
     "s7"  { TSMCCts }
     "s8"  { TSMCPlace opt }
     "s9"  { TSMCOpt postCTS }
     "s10"  { TSMCOpt postRoute }
     "s11"  { editDelete -type Signal }
     "s12"  { TSMCRoute nano }
     "s13"  { TSMCOpt postRoute }
     "s14"  {  }
    }
  }
}
proc TSMCConfig {} {
             global filename
             global WorkDir
             ##--- Load configuration file
             loadConfig [format "%s.conf" $filename] 0
             setUIVar rda_Input ui_gndnet VSS
             setUIVar rda_Input ui_pwrnet VDD
             commitConfig
             create_generated_clock -name "clk3_out" -source clk3 -divide_by 1 [get_ports clk3_out]
             reset_output_delay -clock [get_clocks clk3] [get_ports dout_*]
             set_output_delay -clock [get_clocks clk3_out] -add_delay 0.1 [get_ports dout_*]
             fit
}
proc TSMCMakeChip {} {
             ##--- Set User Grid
             TSMCUserGrid

             ##--- make global connections
             TSMCGlobalConnect
		##some commands that used in ams flow
		setClockMeshMode -propagationMode min_max
		setAnalysisMode -analysisType bcwc
}
proc TSMCUserGrid {} {
  ##--- Set user grids
  setPreference ConstraintUserXGrid 0.1
  setPreference ConstraintUserXOffset 0.1
  setPreference ConstraintUserYGrid 0.1
  setPreference ConstraintUserYOffset 0.1
  setPreference SnapAllCorners 1
  setPreference BlockSnapRule 2
  snapFPlanIO -usergrid
}
proc TSMCGlobalConnect {} {
##--- Define global Power nets - make global connections
		clearGlobalNets
		globalNetConnect VDD -type pgpin -pin VDD -inst * -module {}
		globalNetConnect VSS -type pgpin -pin VSS  -inst * -module {}
		globalNetConnect VDD -type tiehi
		globalNetConnect VSS -type tielo
}
proc TSMCFloorplan {} {
 ##-- Initialize floorplan
		getIoFlowFlag
		setIoFlowFlag 1
		setFPlanRowSpacingAndType 0.56 1
		#cell alaninin: eni - boyu. bu alanin etrafinin; solu - alti - sagi - ustu
		floorPlan -site tsm3site -s 800 2600 200 200 200 200
		uiSetTool select
		getIoFlowFlag
		fit
		#module floorplan'lerinin alana yerlestirilmesi
		#S_1
		selectObject Module S_1
		#S_2
		selectObject Module S_2
		#S_3
		selectObject Module S_3
		#save
		TSMCSave Fplaned
}
proc TSMCLoadIOs {} {
		global filename_2
		loadIoFile [format "%s.io" $filename_2]
}
proc TSMCPowerRoute {} {
  set offset 0.8
		setMultiCpuUsage -numThreads max
		#left right M6
		addRing -lt 1 -spacing_bottom 0.6 -width_left 99 -width_bottom 99 -width_top 99 -top 0 -spacing_top 0.6 -
layer_bottom METAL1 -stacked_via_top_layer METAL6 -width_right 99 -around core -jog_distance 0.66 -offset_bottom 0 -
bottom 0 -layer_top METAL1 -rb 1 -threshold 0.66 -offset_left 0 -spacing_right 0.6 -lb 1 -spacing_left 0.6 -offset_right 0 -rt 1 -
offset_top 0 -layer_right METAL6 -nets {VSS VDD } -follow io -stacked_via_bottom_layer METAL1 -layer_left METAL6
		#left right M5
		addRing -lt 1 -spacing_bottom 0.6 -width_left 99 -width_bottom 99 -width_top 99 -top 0 -spacing_top 0.6 -
layer_bottom METAL1 -stacked_via_top_layer METAL6 -width_right 99 -around core -jog_distance 0.66 -offset_bottom 0 -
bottom 0 -layer_top METAL1 -rb 1 -threshold 0.66 -offset_left 0 -spacing_right 0.6 -lb 1 -spacing_left 0.6 -offset_right 0 -rt 1 -
offset_top 0 -layer_right METAL5 -nets {VSS VDD } -follow io -stacked_via_bottom_layer METAL5 -layer_left METAL5
		# do followpin routing
		sroute  -allowJogging true
		#stripe yatay M5 (M4 below the ring)
		addStripe -block_ring_top_layer_limit METAL6 -max_same_layer_jog_length 0.88 -
padcore_ring_bottom_layer_limit METAL3 -set_to_set_distance 11.2 -stacked_via_top_layer METAL6 -
padcore_ring_top_layer_limit METAL6 -spacing 4.24 -ytop_offset 1.5 -switch_layer_over_obs 1 -ybottom_offset 4.64 -
merge_stripes_value 0.66 -layer METAL5 -block_ring_bottom_layer_limit METAL3 -width 1.36 -nets {VSS VDD } -
stacked_via_bottom_layer METAL1 -direction horizontal
		# M1 kalinlastirma
		addStripe -block_ring_top_layer_limit METAL1 -padcore_ring_bottom_layer_limit METAL1 -set_to_set_distance
11.2 -ybottom_offset 2.8 -area_blockage {200.676 200.451 200.676 2800.494 1000.6895 2800.494 1000.6895 200.4645
200.6765 200.4645 200.6765 200.451} -stacked_via_top_layer METAL2 -padcore_ring_top_layer_limit METAL1 -spacing 0.6
-allow_jog_padcore_ring 0 -direction horizontal -layer METAL1 -block_ring_bottom_layer_limit METAL1 -width 5 -nets
{VSS VDD } -stacked_via_bottom_layer METAL1 -allow_jog_block_ring 0
		#save
```

```
                TSMCSave powered
}
proc TSMCPlace how {
   ##-- Placement
   switch $how {
      "ntd" {
               setPlaceMode -timingDriven false -reorderScan false -congEffort medium \
                                                   -doCongOpt false -modulePlan false -powerDriven true
                         setMultiCpuUsage -numThreads max
               placeDesign -noPrePlaceOpt
              }
      "td" {
               setPlaceMode -timingDriven true -reorderScan false -congEffort medium \
                         -doCongOpt false -modulePlan true -powerDriven true
                         setMultiCpuUsage -numThreads max
               placeDesign -noPrePlaceOpt
                         #save
                         TSMCSave placed
              }
      "opt"   {
               setPlaceMode -timingDriven true -reorderScan false -congEffort high \
                         -doCongOpt true -modulePlan true -wireLenOptEffort high \
                                                   -powerDriven true
                         setMultiCpuUsage -numThreads max
               placeDesign -inPlaceOpt -noPrePlaceOpt
                         #save
                         TSMCSave optPlaced
              }
   }
   TSMCSave placed
}
proc TSMCCts {} {
   global topcellname
   set filename [format "Clock.ctstch"]
   ##-- Specify Clock tree
   specifyClockTree -file $filename
   ##-- delete existing buffers
   #deleteClockTree -clk  <clockroot>
   ##-- Run CTS
   set filename1 [format "%s_cts.guide" $topcellname]
   set filename2 [format "%s_cts.ctsrpt" $topcellname]
   setCTSMode -powerAware true -optAddBuffer true -optLatency true -traceIoPinAsLeaf true
   setMultiCpuUsage -numThreads max
   ckSynthesis -rguide $filename1 -report $filename2
   #save
   TSMCSave clkplaced
}
proc TSMCRoute {router {effort 5}} {
    switch $router {
      "nano" {
                         ##-- Run Routing
                         ##-- Nano-Route
                         getNanoRouteMode -quiet
                         getNanoRouteMode -quiet envSuperThreading
                         setNanoRouteMode -quiet -drouteFixAntenna true
                         setNanoRouteMode -quiet -routeInsertAntennaDiode false
                         setNanoRouteMode -quiet -timingEngine CTE
                         setNanoRouteMode -quiet -routeWithTimingDriven true
                         setNanoRouteMode -quiet -routeWithEco false
                         setNanoRouteMode -quiet -routeWithSiDriven false
                         setNanoRouteMode -quiet -routeTdrEffort $effort
                         setNanoRouteMode -quiet -routeSiEffort normal
                         setNanoRouteMode -quiet -routeWithSiPostRouteFix false
                         setNanoRouteMode -quiet -drouteAutoStop false
                         setNanoRouteMode -quiet -routeSelectedNetOnly false
                         setNanoRouteMode -quiet -drouteStartIteration default
                         setNanoRouteMode -quiet -envNumberProcessor 24
                         setNanoRouteMode -quiet -drouteEndIteration default
                         globalDetailRoute
                         #save
                         TSMCSave routed
             }
    "wroute" {
              ##-- WROUTE
              wroute
```

```
            }
    "repair" {
             ##-- REPAIR
             setNanoRouteMode -quiet -drouteStartIteration 1
             setNanoRouteMode -quiet -envNumberProcessor 24
             setNanoRouteMode -quiet -drouteEndIteration default
             globalDetailRoute
            }
    }
}
proc TSMCOpt {state} {
   setOptMode -yieldEffort none
   setOptMode -effort high
   setOptMode -maxDensity 0.95
   #setOptMode -drcMargin 0.0
   setOptMode -holdTargetSlack 0.0 -setupTargetSlack 0.0
   setOptMode -simplifyNetlist false
   setOptMode -usefulSkew false
   setOptMode -fixCap true
   setOptMode -fixTran true
   setOptMode -fixFanoutLoad false
   optDesign -$state
   optDesign -$state -hold
}
proc TSMCSave postfix {
   global filename_2
   set filename [format "%s_%s.enc" $filename_2 $postfix]
   saveDesign $filename
}
proc TSMCOpCond cond {

    switch $cond {
     "typ" {
                                                 setOpCond -min typical -max typical
         }
     "minmax" {
                                                 setOpCond -min best -max worst

     }
     "min" {
                                                 setOpCond -min best -max best
     }
     "max" {
                                                 setOpCond -min worst -max worst
     }
   }
}
proc TSMCWrite postfix {
   global topcellname
   ##-- Save Design
   TSMCSave $postfix
   ##-- Write GDS2
   set filename [format "%s_%s.gds" $topcellname $postfix]
   #set mapdir "/work/kits/lf/1.8.0/PDK_TSMC150i_V1_8_0/libraries/techfiles"
   #streamOut $filename -mapFile $mapdir/encounter_layer.map -libName DesignLib -structureName $topcellname \
   #      -attachInstanceName -attachNetName -stripes 1 -units 1000 -mode ALL
   ##-- Verilog Netlist
   set filename [format "%s_%s.v" $topcellname $postfix]
   saveNetlist $filename
   ##-- Extract detail parasitics
   setXCapThresholds -totalCThreshold 5.0 -relativeCThreshold 0.01
   extractRC
   set filename [format "%s_%s.spef" $topcellname $postfix]
   ##-- run QX extraction
   #runqx
   #set filename [format "%s_%s_qx.spef" $topcellname $postfix]
   #rcOut -spef $filename
}
proc TSMCWriteSDF {} {
   global topcellname
   ##-- Parasitic Extraction
   #runQX
   ##-- typical SDF
   TSMCOpCond typ
   set filename_t [format "%s_typ.sdf" $topcellname]
```

```
    write_sdf -version 2.1 -prec 3 -edges check_edge -average_typ_delays  \
      -remashold -splitrecrem -splitsetuphold -force_calculation \
      $filename_t
    ##-- best case SDF
    TSMCOpCond min
    setAnalysisMode -hold
    set filename_b [format "%s_best.sdf" $topcellname]
    write_sdf -early -version 2.1 -prec 3 -edges check_edge -average_typ_delays  \
      -remashold -splitrecrem -splitsetuphold -force_calculation \
      $filename_b
    ##-- worst case SDF
    TSMCOpCond max
    setAnalysisMode -setup
    set filename_w [format "%s_worst.sdf" $topcellname]
    write_sdf -late -version 2.1 -prec 3 -edges check_edge -average_typ_delays  \
      -remashold -splitrecrem -splitsetuphold -force_calculation \
      $filename_w
    ##-- Combine all SDFs
    set filename [format "%s_all.sdf" $topcellname]
    sdfCombine -file $filename_b $filename_t $filename_w -output $filename
    print "### Combined SDF File for best/typ/worst written!!"
}
proc TSMCTa {state consList} {
    global topcellname
    foreach cons $consList {
      clearClockDomains
      setClockDomains -all
      TSMCLoadCons ## $cons
      set filename [format "%s_%s" $cons $state]
      switch $state {
        "prePlace" {timeDesign -prePlace -idealClock -pathReports -drvReports -slackReports -numPaths 50 \
                    -prefix $filename -outDir timingReports }
        "preCTS" {timeDesign -preCTS -idealClock -pathReports -drvReports -slackReports -numPaths 50 \
                    -prefix $filename -outDir timingReports }
        "postCTS" {timeDesign -postCTS -pathReports -drvReports -slackReports -numPaths 50 \
                    -prefix $filename -outDir timingReports
               clearClockDomains
               ## setClockDomains -all
               timeDesign -postCTS -hold -pathReports -slackReports -numPaths 50 \
                    -prefix $filename -outDir timingReports
               }
        "postRoute" {timeDesign -postRoute -pathReports -drvReports -slackReports -numPaths 50 \
                    -prefix $filename -outDir timingReports
               clearClockDomains
               setClockDomains -all
               timeDesign -postRoute -hold -pathReports -slackReports -numPaths 50 \
                    -prefix $filename -outDir timingReports
               }
        "signOff" {timeDesign -signOff -pathReports -drvReports -slackReports -numPaths 50 \
                    -prefix $filename -outDir timingReports
               clearClockDomains
               setClockDomains -all
               timeDesign -signOff -hold -pathReports -slackReports -numPaths 50 \
                    -prefix $filename -outDir timingReports
               }
      }
    }
}
proc TSMCFillcore {} {
    ##-- Add Core Filler cells
              addFiller -cell FILL1 FILL2 FILL4 FILL8 FILL16 FILL32 FILL64 -prefix FILL
}
proc TSMCViaFill {} {
              #power analizlerinden once bunu yapman lazÄ±m
              editPowerVia -bottom_layer METAL5 -add_vias 1 -orthogonal_only 0 -top_layer METAL6
              editPowerVia -bottom_layer METAL4 -add_vias 1 -orthogonal_only 0 -top_layer METAL5
              editPowerVia -bottom_layer METAL3 -add_vias 1 -orthogonal_only 0 -top_layer METAL4
              editPowerVia -bottom_layer METAL2 -add_vias 1 -orthogonal_only 0 -top_layer METAL3
              editPowerVia -bottom_layer METAL1 -add_vias 1 -orthogonal_only 0 -top_layer METAL2
}
proc TSMCAddEndCaps {} {
    ##-- add CAP cells
    addEndCap -preCap FILL1 -postCap FILL1 -prefix ENDCAP
}
```

## E.3. SDF File for Improved Architecture

```
(DELAYFILE
 (SDFVERSION "2.1")
 (DESIGN "Suzgec_Uclu")
 (DATE "Mon Nov 19 13:50:15 2012")
 (VENDOR "Cadence Design Systems, Inc.")
 (PROGRAM "Encounter")
 (VERSION "v09.12-s159_1 ((32bit) 07/15/2010 13:17 (Linux 2.6))")
 (DIVIDER /)
 (VOLTAGE 1.800000:1.800000:1.800000)
 (PROCESS "1.000000:1.000000:1.000000")
 (TEMPERATURE 25.000000:25.000000:25.000000)
 (TIMESCALE 1.0 ns)

 (CELL
  (CELLTYPE  "Suzgec_Uclu")
  (INSTANCE)
    (DELAY
          (ABSOLUTE
          (INTERCONNECT FE_PHC2835_din_r_15_/Y I_M/retime_s2_80_reg/D  (0.003:0.003:0.003)
(0.003:0.003:0.003))
          (INTERCONNECT FE_PHC2835_din_r_15_/Y O_M/dout_r_reg\[15\]/D  (0.007:0.007:0.007) (0.007:0.007:0.007))
          (INTERCONNECT FE_PHC2832_din_r_14_/Y I_M/retime_s2_84_reg/D  (0.001:0.001:0.001)
(0.001:0.001:0.001))
          (INTERCONNECT FE_PHC2832_din_r_14_/Y O_M/dout_r_reg\[14\]/D  (0.006:0.006:0.006) (0.006:0.006:0.006))
          (INTERCONNECT FE_PHC2831_din_r_8_/Y I_M/retime_s2_81_reg/D  (0.002:0.002:0.002) (0.002:0.002:0.002))
          (INTERCONNECT FE_PHC2831_din_r_8_/Y O_M/dout_r_reg\[8\]/D  (0.010:0.010:0.010) (0.010:0.010:0.010))
          (INTERCONNECT FE_PHC2830_din_i_11_/Y I_M/retime_s2_99_reg/D  (0.001:0.001:0.001) (0.001:0.001:0.001))
          (INTERCONNECT FE_PHC2830_din_i_11_/Y O_M/dout_i_reg\[11\]/D  (0.003:0.003:0.003) (0.003:0.003:0.003))
          (INTERCONNECT FE_PHC2829_din_r_12_/Y I_M/retime_s2_53_reg/D  (0.007:0.007:0.007)
(0.007:0.007:0.007))
          (INTERCONNECT FE_PHC2829_din_r_12_/Y O_M/dout_r_reg\[12\]/D  (0.007:0.007:0.007) (0.007:0.007:0.007))
          (INTERCONNECT FE_PHC2828_din_r_9_/Y I_M/retime_s2_82_reg/D  (0.001:0.001:0.001) (0.001:0.001:0.001))
          (INTERCONNECT FE_PHC2828_din_r_9_/Y O_M/dout_r_reg\[9\]/D  (0.012:0.012:0.012) (0.012:0.012:0.012))
          (INTERCONNECT FE_PHC2827_din_i_10_/Y I_M/g3671/A  (0.003:0.003:0.003) (0.003:0.003:0.003))
          …
(INTERCONNECT din_i[2] S_1/FE_PHC2759_din_i_2_/A  (0.308:0.308:0.308) (0.308:0.308:0.308))
          (INTERCONNECT din_i[1] FE_PHC2747_din_i_1_/A  (0.035:0.035:0.035) (0.035:0.035:0.035))
          (INTERCONNECT din_i[0] FE_PHC2744_din_i_0_/A  (0.042:0.042:0.042) (0.042:0.042:0.042))
          )
    )
 )

 (CELL
  (CELLTYPE  "CLKBUFX2")
  (INSTANCE  FE_PHC2835_din_r_15_)
    (DELAY
          (ABSOLUTE
          (IOPATH A Y  (0.308:0.308:0.308) (0.364:0.364:0.364))
          )
    )
 )

 (CELL
  (CELLTYPE  "BUFX1")
  (INSTANCE  FE_PHC2832_din_r_14_)
    (DELAY
          (ABSOLUTE
          (IOPATH A Y  (0.434:0.434:0.434) (0.324:0.324:0.324))
          )
    )
 )

…


 (CELL
  (CELLTYPE  "DFFRHQXL")
  (INSTANCE  retime_s1_34_reg)
    (DELAY
          (ABSOLUTE
          (IOPATH RN Q  () (0.406:0.406:0.406))
```

```
            (IOPATH CK Q  (0.753:0.753:0.753) (0.482:0.482:0.482))
            )
    )
    (TIMINGCHECK
            (WIDTH (negedge RN) (0.255:0.255:0.255))
            (WIDTH (posedge CK) (0.124:0.124:0.124))
            (WIDTH (negedge CK) (0.177:0.177:0.177))
            (HOLD (posedge D) (posedge CK) (-0.033:-0.033:-0.034))
            (HOLD (negedge D) (posedge CK) (0.000:0.000:-0.002))
            (SETUP (posedge D) (posedge CK) (0.079:0.079:0.080))
            (SETUP (negedge D) (posedge CK) (0.175:0.175:0.177))
            (HOLD (posedge RN) (posedge CK) (-0.034:-0.034:-0.034))
            (RECOVERY (posedge RN) (posedge CK) (0.101:0.101:0.101))
    )
  )

  (CELL
   (CELLTYPE  "DFFRHQXL")
   (INSTANCE  retime_s1_10_reg)
    (DELAY
            (ABSOLUTE
            (IOPATH RN Q  () (0.394:0.394:0.394))
            (IOPATH CK Q  (0.720:0.720:0.720) (0.472:0.472:0.472))
            )
    )
    (TIMINGCHECK
            (WIDTH (negedge RN) (0.255:0.255:0.255))
            (WIDTH (posedge CK) (0.124:0.124:0.124))
            (WIDTH (negedge CK) (0.177:0.177:0.177))
            (HOLD (posedge D) (posedge CK) (-0.034:-0.034:-0.034))
            (HOLD (negedge D) (posedge CK) (0.001:0.001:-0.003))
            (SETUP (posedge D) (posedge CK) (0.079:0.079:0.079))
            (SETUP (negedge D) (posedge CK) (0.175:0.175:0.180))
            (HOLD (posedge RN) (posedge CK) (-0.033:-0.033:-0.033))
            (RECOVERY (posedge RN) (posedge CK) (0.099:0.099:0.099))
    )
  )

  (CELL
   (CELLTYPE  "DFFRHQXL")
   (INSTANCE  retime_s1_3_reg)
    (DELAY
            (ABSOLUTE
            (IOPATH RN Q  () (0.404:0.404:0.404))
            (IOPATH CK Q  (0.748:0.748:0.748) (0.479:0.479:0.479))
            )
    )
    (TIMINGCHECK
            (WIDTH (negedge RN) (0.255:0.255:0.255))
            (WIDTH (posedge CK) (0.124:0.124:0.124))
            (WIDTH (negedge CK) (0.177:0.177:0.177))
            (HOLD (posedge D) (posedge CK) (-0.034:-0.034:-0.034))
            (HOLD (negedge D) (posedge CK) (0.001:0.001:-0.001))
            (SETUP (posedge D) (posedge CK) (0.079:0.079:0.079))
            (SETUP (negedge D) (posedge CK) (0.175:0.175:0.178))
            (HOLD (posedge RN) (posedge CK) (-0.033:-0.033:-0.033))
            (RECOVERY (posedge RN) (posedge CK) (0.099:0.099:0.099))
    )
  )

  (CELL
   (CELLTYPE  "CLKINVX8")
   (INSTANCE  g73)
    (DELAY
            (ABSOLUTE
            (IOPATH A Y  (0.149:0.149:0.149) (0.126:0.126:0.126))
            )
    )
  )
)
```

# APPENDIX I

## I.1. Synthesis

### I.1.1. Getting started

First thing to do is creating a new folder to work and copying all Verilog model files and the script file in it. Then “rc –gui” command is entered to open RTL Compiler in gui mode. After that, the only thing to do is writing “source script.tcl” in command window of RTL Compiler. The script will be run and outputs will collected then.

### I.1.2. Specs & script

Tcl script consists of some parts including reading input files, setting synthesis specifications and writing output files.

Synthesis operation requires model files to be read. First files to be read are library files having gate models. After that, reading of Verilog model files takes place. Verilog models are read with “read_hdl” command. Last thing to do is elaborating design with a name.

Second part starts with defining clock signals. That operation is done with “define_clock” command and parameters like period, name of clock and for some cases, clock division ratio are given. After that, clock skews are given with “set_attribute” command and external delays with “external_delay” command. Moreover, retiming specifications can be set with “set_attribute” command at that point. Then retime can be performed with “retime –min_delay” command. At last, synthesis operation is performed with “synthesize –to_mapped” command.

In the end, outputs of synthesis are written to files to be an input to later tools. At this point, “write_encounter” command creates all files that are required for both simulation and PAR. By doing that, .v, .mode, .conf and .sdc files are generated. Also, timing and area reports can be generated to check the performance of the synthesis.

Example tcl-script, timing and area report files are in Appendix R (RTL Compiler).

## I.2. PAR

### I.2.1. Importing data from RTL compiler

First step of PAR started with creating a folder and copy .v, .mode, .conf and .sdc files and generated by RTL Compiler and a script file in it. Then, a modification is needed to be done in .sdc file. Some lines started with “set_input_delay” command define a delay between clock signals which are not desired. These lines (four lines total) should be removed in order to work properly. After that, if retiming was performed in synthesis, “uniquifyNetlist” command should be run at unix terminal. That modifies synthesized complex Verilog model and simplifies it. By using “velocity” command, Encounter is run.

When Encounter gui was opened, “source script.tcl” is written to command line and run. By doing that, predefined commands in script are loaded to Encounter and can be called with single name commands. Script also made Encounter read input files except .conf file. To read it, Design -> Import Design -> Load -> .conf -> Open is selected in gui and at advanced tab, Power -> VDD VSS -> OK should be selected.

### I.2.2. Steps of PAR

First, the command that is used is “LFMakeChip”, which prepares the tool for PAR flow. Then, “LFFloorplan core 0.7 50 1” is entered. That command states that; design is a subchip (not entire), %70 of the area is left for cells, left 50 μm for power routings and made floorplan as square shaped. After that, power routings are done with “LFPowerRoute {{VDD 24} {VSS 24}}” command. It makes 2 power signals VDD and VSS with 24 μm width around the floorplan. Designing floorplan finishes here, so saving with Design -> Save -> FloorPlan is recommended.

Next step starts with pin placement by selecting Edit -> Pin Editor. In the pin editor, appropriate pins’ shapes, size, place and metal types are selected and saved with Design -> Save -> IO file. After that, placement of the cells happens with “LFPlace td” command. That makes cell placement with minimized delays possible. After that, “LFCts” command is entered to synthesize a clocks tree. With a clock tree implemented, a placement optimization with “LFPlace opt” command is recommended at this time. Another optimization is done by selecting Timing -> Optimize -> Setup time from gui. That improves setup time for registers. After optimizations were performed, “LFFillcore” command is entered to fill empty areas with capacitances that help supply voltages to not change so much. At this time, Placement is done and it is a good idea to save it with Design -> Save -> Place.

Last step of the design is routing with command “LFRoute nano”. That starts nano router which has a better performance that conventional router of Encounter. When seeing “zero violations” at the command line, design is finished. It can be saved by Design -> Save -> Route.

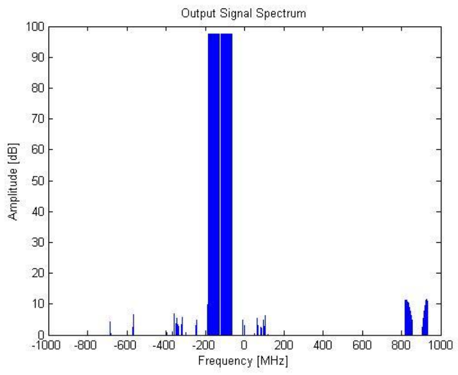
### I.2.3. Outputs: GDS-II & SDF

GDS-II file is generated with “LFWrite final” command. It also generates a Verilog model for simulation. SDF file is generated with “LFWriteSDF” command.

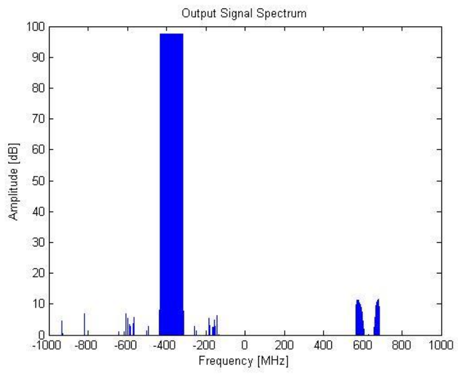
## APPENDIX P

### P.1. Negative Frequency Plots of Section 3.1.3.9.2

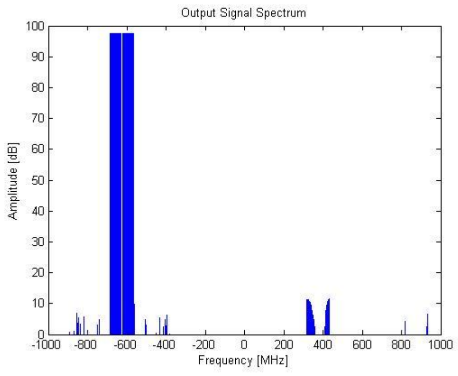
Spectrum plots associated with the negative modulation frequencies are given in this section. See Figure P.1.



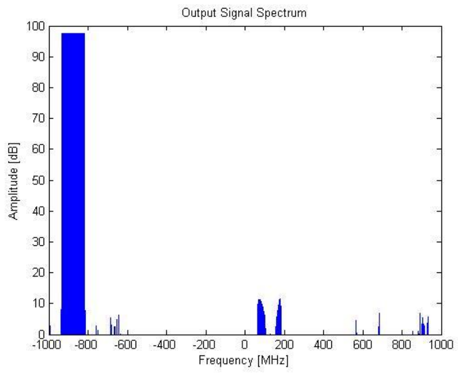
(a)



(b)



(c)

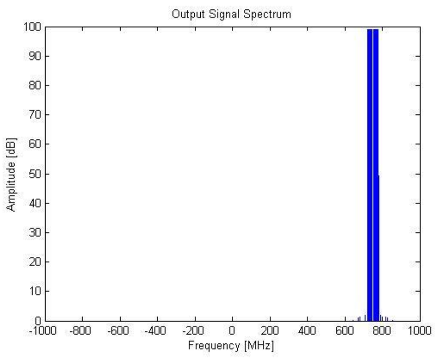


(d)

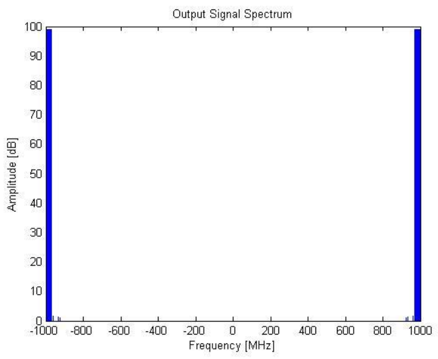
**Figure P.1 :** Output Spectrums of Mid-Modes. (a): -$F_{DAC}$/16, (b): -3$F_{DAC}$/16, (c): -5$F_{DAC}$/16, (d): -7$F_{DAC}$/16

### P.2. Other Plots of Section 4.3.2

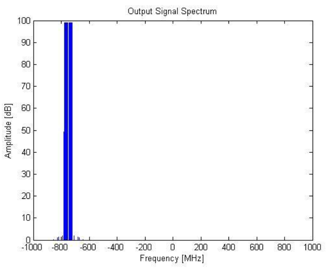
Spectrum plots of the filter modes. Modulations with 2x interpolation are in Figure P.2:



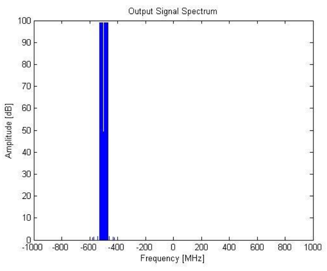
(a)



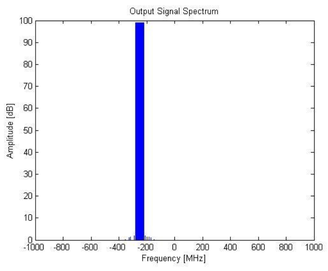
(b)



(c)



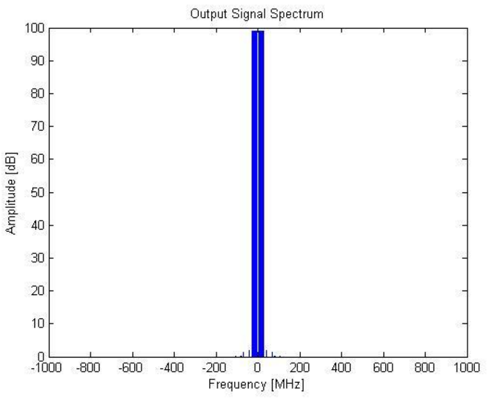
(d)



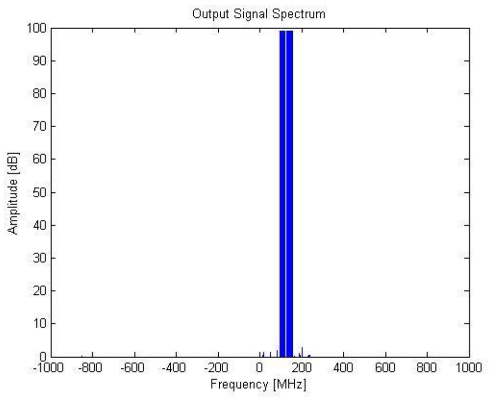
(e)

**Figure P.2 :** Output Spectrums of the Modes that Modulates with 2x Interpolation. (a): $3F_{DAC}/8$, (b): $4F_{DAC}/8$, (c): $-3F_{DAC}/8$, (d): $-2F_{DAC}/8$, (e): $-F_{DAC}/8$

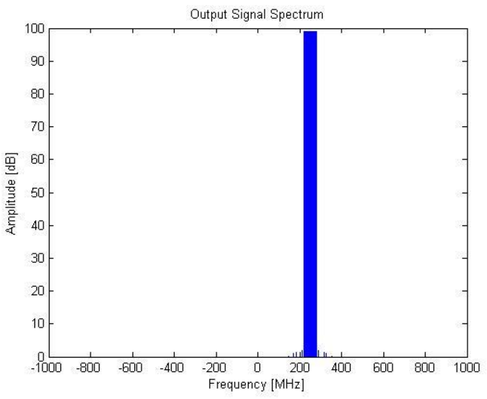
Spectrum plots of the filter modes. Plots for modulations with 4x and 8x interpolation are the same, so only one of the plots is given. See Figure P.3.



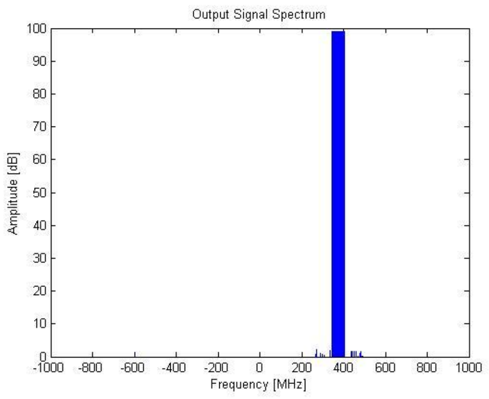
(a)



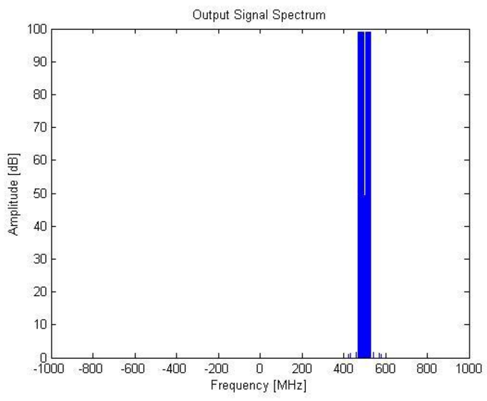
(b)



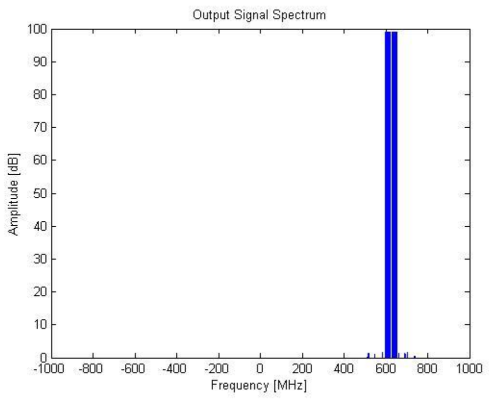
(c)



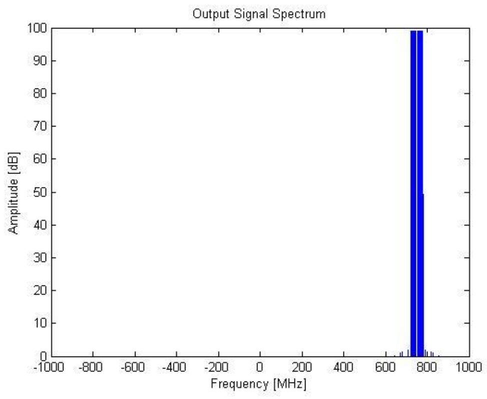
(d)


Output Signal Spectrum
Amplitude [dB]
Frequency [MHz]

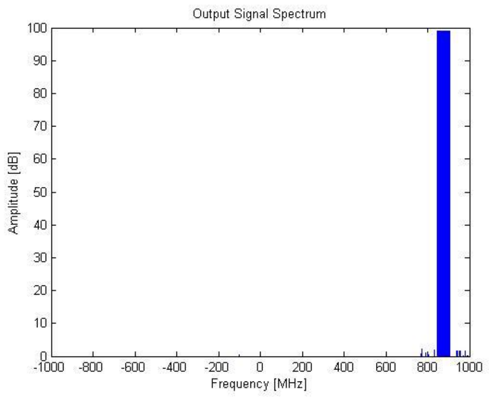
(e)


Output Signal Spectrum
Amplitude [dB]
Frequency [MHz]

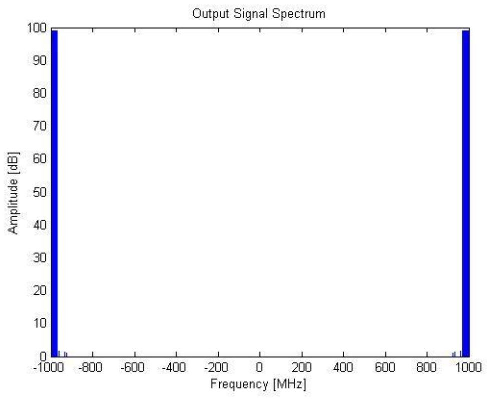
(f)


Output Signal Spectrum
Amplitude [dB]
Frequency [MHz]

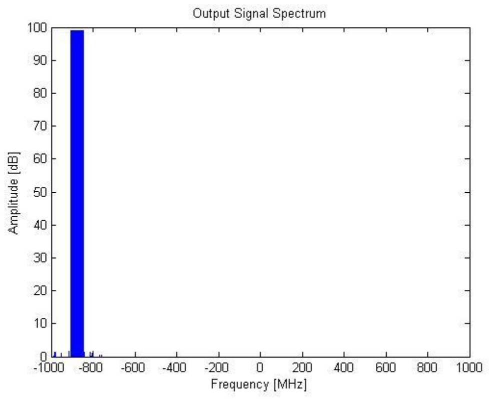
(g)


Output Signal Spectrum
Amplitude [dB]
Frequency [MHz]

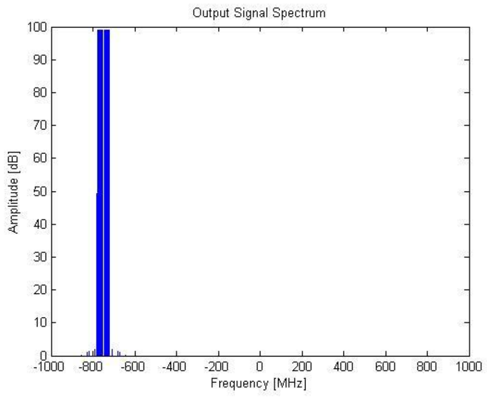
(h)


Output Signal Spectrum
Amplitude [dB]
Frequency [MHz]

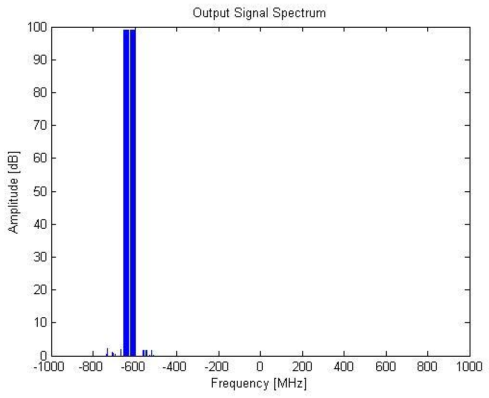
(i)


Output Signal Spectrum
Amplitude [dB]
Frequency [MHz]

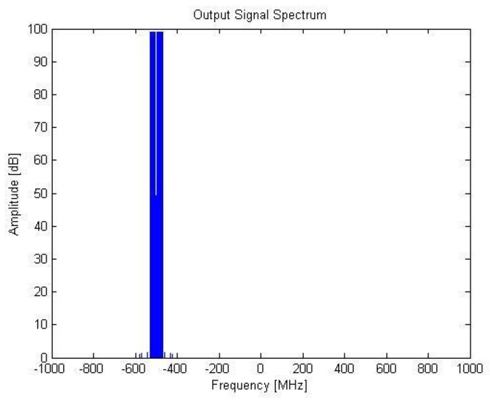
(j)


Output Signal Spectrum
Amplitude [dB]
Frequency [MHz]

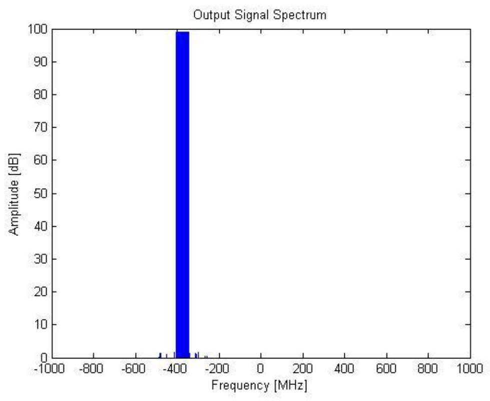
(k)


Output Signal Spectrum
Amplitude [dB]
Frequency [MHz]

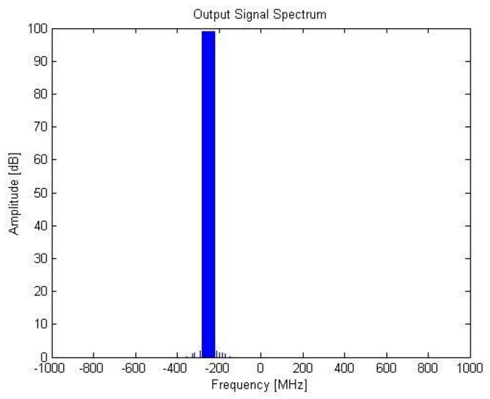
(l)



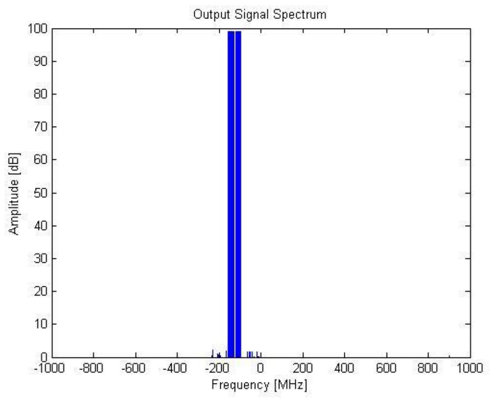
(m)



(n)



(o)



(p)

**Figure P.3 :** Output spectrums of the modes that modulates with 4x and 8x interpolation. (a): baseband, (b): $F_{DAC}$/16, (c): $2F_{DAC}$/16, (d): $3F_{DAC}$/16, (e): $4F_{DAC}$/16, (f): $5F_{DAC}$/16, (g): $6F_{DAC}$/16, (h): $7F_{DAC}$/16, (i): $8F_{DAC}$/16, (j): $-7F_{DAC}$/16, (k): $-6F_{DAC}$/16, (l): $-5F_{DAC}$/16, (m): $-4F_{DAC}$/16, (n): $-3F_{DAC}$/16, (o): $-2F_{DAC}$/16, (p): $-F_{DAC}$/16.

## APPENDIX R

### R.1. Tcl Script for Standard Architecture

```
# Cadence Encounter(r) RTL Compiler
#    Special thanks to: Gurer Ozbek
#kutuphane adresi:
set_attribute lib_search_path /work/kits/lf/1.8.0/PDK_LF150i_V1_8_0/digital/liberty
#kutuphanenin adi:
set_attribute library {LF150DI_HS_F_V1_5_typical_conditional.lib}
#LEF kutuphanesi adresi
set_attribute lib_search_path /work/kits/lf/1.8.0/PDK_LF150i_V1_8_0/digital/lef
#LEF kutuphanesi adi
set_attribute lef_library {LF150DI_HS_F_V1_5.lef}

#okunacak dosyalar:
read_hdl -v2001 Suzgec_1.v
read_hdl -v2001 Suzgec_2.v
read_hdl -v2001 Suzgec_3.v
read_hdl -v2001 Suzgec_Uclu.v
elaborate Suzgec_Uclu

report datapath > datapath.txt

#saat isareti tanimlamalari
define_clock -period 7200 -name clk0 /designs/Suzgec_Uclu/ports_in/clk0
define_clock -period 7200 -name clk1 -divide_period 2 /designs/Suzgec_Uclu/ports_in/clk1
define_clock -period 7200 -name clk2 -divide_period 4 /designs/Suzgec_Uclu/ports_in/clk2
define_clock -period 7200 -name clk3 -divide_period 8 /designs/Suzgec_Uclu/ports_in/clk3

set_attribute slew {40 40 80 80} [find -clock clk0]
set_attribute slew {40 40 80 80} [find -clock clk1]
set_attribute slew {40 40 80 80} [find -clock clk2]
set_attribute slew {40 40 80 80} [find -clock clk3]

external_delay -input 500 -clock [find -clock clk0] -edge_rise /designs/Suzgec_Uclu/ports_in/*

external_delay -output 100 -clock [find -clock clk3] -edge_rise /designs/Suzgec_Uclu/ports_out/*

#synthesize -to_mapped -effort high

#retime islemleri
set_attribute dont_retime false [all::all_seqs -clock clk1]
set_attribute dont_retime true [all::all_seqs -clock clk2]
retime -min_delay

synthesize -to_mapped -effort high

write_encounter design -basename Suzgec_Uclu_syn
report_timing > timing_report.txt
report area > area_report.txt
```

## R.2. Tcl Script for Improved Architecture

```
# Cadence Encounter(r) RTL Compiler
#     Special thanks to: Gurer Ozbek
#kutuphane adresi:
set_attribute lib_search_path /work/kits/tsmc/lib/180/stdcel_cl018g_2004q3v1/aci/sc/synopsys
#kutuphanenin adi:
set_attribute library {typical.lib}
#LEF kutuphanesi adresi
set_attribute lib_search_path /work/kits/tsmc/lib/180/stdcel_cl018g_2004q3v1/aci/sc/lef
#LEF kutuphanesi adi
set_attribute lef_library {tsmc18_6lm.lef}
#okunacak dosyalar:
read_hdl -v2001 Input_Muxes.v
read_hdl -v2001 Suzgec_1.v
read_hdl -v2001 Suzgec_2.v
read_hdl -v2001 Suzgec_3_pipe.v
read_hdl -v2001 Output_Muxes.v
read_hdl -v2001 FILTER_END.v
read_hdl -v2001 FILTEROUT_BANK.v
read_hdl -v2001 Suzgec_Uclu.v
elaborate Suzgec_Uclu

#saat isareti tanimlamalari
define_clock -period 1000000 -name sclk -domain domain_1 /designs/Suzgec_Uclu/ports_in/clk0
define_clock -period 5800 -name clk0 -domain domain_2 /designs/Suzgec_Uclu/ports_in/clk0
define_clock -period 5800 -name clk1 -domain domain_2 -divide_period 2 /designs/Suzgec_Uclu/ports_in/clk1
define_clock -period 5800 -name clk2 -domain domain_2 -divide_period 4 /designs/Suzgec_Uclu/ports_in/clk2
define_clock -period 5800 -name clk3 -domain domain_2 -divide_period 8 /designs/Suzgec_Uclu/ports_in/clk3
set_attribute slew {100 100 200 200} [find -clock sclk]
set_attribute slew {50 50 100 100} [find -clock clk0]
set_attribute slew {50 50 100 100} [find -clock clk1]
set_attribute slew {50 50 100 100} [find -clock clk2]
set_attribute slew {50 50 100 100} [find -clock clk3]

external_delay -input 500 -clock [find -clock sclk] -edge_rise /designs/Suzgec_Uclu/ports_in/resetRegOut
external_delay -input 500 -clock [find -clock sclk] -edge_rise /designs/Suzgec_Uclu/ports_in/write
external_delay -input 500 -clock [find -clock sclk] -edge_rise /designs/Suzgec_Uclu/ports_in/CE*
external_delay -output 1000 -clock [find -clock sclk] -edge_rise /designs/Suzgec_Uclu/ports_out/dataR*
external_delay -input 0 -clock [find -clock clk0] -edge_rise /designs/Suzgec_Uclu/ports_in/int_mode*
external_delay -input 0 -clock [find -clock clk0] -edge_rise /designs/Suzgec_Uclu/ports_in/hb*
external_delay -input 0 -clock [find -clock clk0] -edge_rise /designs/Suzgec_Uclu/ports_in/clk*
external_delay -input 0 -clock [find -clock clk0] -edge_rise /designs/Suzgec_Uclu/ports_in/reset
external_delay -input 300 -clock [find -clock clk3] -edge_rise /designs/Suzgec_Uclu/ports_in/din_*
external_delay -output 100 -clock [find -clock clk3] -edge_rise /designs/Suzgec_Uclu/ports_out/dout_*

#retime islemleri
set_attribute dont_retime true [all::all_seqs -clock sclk]
set_attribute dont_retime true [all::all_seqs -clock clk0]
set_attribute dont_retime true [all::all_seqs -clock clk1]
set_attribute dont_retime true [find / -instance ent_out_*]
set_attribute dont_retime true [find / -instance FF_oys_*]
set_attribute dont_retime true [all::all_seqs -clock clk3]
retime -prepare
retime -min_delay -effort high

synthesize -to_mapped -effort high

#cikislari yaz
write_encounter design -basename Suzgec_Uclu_syn
report_timing > timing_report.txt
report area > area_report.txt
```

## APPENDIX S

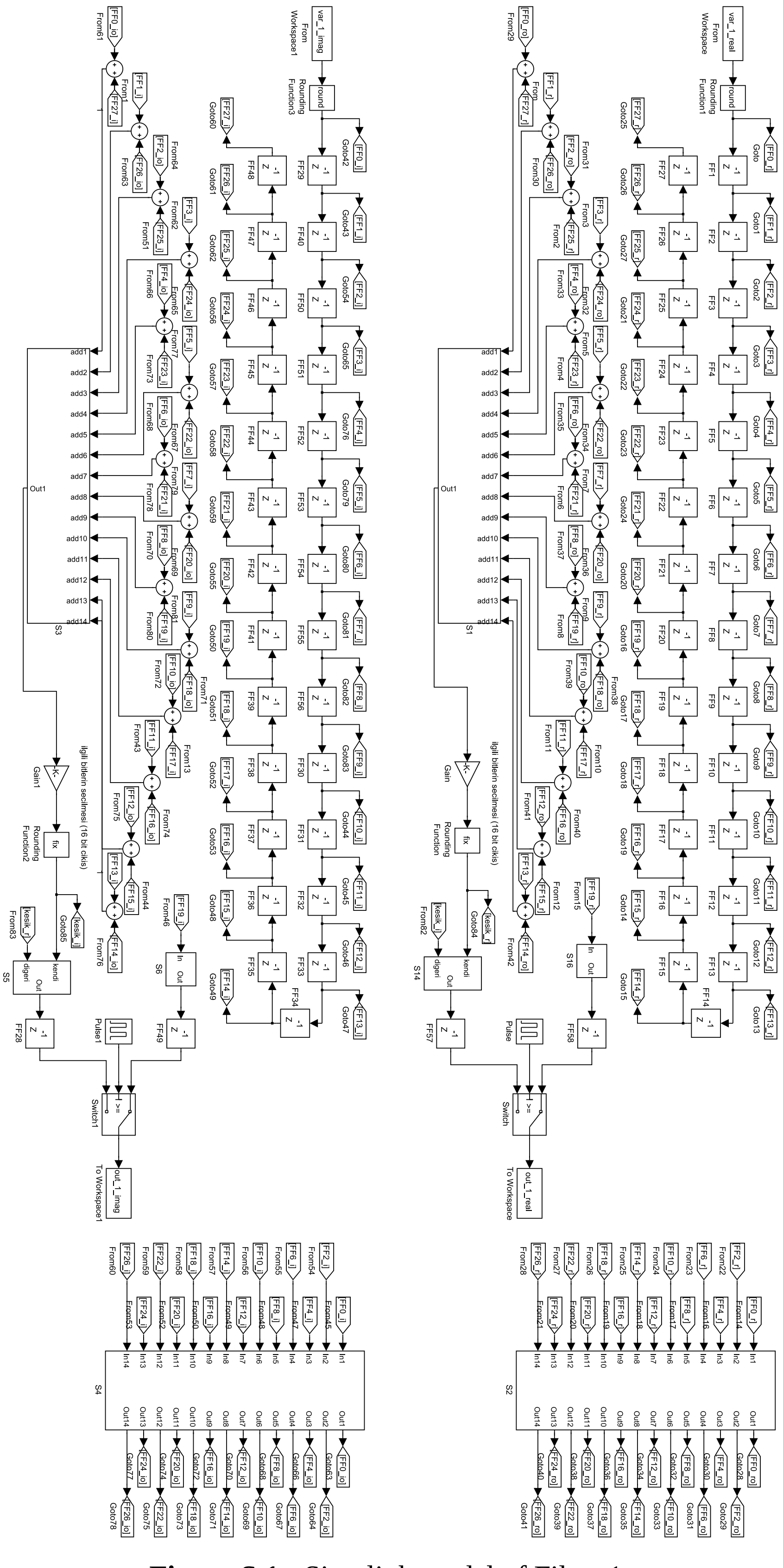


**Figure S.1 :** Simulink model of Filter-1.

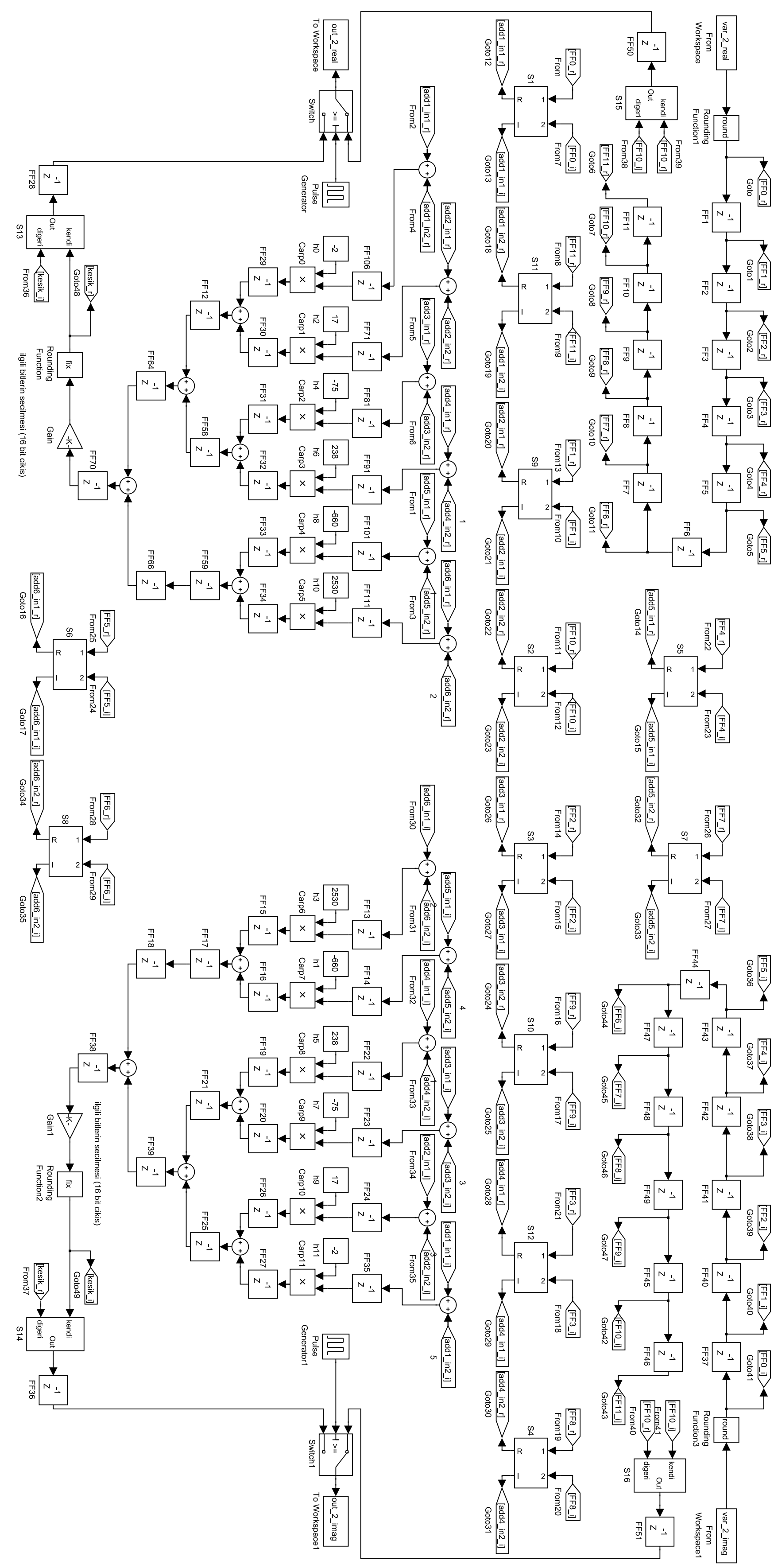


**Figure S.2 :** Simulink model of Filter-2.

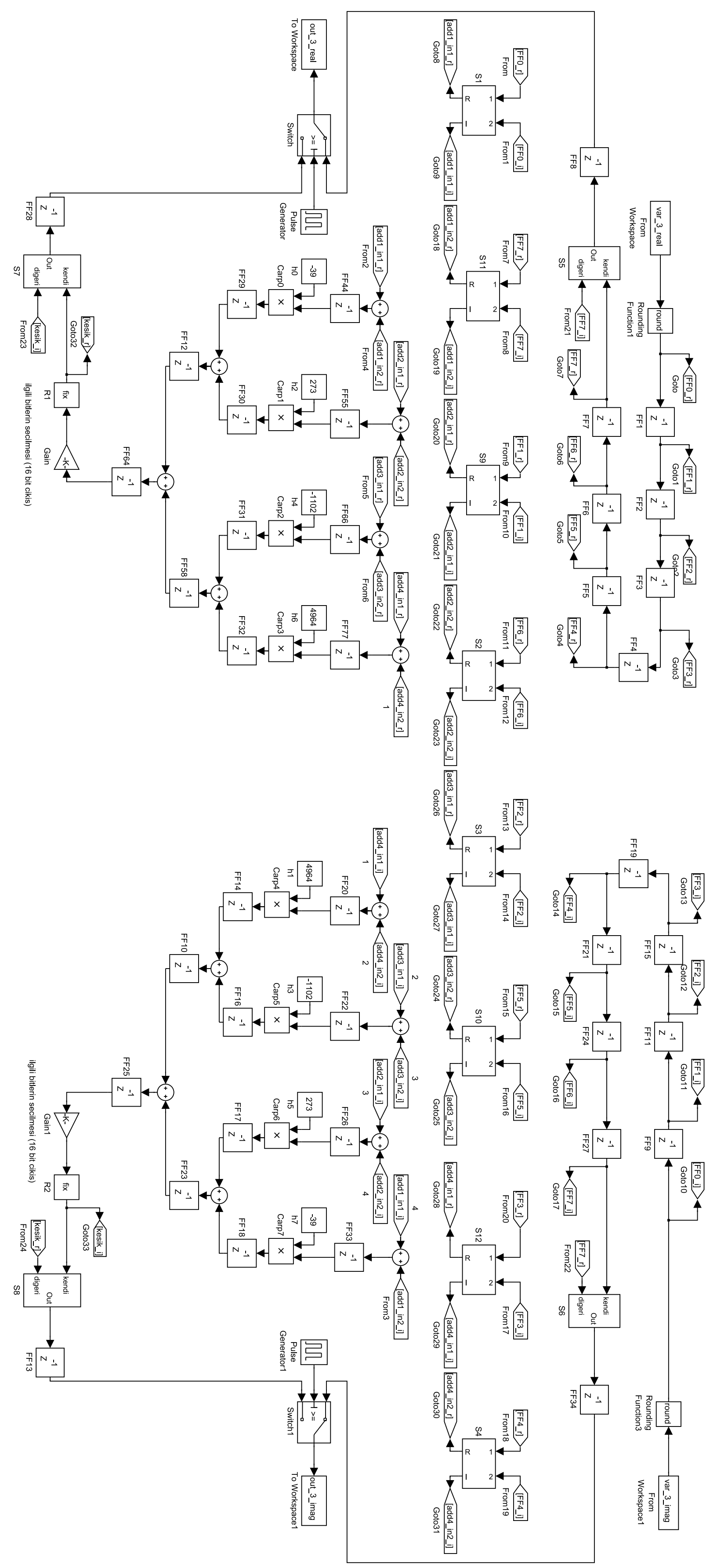


**Figure S.3 :** Simulink model of Filter-3.

## APPENDIX V

### V.1. Verilog Code of Fillter-1

```
`timescale 1ns / 1ps
module Suzgec_1(clk250, clk500, reset, hb1_mod, din_r, din_i, dout_r, dout_i
);
input clk250, clk500, reset;
input[1:0] hb1_mod;
input signed[15:0] din_r, din_i;
output reg signed[15:0] dout_r, dout_i;

reg signed[15:0] FF_r [0:27];
reg signed[15:0] FF_i [0:27];
reg signed[15:0] FF_oys_r, FF_oys_i;
reg signed[16:0] ilk_toplam_r [0:13];
reg signed[16:0] ilk_toplam_i [0:13];
wire signed[15:0] h [0:13];

assign h[0] = -4;
assign h[1] = 13;
assign h[2] = -34;
assign h[3] = 72;
assign h[4] = -138;
assign h[5] = 245;
assign h[6] = -408;
assign h[7] = 650;
assign h[8] = -1003;
assign h[9] = 1521;
assign h[10] = -2315;
assign h[11] = 3671;
assign h[12] = -6642;
assign h[13] = 20755;

reg signed[32:0] carpim_r [0:13];
reg signed[32:0] carpim_i [0:13];
reg signed[33:0] agac_sev_1_r [0:6];
reg signed[33:0] agac_sev_1_i [0:6];
reg signed[34:0] agac_sev_2_r [0:3];
reg signed[34:0] agac_sev_2_i [0:3];
reg signed[35:0] agac_sev_3_r [0:1];
reg signed[35:0] agac_sev_3_i [0:1];
reg signed[36:0] agac_sev_4_r, agac_sev_4_i;
reg signed[15:0] kesik_r, kesik_i;
reg signed[15:0] ent_out_r, ent_out_i;
reg bir_say;
integer i;

//ötelemeli yazıcı
always@(posedge clk250, posedge reset)begin
		if(reset == 1)begin
				for( i=0; i<=27; i=i+1 )begin
						FF_r[i] <= 0;
						FF_i[i] <= 0;
				end
		end
		else begin
				FF_r[0] <= din_r;
				FF_i[0] <= din_i;
				for( i=0; i<=26; i=i+1 )begin
						FF_r[i+1] <= FF_r[i];
						FF_i[i+1] <= FF_i[i];
				end
		end
end
//oys registeri //otelemeli yazici sonu
always@(posedge clk250, posedge reset)begin
		if(reset == 1)begin
				FF_oys_r <= 0;
				FF_oys_i <= 0;
		end
		else begin
```

```
				case(hb1_mod)
					0 : begin
						FF_oys_r <= FF_r[19];
						FF_oys_i <= FF_i[19];
					end
					1 : begin
						FF_oys_r <= -FF_r[19];
						FF_oys_i <= -FF_i[19];
					end
					2 : begin
						FF_oys_r <= -FF_r[19];
						FF_oys_i <= -FF_i[19];
					end
					3 : begin
						FF_oys_r <= FF_r[19];
						FF_oys_i <= FF_i[19];
					end
				endcase
			end
end
//toplayıcılar
always@(posedge clk250, posedge reset)begin
		if(reset == 1)begin
				for( i=0; i<=13; i=i+1 )begin
					ilk_toplam_r[i] <= 0;
					ilk_toplam_i[i] <= 0;
				end
		end
		else begin
				case(hb1_mod)
					0 : begin
						for( i=0; i<=13; i=i+1 )begin
							ilk_toplam_r[i] <= FF_r[i] + FF_r [27-i];
							ilk_toplam_i[i] <= FF_i[i] + FF_i [27-i];
						end
					end
					1 : begin
						for( i=0; i<=12; i=i+2 )begin
							ilk_toplam_r[i] <= -FF_r[i] + FF_r [27-i];
							ilk_toplam_i[i] <= -FF_i[i] + FF_i [27-i];
						end
						for( i=1; i<=13; i=i+2 )begin
							ilk_toplam_r[i] <= FF_r[i] - FF_r [27-i];
							ilk_toplam_i[i] <= FF_i[i] - FF_i [27-i];
						end
					end
					2 : begin
						for( i=0; i<=13; i=i+1 )begin
							ilk_toplam_r[i] <= FF_r[i] + FF_r [27-i];
							ilk_toplam_i[i] <= FF_i[i] + FF_i [27-i];
						end
					end
					3 : begin
						for( i=0; i<=12; i=i+2 )begin
							ilk_toplam_r[i] <= -FF_r[i] + FF_r [27-i];
							ilk_toplam_i[i] <= -FF_i[i] + FF_i [27-i];
						end
						for( i=1; i<=13; i=i+2 )begin
							ilk_toplam_r[i] <= FF_r[i] - FF_r [27-i];
							ilk_toplam_i[i] <= FF_i[i] - FF_i [27-i];
						end
					end
				endcase
		end
end
//carpicilar
always@(posedge clk250, posedge reset)begin
		if(reset == 1)begin
				for( i=0; i<=13; i=i+1 )begin
					carpim_r[i] <= 0;
					carpim_i[i] <= 0;
				end
		end
		else begin
				for( i=0; i<=13; i=i+1 )begin
				carpim_r[i] <= ilk_toplam_r[i] * h[i];
				carpim_i[i] <= ilk_toplam_i[i] * h[i];
			end
		end
end
//toplayici agaci seviye 1
always@(posedge clk250, posedge reset)begin
		if(reset == 1)begin
			for( i=0; i<=6; i=i+1 )begin
				agac_sev_1_r[i] <= 0;
				agac_sev_1_i[i] <= 0;
			end
		end
		else begin
			for( i=0; i<=6; i=i+1 )begin
				agac_sev_1_r[i] <= carpim_r[2*i] + carpim_r[2*i+1];
				agac_sev_1_i[i] <= carpim_i[2*i] + carpim_i[2*i+1];
			end
		end
end
//toplayici agaci seviye 2
always@(posedge clk250, posedge reset)begin
		if(reset == 1)begin
			for( i=0; i<=3; i=i+1 )begin
				agac_sev_2_r[i] <= 0;
				agac_sev_2_i[i] <= 0;
			end
		end
		else begin
			agac_sev_2_r[3] <= agac_sev_1_r[6];
			agac_sev_2_i[3] <= agac_sev_1_i[6];
			for( i=0; i<=2; i=i+1 )begin
				agac_sev_2_r[i] <= agac_sev_1_r[2*i] + agac_sev_1_r[2*i+1];
				agac_sev_2_i[i] <= agac_sev_1_i[2*i] + agac_sev_1_i[2*i+1];
			end
		end
end
//toplayici agaci seviye 3
always@(posedge clk250, posedge reset)begin
		if(reset == 1)begin
			agac_sev_3_r[0] <= 0;
			agac_sev_3_r[1] <= 0;
			agac_sev_3_i[0] <= 0;
			agac_sev_3_i[1] <= 0;
		end
		else begin
			agac_sev_3_r[0] <= agac_sev_2_r[0] + agac_sev_2_r[1];
			agac_sev_3_r[1] <= agac_sev_2_r[2] + agac_sev_2_r[3];
			//
			agac_sev_3_i[0] <= agac_sev_2_i[0] + agac_sev_2_i[1];
			agac_sev_3_i[1] <= agac_sev_2_i[2] + agac_sev_2_i[3];
		end
end
//toplayici agaci seviye 4
always@(posedge clk250, posedge reset)begin
		if(reset == 1)begin
			agac_sev_4_r <= 0;
			agac_sev_4_i <= 0;
		end
		else begin
			agac_sev_4_r <= agac_sev_3_r[0] + agac_sev_3_r[1];
			agac_sev_4_i <= agac_sev_3_i[0] + agac_sev_3_i[1];
		end
end
//37bit - 16 bit donusturucu (cikis kesicisi)
always@(*)begin
		if(agac_sev_4_r[30] == 1)begin			//sayi negatif ise
			if( |agac_sev_4_r[14:0] == 1 )begin				//sayi, tamsayisinden küçükse
				kesik_r <= agac_sev_4_r[30:15] + 1;
			end
			else begin
				kesik_r <= agac_sev_4_r[30:15];
			end
		end
		else begin
```

```
                kesik_r <= agac_sev_4_r[30:15];
        end
        //
        if(agac_sev_4_i[30] == 1)begin              //sayi negatif ise
                if( |agac_sev_4_i[14:0] == 1 )begin                 //sayi, tamsayisinden küçükse
                        kesik_i <= agac_sev_4_i[30:15] + 1;
                end
                else begin
                        kesik_i <= agac_sev_4_i[30:15];
                end
        end
        else begin
                kesik_i <= agac_sev_4_i[30:15];
        end
end
//enterpolasyon sonrasi cikis
always@(posedge clk250, posedge reset)begin
        if(reset == 1)begin
                ent_out_r <= 0;
                ent_out_i <= 0;
        end
        else begin
                case(hb1_mod)

                        1 : begin
                                ent_out_r <= kesik_i;
                                ent_out_i <= kesik_r;
                        end

                        3 : begin
                                ent_out_r <= kesik_i;
                                ent_out_i <= kesik_r;
                        end

                        default : begin
                                ent_out_r <= kesik_r;
                                ent_out_i <= kesik_i;
                        end

                endcase
        end
end
//cikis secicisi
always@(posedge clk500, posedge reset)begin
        if(reset == 1)begin
                dout_r <= 0;
                dout_i <= 0;
        end
        else begin
                if(bir_say == 1)begin
                        dout_r <= ent_out_r;
                        dout_i <= ent_out_i;
                end
                else begin
                        dout_r <= FF_oys_r;
                        dout_i <= FF_oys_i;
                end
        end
end
//bire kadar sayan sayici
always@(posedge clk500, posedge reset)begin
        if(reset == 1)begin
                bir_say <= 0;
        end
        else begin
                bir_say <= ~bir_say;
        end
end
endmodule
```

## V.2. Verilog Code of Fillter-2

```
`timescale 1ns / 1ps
module Suzgec_2(clk500, clk1, reset, hb2_mod, din_r, din_i, dout_r, dout_i
);
input clk500, clk1, reset;
input[2:0] hb2_mod;
input signed[15:0] din_r, din_i;
output reg signed[15:0] dout_r, dout_i;

reg signed[15:0] FF_r [0:11];
reg signed[15:0] FF_i [0:11];
reg signed[15:0] FF_oys_r, FF_oys_i;
reg signed[16:0] toplam_oys_r, toplam_oys_i;
wire signed[16:0] sqrt2;

assign sqrt2 = 17'b01011010100000101;                    //16'd46341

wire signed[33:0] carpim_oys_r, carpim_oys_i;
reg signed[15:0] kesik_oys_r, kesik_oys_i;
reg signed[16:0] ilk_toplam_r [0:5];
reg signed[16:0] ilk_toplam_i [0:5];
wire signed[12:0] h [0:5];

assign h[0] = -2;
assign h[1] = 17;
assign h[2] = -75;
assign h[3] = 238;
assign h[4] = -660;
assign h[5] = 2530;

reg signed[29:0] carpim_r [0:5];
reg signed[29:0] carpim_i [0:5];
reg signed[30:0] agac_sev_1_r [0:2];
reg signed[30:0] agac_sev_1_i [0:2];
reg signed[31:0] agac_sev_2_r [0:1];
reg signed[31:0] agac_sev_2_i [0:1];
reg signed[32:0] agac_sev_3_r, agac_sev_3_i;
reg signed[15:0] kesik_r, kesik_i;
reg signed[15:0] ent_out_r, ent_out_i;
reg bir_say;
integer i;

//ötelemeli yazıcı
always@(posedge clk500, posedge reset)begin
        if(reset == 1)begin
                for( i=0; i<=11; i=i+1 )begin
                        FF_r[i] <= 0;
                        FF_i[i] <= 0;
                end
        end
        else begin
                FF_r[0] <= din_r;
                FF_i[0] <= din_i;
                //FF[1] <= FF[0];
                //...
                //FF[11] <= FF[10];
                for( i=0; i<=10; i=i+1 )begin
                        FF_r[i+1] <= FF_r[i];
                        FF_i[i+1] <= FF_i[i];
                end
        end
end

//ötelemeli yazici sonu (oys) toplayici
always@(*)begin
        case(hb2_mod)
                1 : begin
                        toplam_oys_r = FF_r[10] - FF_i[10];
                        toplam_oys_i = FF_i[10] + FF_r[10];
                end
                3 : begin
                        toplam_oys_r = FF_r[10] + FF_i[10];
                        toplam_oys_i = FF_i[10] - FF_r[10];
```

```
			end
			5 : begin
				toplam_oys_r = FF_r[10] - FF_i[10];
				toplam_oys_i = FF_i[10] + FF_r[10];
			end
			7 : begin
				toplam_oys_r = FF_r[10] + FF_i[10];
				toplam_oys_i = FF_i[10] - FF_r[10];
			end
			default : begin
				toplam_oys_r = FF_r[10] + FF_i[10];
				toplam_oys_i = FF_i[10] + FF_r[10];
			end
		endcase
end

//ötelemeli yazici sonu carpici (kök 2 ile çarpım)
assign carpim_oys_r = toplam_oys_r * sqrt2;
assign carpim_oys_i = toplam_oys_i * sqrt2;

//oys kesicisi (2'ye bolme) (ya da 4e denebilir belki)
always@(*)begin
		if(carpim_oys_r[31] == 1)begin			//sayi negatif ise
			if( |carpim_oys_r[15:0] == 1 )begin			//sayi, tamsayisinden küçükse
				kesik_oys_r <= carpim_oys_r[31:16] + 1;
			end
			else begin
				kesik_oys_r <= carpim_oys_r[31:16];
			end
		end
		else begin
			kesik_oys_r <= carpim_oys_r[31:16];
		end
		if(carpim_oys_i[31] == 1)begin			//sayi negatif ise
			if( |carpim_oys_i[15:0] == 1 )begin			//sayi, tamsayisinden küçükse
				kesik_oys_i <= carpim_oys_i[31:16] + 1;
			end
			else begin
				kesik_oys_i <= carpim_oys_i[31:16];
			end
		end
		else begin
			kesik_oys_i <= carpim_oys_i[31:16];
		end

end
//oys registeri
always@(posedge clk500, posedge reset)begin
		if(reset == 1)begin
			FF_oys_r <= 0;
			FF_oys_i <= 0;
		end
		else begin
			case(hb2_mod)
				0 : begin
					FF_oys_r <= FF_r[10];
					FF_oys_i <= FF_i[10];
				end
				1 : begin
					FF_oys_r <= -kesik_oys_r;
					FF_oys_i <= -kesik_oys_i;
				end
				2 : begin
					FF_oys_r <= -FF_r[10];
					FF_oys_i <= -FF_i[10];
				end
				3 : begin
					FF_oys_r <= kesik_oys_r;
					FF_oys_i <= kesik_oys_i;
				end
				4 : begin
					FF_oys_r <= -FF_r[10];
					FF_oys_i <= -FF_i[10];
				end
				5 : begin
```

```
                        FF_oys_r <= kesik_oys_r;
                        FF_oys_i <= kesik_oys_i;
                    end
                    6 : begin
                        FF_oys_r <= FF_r[10];
                        FF_oys_i <= FF_i[10];
                    end
                    7 : begin
                        FF_oys_r <= -kesik_oys_r;
                        FF_oys_i <= -kesik_oys_i;
                    end
                    default : begin
                        FF_oys_r <= FF_r[10];
                        FF_oys_i <= FF_i[10];
                    end
                endcase
            end
end

//toplayıcılar
always@(posedge clk500, posedge reset)begin
        if(reset == 1)begin
                for( i=0; i<=5; i=i+1 )begin
                        ilk_toplam_r[i] <= 0;
                        ilk_toplam_i[i] <= 0;
                end
        end
        else begin

                case(hb2_mod)
                        0 : begin
                                ilk_toplam_r[0] <= FF_r[0] + FF_r[11];          //add1
                                ilk_toplam_r[1] <= FF_r[1] + FF_r[10];          //add2
                                ilk_toplam_r[2] <= FF_r[2] + FF_r[9];           //add3
                                ilk_toplam_r[3] <= FF_r[3] + FF_r[8];           //add4
                                ilk_toplam_r[4] <= FF_r[4] + FF_r[7];           //add5
                                ilk_toplam_r[5] <= FF_r[5] + FF_r[6];           //add6
                                //imags
                                ilk_toplam_i[0] <= FF_i[0] + FF_i[11];          //add1
                                ilk_toplam_i[1] <= FF_i[1] + FF_i[10];          //add2
                                ilk_toplam_i[2] <= FF_i[2] + FF_i[9];           //add3
                                ilk_toplam_i[3] <= FF_i[3] + FF_i[8];           //add4
                                ilk_toplam_i[4] <= FF_i[4] + FF_i[7];           //add5
                                ilk_toplam_i[5] <= FF_i[5] + FF_i[6];           //add6
                        end
                        1 : begin
                                ilk_toplam_r[0] <= -FF_i[0] + FF_r[11];     //add1
                                ilk_toplam_r[1] <= -FF_r[1] + FF_i[10];     //add2
                                ilk_toplam_r[2] <= FF_i[2] - FF_r[9];           //add3
                                ilk_toplam_r[3] <= FF_r[3] - FF_i[8];           //add4
                                ilk_toplam_r[4] <= -FF_i[4] + FF_r[7];          //add5
                                ilk_toplam_r[5] <= -FF_r[5] + FF_i[6];          //add6
                                //imags
                                ilk_toplam_i[0] <= FF_r[0] + FF_i[11];          //add1
                                ilk_toplam_i[1] <= -FF_i[1] - FF_r[10];     //add2
                                ilk_toplam_i[2] <= -FF_r[2] - FF_i[9];          //add3
                                ilk_toplam_i[3] <= FF_i[3] + FF_r[8];           //add4
                                ilk_toplam_i[4] <= FF_r[4] + FF_i[7];           //add5
                                ilk_toplam_i[5] <= -FF_i[5] - FF_r[6];          //add6
                        end
                        2 : begin
                                ilk_toplam_r[0] <= -FF_r[0] + FF_r[11];     //add1
                                ilk_toplam_r[1] <= FF_r[1] - FF_r[10];          //add2
                                ilk_toplam_r[2] <= -FF_r[2] + FF_r[9];          //add3
                                ilk_toplam_r[3] <= FF_r[3] - FF_r[8];           //add4
                                ilk_toplam_r[4] <= -FF_r[4] + FF_r[7];          //add5
                                ilk_toplam_r[5] <= FF_r[5] - FF_r[6];           //add6
                                //imags
                                ilk_toplam_i[0] <= FF_i[0] - FF_i[11];          //add1
                                ilk_toplam_i[1] <= -FF_i[1] + FF_i[10];     //add2
                                ilk_toplam_i[2] <= FF_i[2] - FF_i[9];           //add3
                                ilk_toplam_i[3] <= -FF_i[3] + FF_i[8];          //add4
                                ilk_toplam_i[4] <= FF_i[4] - FF_i[7];           //add5
                                ilk_toplam_i[5] <= -FF_i[5] + FF_i[6];          //add6
                        end
```

```
3 : begin
        ilk_toplam_r[0] <= FF_i[0] + FF_r[11];                  //add1
        ilk_toplam_r[1] <= -FF_r[1] - FF_i[10];     //add2
        ilk_toplam_r[2] <= -FF_i[2] - FF_r[9];                  //add3
        ilk_toplam_r[3] <= FF_r[3] + FF_i[8];                   //add4
        ilk_toplam_r[4] <= FF_i[4] + FF_r[7];                   //add5
        ilk_toplam_r[5] <= -FF_r[5] - FF_i[6];                  //add6
        //imags
        ilk_toplam_i[0] <= -FF_r[0] + FF_i[11];     //add1
        ilk_toplam_i[1] <= -FF_i[1] + FF_r[10];     //add2
        ilk_toplam_i[2] <= FF_r[2] - FF_i[9];                   //add3
        ilk_toplam_i[3] <= FF_i[3] - FF_r[8];                   //add4
        ilk_toplam_i[4] <= -FF_r[4] + FF_i[7];                  //add5
        ilk_toplam_i[5] <= -FF_i[5] + FF_r[6];                  //add6
end
4 : begin
        ilk_toplam_r[0] <= FF_r[0] + FF_r[11];                  //add1
        ilk_toplam_r[1] <= FF_r[1] + FF_r[10];                  //add2
        ilk_toplam_r[2] <= FF_r[2] + FF_r[9];                   //add3
        ilk_toplam_r[3] <= FF_r[3] + FF_r[8];                   //add4
        ilk_toplam_r[4] <= FF_r[4] + FF_r[7];                   //add5
        ilk_toplam_r[5] <= FF_r[5] + FF_r[6];                   //add6
        //imags
        ilk_toplam_i[0] <= FF_i[0] + FF_i[11];                  //add1
        ilk_toplam_i[1] <= FF_i[1] + FF_i[10];                  //add2
        ilk_toplam_i[2] <= FF_i[2] + FF_i[9];                   //add3
        ilk_toplam_i[3] <= FF_i[3] + FF_i[8];                   //add4
        ilk_toplam_i[4] <= FF_i[4] + FF_i[7];                   //add5
        ilk_toplam_i[5] <= FF_i[5] + FF_i[6];                   //add6
end
5 : begin
        ilk_toplam_r[0] <= -FF_i[0] + FF_r[11];     //add1
        ilk_toplam_r[1] <= -FF_r[1] + FF_i[10];     //add2
        ilk_toplam_r[2] <= FF_i[2] - FF_r[9];                   //add3
        ilk_toplam_r[3] <= FF_r[3] - FF_i[8];                   //add4
        ilk_toplam_r[4] <= -FF_i[4] + FF_r[7];                  //add5
        ilk_toplam_r[5] <= -FF_r[5] + FF_i[6];                  //add6
        //imags
        ilk_toplam_i[0] <= FF_r[0] + FF_i[11];                  //add1
        ilk_toplam_i[1] <= -FF_i[1] - FF_r[10];     //add2
        ilk_toplam_i[2] <= -FF_r[2] - FF_i[9];                  //add3
        ilk_toplam_i[3] <= FF_i[3] + FF_r[8];                   //add4
        ilk_toplam_i[4] <= FF_r[4] + FF_i[7];                   //add5
        ilk_toplam_i[5] <= -FF_i[5] - FF_r[6];                  //add6

end
6 : begin
        ilk_toplam_r[0] <= -FF_r[0] + FF_r[11];     //add1
        ilk_toplam_r[1] <= FF_r[1] - FF_r[10];                  //add2
        ilk_toplam_r[2] <= -FF_r[2] + FF_r[9];                  //add3
        ilk_toplam_r[3] <= FF_r[3] - FF_r[8];                   //add4
        ilk_toplam_r[4] <= -FF_r[4] + FF_r[7];                  //add5
        ilk_toplam_r[5] <= FF_r[5] - FF_r[6];                   //add6
        //imags
        ilk_toplam_i[0] <= FF_i[0] - FF_i[11];                  //add1
        ilk_toplam_i[1] <= -FF_i[1] + FF_i[10];     //add2
        ilk_toplam_i[2] <= FF_i[2] - FF_i[9];                   //add3
        ilk_toplam_i[3] <= -FF_i[3] + FF_i[8];                  //add4
        ilk_toplam_i[4] <= FF_i[4] - FF_i[7];                   //add5
        ilk_toplam_i[5] <= -FF_i[5] + FF_i[6];                  //add6
end
7 : begin
        ilk_toplam_r[0] <= FF_i[0] + FF_r[11];                  //add1
        ilk_toplam_r[1] <= -FF_r[1] - FF_i[10];     //add2
        ilk_toplam_r[2] <= -FF_i[2] - FF_r[9];                  //add3
        ilk_toplam_r[3] <= FF_r[3] + FF_i[8];                   //add4
        ilk_toplam_r[4] <= FF_i[4] + FF_r[7];                   //add5
        ilk_toplam_r[5] <= -FF_r[5] - FF_i[6];                  //add6
        //imags
        ilk_toplam_i[0] <= -FF_r[0] + FF_i[11];     //add1
        ilk_toplam_i[1] <= -FF_i[1] + FF_r[10];     //add2
        ilk_toplam_i[2] <= FF_r[2] - FF_i[9];                   //add3
        ilk_toplam_i[3] <= FF_i[3] - FF_r[8];                   //add4
        ilk_toplam_i[4] <= -FF_r[4] + FF_i[7];                  //add5
        ilk_toplam_i[5] <= -FF_i[5] + FF_r[6];                  //add6
```

```
					end
				endcase
			end
end

//carpicilar
always@(posedge clk500, posedge reset)begin
			if(reset == 1)begin
				for( i=0; i<=5; i=i+1 )begin
					carpim_r[i] <= 0;
					carpim_i[i] <= 0;
				end
			end
			else begin
				for( i=0; i<=5; i=i+1 )begin
					carpim_r[i] <= ilk_toplam_r[i] * h[i];
					carpim_i[i] <= ilk_toplam_i[i] * h[i];
				end
			end
end

//toplayici agaci seviye 1
always@(posedge clk500, posedge reset)begin
			if(reset == 1)begin
				for( i=0; i<=2; i=i+1 )begin
					agac_sev_1_r[i] <= 0;
					agac_sev_1_i[i] <= 0;
				end
			end
			else begin
				for( i=0; i<=2; i=i+1 )begin
					agac_sev_1_r[i] <= carpim_r[2*i] + carpim_r[2*i+1];
					agac_sev_1_i[i] <= carpim_i[2*i] + carpim_i[2*i+1];
				end
			end
end

//toplayici agaci seviye 2
always@(posedge clk500, posedge reset)begin
			if(reset == 1)begin
				for( i=0; i<=1; i=i+1 )begin
					agac_sev_2_r[i] <= 0;
					agac_sev_2_i[i] <= 0;
				end
			end
			else begin
				agac_sev_2_r[0] <= agac_sev_1_r[0] + agac_sev_1_r[1];
				agac_sev_2_r[1] <= agac_sev_1_r[2];
				//
				agac_sev_2_i[0] <= agac_sev_1_i[0] + agac_sev_1_i[1];
				agac_sev_2_i[1] <= agac_sev_1_i[2];
			end
end

//toplayici agaci seviye 3
always@(posedge clk500, posedge reset)begin
			if(reset == 1)begin
				agac_sev_3_r <= 0;
				agac_sev_3_i <= 0;
			end
			else begin
				agac_sev_3_r <= agac_sev_2_r[0] + agac_sev_2_r[1];
				agac_sev_3_i <= agac_sev_2_i[0] + agac_sev_2_i[1];
			end
end

//36bit - 16 bit donusturucu (cikis kesicisi)
always@(*)begin
			if(agac_sev_3_r[27] == 1)begin			//sayi negatif ise
				if( |agac_sev_3_r[11:0] == 1 )begin				//sayi, tamsayisinden küçükse
					kesik_r <= agac_sev_3_r[27:12] + 1;
				end
				else begin
					kesik_r <= agac_sev_3_r[27:12];
				end
```

```
		end
		else begin
			kesik_r <= agac_sev_3_r[27:12];
		end
		//
		if(agac_sev_3_i[27] == 1)begin			//sayi negatif ise
			if( |agac_sev_3_i[11:0] == 1 )begin			//sayi, tamsayisinden küçükse
				kesik_i <= agac_sev_3_i[27:12] + 1;
			end
			else begin
				kesik_i <= agac_sev_3_i[27:12];
			end
		end
		else begin
			kesik_i <= agac_sev_3_i[27:12];
		end
end

//enterpolasyon sonrasi cikis
always@(posedge clk500, posedge reset)begin
		if(reset == 1)begin
			ent_out_r <= 0;
			ent_out_i <= 0;
		end
		else begin
			case(hb2_mod)
				2 : begin
					ent_out_r <= kesik_i;
					ent_out_i <= kesik_r;
				end
				6 : begin
					ent_out_r <= kesik_i;
					ent_out_i <= kesik_r;
				end
				default : begin
					ent_out_r <= kesik_r;
					ent_out_i <= kesik_i;
				end
			endcase
		end
end

//cikis secicisi
always@(posedge clk1, posedge reset)begin
		if(reset == 1)begin
			dout_r <= 0;
			dout_i <= 0;
		end
		else begin
			if(bir_say == 0)begin
				dout_r <= ent_out_r;
				dout_i <= ent_out_i;
			end
			else begin
				dout_r <= FF_oys_r;
				dout_i <= FF_oys_i;
			end
		end
end

//bire kadar sayan sayici
always@(posedge clk1, posedge reset)begin
		if(reset == 1)begin
			bir_say <= 0;
		end
		else begin
			bir_say <= ~bir_say;
		end
end

endmodule
```

## V.3. Verilog Code of Fillter-3

```
`timescale 1ns / 1ps
module Suzgec_3(clk1, clk2, reset, hb3_mod, din_r, din_i, dout_r, dout_i
);
input clk1, clk2, reset;
input[2:0] hb3_mod;
input signed[15:0] din_r, din_i;
output reg signed[15:0] dout_r, dout_i;

reg signed[15:0] FF_r [0:8];                // bu FF'lardan 8.si sadece gecikme için kullanılıyor, çıkışı toplanıp çarpılmıyor
reg signed[15:0] FF_i [0:8];
reg signed[16:0] toplam_oys_r, toplam_oys_i;
wire signed[16:0] sqrt2;

assign sqrt2 = 17'sb01011010100000101;                //16'd46341

wire signed[33:0] carpim_oys_r, carpim_oys_i;
reg signed[15:0] kesik_oys_r, kesik_oys_i;
reg signed[16:0] ilk_toplam_r [0:3];
reg signed[16:0] ilk_toplam_i [0:3];
reg signed[30:0] carpim_r [0:3];
reg signed[30:0] carpim_i [0:3];
wire signed[13:0] h [0:3];

assign h[0] = -39;
assign h[1] = 273;
assign h[2] = -1102;
assign h[3] = 4964;
reg signed[31:0] agac_sev_1_r [0:1];
reg signed[31:0] agac_sev_1_i [0:1];
reg signed[32:0] agac_sev_2_r, agac_sev_2_i;
reg signed[15:0] kesik_r, kesik_i;
reg signed[15:0] ent_out_r, ent_out_i;
reg bir_say;
integer i;

//ötelemeli yazıcı
always@(posedge clk1, posedge reset)begin
        if(reset == 1)begin
                for( i=0; i<=7; i=i+1 )begin
                        FF_r[i] <= 0;
                        FF_i[i] <= 0;
                end
        end
        else begin
                FF_r[0] <= din_r;
                FF_i[0] <= din_i;
                for( i=0; i<=6; i=i+1 )begin
                        FF_r[i+1] <= FF_r[i];
                        FF_i[i+1] <= FF_i[i];
                end
        end
end
//ötelemeli yazici sonu (oys) toplayici
always@(*)begin
        case(hb3_mod)
                1 : begin
                        toplam_oys_r = FF_r[7] - FF_i[7];
                        toplam_oys_i = FF_i[7] + FF_r[7];
                end
                3 : begin
                        toplam_oys_r = FF_r[7] + FF_i[7];
                        toplam_oys_i = FF_i[7] - FF_r[7];
                end
                5 : begin
                        toplam_oys_r = FF_r[7] - FF_i[7];
                        toplam_oys_i = FF_i[7] + FF_r[7];
                end
                7 : begin
                        toplam_oys_r = FF_r[7] + FF_i[7];
                        toplam_oys_i = FF_i[7] - FF_r[7];
                end
```

```
					default : begin
							toplam_oys_r = FF_r[7] + FF_i[7];
							toplam_oys_i = FF_i[7] + FF_r[7];
					end
			endcase
end
//ötelemeli yazici sonu carpici (kök 2 ile çarpım)
assign carpim_oys_r = toplam_oys_r * sqrt2;
assign carpim_oys_i = toplam_oys_i * sqrt2;
//oys kesicisi (2'ye bolme) (ya da 4e denebilir belki)
always@(*)begin
	if(carpim_oys_r[31] == 1)begin			//sayi negatif ise
		if( |carpim_oys_r[15:0] == 1 )begin			//sayi, tamsayisinden küçükse
			kesik_oys_r <= carpim_oys_r[31:16] + 1;
		end
		else begin
			kesik_oys_r <= carpim_oys_r[31:16];
		end
	end
	else begin
		kesik_oys_r <= carpim_oys_r[31:16];
	end
	if(carpim_oys_i[31] == 1)begin			//sayi negatif ise
		if( |carpim_oys_i[15:0] == 1 )begin			//sayi, tamsayisinden küçükse
			kesik_oys_i <= carpim_oys_i[31:16] + 1;
		end
		else begin
			kesik_oys_i <= carpim_oys_i[31:16];
		end
	end
	else begin
		kesik_oys_i <= carpim_oys_i[31:16];
	end
end
//oys registeri
always@(posedge clk1, posedge reset)begin
	if(reset == 1)begin
		FF_r[8] <= 0;
		FF_i[8] <= 0;
	end
	else begin
		case(hb3_mod)
			0 : begin
				FF_r[8] <= FF_r[7];
				FF_i[8] <= FF_i[7];
			end
			1 : begin
				FF_r[8] <= kesik_oys_r;
				FF_i[8] <= kesik_oys_i;
			end
			2 : begin
				FF_r[8] <= -FF_r[7];
				FF_i[8] <= -FF_i[7];
			end
			3 : begin
				FF_r[8] <= -kesik_oys_r;
				FF_i[8] <= -kesik_oys_i;
			end
			4 : begin
				FF_r[8] <= -FF_r[7];
				FF_i[8] <= -FF_i[7];
			end
			5 : begin
				FF_r[8] <= -kesik_oys_r;
				FF_i[8] <= -kesik_oys_i;
			end
			6 : begin
				FF_r[8] <= FF_r[7];
				FF_i[8] <= FF_i[7];
			end
			7 : begin
				FF_r[8] <= kesik_oys_r;
				FF_i[8] <= kesik_oys_i;
			end
			default : begin
```

```
                        FF_r[8] <= FF_r[7];
                        FF_i[8] <= FF_i[7];
                    end
                endcase
            end
end
//toplayıcılar
always@(posedge clk1, posedge reset)begin
        if(reset == 1)begin
                for( i=0; i<=3; i=i+1 )begin
                        ilk_toplam_r[i] <= 0;
                        ilk_toplam_i[i] <= 0;
                end
        end
        else begin
                case(hb3_mod)
                        0 : begin
                                ilk_toplam_r[0] <= FF_r[0] + FF_r[7];           //add1
                                ilk_toplam_r[1] <= FF_r[1] + FF_r[6];           //add2
                                ilk_toplam_r[2] <= FF_r[2] + FF_r[5];           //add3
                                ilk_toplam_r[3] <= FF_r[3] + FF_r[4];           //add4
                                //imags
                                ilk_toplam_i[0] <= FF_i[0] + FF_i[7];           //add1
                                ilk_toplam_i[1] <= FF_i[1] + FF_i[6];           //add2
                                ilk_toplam_i[2] <= FF_i[2] + FF_i[5];           //add3
                                ilk_toplam_i[3] <= FF_i[3] + FF_i[4];           //add4
                        end
                        1 : begin
                                ilk_toplam_r[0] <= -FF_i[0] + FF_r[7];          //add1
                                ilk_toplam_r[1] <= -FF_r[1] + FF_i[6];          //add2
                                ilk_toplam_r[2] <= FF_i[2] - FF_r[5];           //add3
                                ilk_toplam_r[3] <= FF_r[3] - FF_i[4];           //add4
                                //imags
                                ilk_toplam_i[0] <= FF_r[0] + FF_i[7];           //add1
                                ilk_toplam_i[1] <= -FF_i[1] - FF_r[6];          //add2
                                ilk_toplam_i[2] <= -FF_r[2] - FF_i[5];          //add3
                                ilk_toplam_i[3] <= FF_i[3] + FF_r[4];           //add4
                        end
                        2 : begin
                                ilk_toplam_r[0] <= -FF_r[0] + FF_r[7];          //add1
                                ilk_toplam_r[1] <= FF_r[1] - FF_r[6];           //add2
                                ilk_toplam_r[2] <= -FF_r[2] + FF_r[5];          //add3
                                ilk_toplam_r[3] <= FF_r[3] - FF_r[4];           //add4
                                //imags
                                ilk_toplam_i[0] <= FF_i[0] - FF_i[7];           //add1
                                ilk_toplam_i[1] <= -FF_i[1] + FF_i[6];          //add2
                                ilk_toplam_i[2] <= FF_i[2] - FF_i[5];           //add3
                                ilk_toplam_i[3] <= -FF_i[3] + FF_i[4];          //add4
                        end
                        3 : begin
                                ilk_toplam_r[0] <= FF_i[0] + FF_r[7];           //add1
                                ilk_toplam_r[1] <= -FF_r[1] - FF_i[6];          //add2
                                ilk_toplam_r[2] <= -FF_i[2] - FF_r[5];          //add3
                                ilk_toplam_r[3] <= FF_r[3] + FF_i[4];           //add4
                                //imags
                                ilk_toplam_i[0] <= -FF_r[0] + FF_i[7];          //add1
                                ilk_toplam_i[1] <= -FF_i[1] + FF_r[6];          //add2
                                ilk_toplam_i[2] <= FF_r[2] - FF_i[5];           //add3
                                ilk_toplam_i[3] <= FF_i[3] - FF_r[4];           //add4
                        end
                        4 : begin
                                ilk_toplam_r[0] <= FF_r[0] + FF_r[7];           //add1
                                ilk_toplam_r[1] <= FF_r[1] + FF_r[6];           //add2
                                ilk_toplam_r[2] <= FF_r[2] + FF_r[5];           //add3
                                ilk_toplam_r[3] <= FF_r[3] + FF_r[4];           //add4
                                //imags
                                ilk_toplam_i[0] <= FF_i[0] + FF_i[7];           //add1
                                ilk_toplam_i[1] <= FF_i[1] + FF_i[6];           //add2
                                ilk_toplam_i[2] <= FF_i[2] + FF_i[5];           //add3
                                ilk_toplam_i[3] <= FF_i[3] + FF_i[4];           //add4
                        end
                        5 : begin
                                ilk_toplam_r[0] <= -FF_i[0] + FF_r[7];          //add1
                                ilk_toplam_r[1] <= -FF_r[1] + FF_i[6];          //add2
                                ilk_toplam_r[2] <= FF_i[2] - FF_r[5];           //add3
                ilk_toplam_r[3] <= FF_r[3] - FF_i[4];           //add4
                //imags
                ilk_toplam_i[0] <= FF_r[0] + FF_i[7];           //add1
                ilk_toplam_i[1] <= -FF_i[1] - FF_r[6];          //add2
                ilk_toplam_i[2] <= -FF_r[2] - FF_i[5];          //add3
                ilk_toplam_i[3] <= FF_i[3] + FF_r[4];           //add4
            end
            6 : begin
                ilk_toplam_r[0] <= -FF_r[0] + FF_r[7];          //add1
                ilk_toplam_r[1] <= FF_r[1] - FF_r[6];           //add2
                ilk_toplam_r[2] <= -FF_r[2] + FF_r[5];          //add3
                ilk_toplam_r[3] <= FF_r[3] - FF_r[4];           //add4
                //imags
                ilk_toplam_i[0] <= FF_i[0] - FF_i[7];           //add1
                ilk_toplam_i[1] <= -FF_i[1] + FF_i[6];          //add2
                ilk_toplam_i[2] <= FF_i[2] - FF_i[5];           //add3
                ilk_toplam_i[3] <= -FF_i[3] + FF_i[4];          //add4
            end
            7 : begin
                ilk_toplam_r[0] <= FF_i[0] + FF_r[7];           //add1
                ilk_toplam_r[1] <= -FF_r[1] - FF_i[6];          //add2
                ilk_toplam_r[2] <= -FF_i[2] - FF_r[5];          //add3
                ilk_toplam_r[3] <= FF_r[3] + FF_i[4];           //add4
                //imags
                ilk_toplam_i[0] <= -FF_r[0] + FF_i[7];          //add1
                ilk_toplam_i[1] <= -FF_i[1] + FF_r[6];          //add2
                ilk_toplam_i[2] <= FF_r[2] - FF_i[5];           //add3
                ilk_toplam_i[3] <= FF_i[3] - FF_r[4];           //add4
            end
        endcase
    end
end
//carpicilar
always@(posedge clk1, posedge reset)begin
    if(reset == 1)begin
        for( i=0; i<=3; i=i+1 )begin
            carpim_r[i] <= 0;
            carpim_i[i] <= 0;
        end
    end
    else begin
        for( i=0; i<=3; i=i+1 )begin
            carpim_r[i] <= ilk_toplam_r[i] * h[i];
            carpim_i[i] <= ilk_toplam_i[i] * h[i];
        end
    end
end
//toplayici agaci seviye 1
always@(posedge clk1, posedge reset)begin
    if(reset == 1)begin
        for( i=0; i<=1; i=i+1 )begin
            agac_sev_1_r[i] <= 0;
            agac_sev_1_i[i] <= 0;
        end
    end
    else begin
        agac_sev_1_r[0] <= carpim_r[0] + carpim_r[1];
        agac_sev_1_r[1] <= carpim_r[2] + carpim_r[3];
        //
        agac_sev_1_i[0] <= carpim_i[0] + carpim_i[1];
        agac_sev_1_i[1] <= carpim_i[2] + carpim_i[3];
    end
end
//toplayici agaci seviye 2
always@(posedge clk1, posedge reset)begin
    if(reset == 1)begin
        agac_sev_2_r <= 0;
        agac_sev_2_i <= 0;
    end
    else begin
        agac_sev_2_r <= agac_sev_1_r[0] + agac_sev_1_r[1];
        agac_sev_2_i <= agac_sev_1_i[0] + agac_sev_1_i[1];
    end
end
//33bit - 16 bit donusturucu (cikis kesicisi)
```

```
always@(*)begin
        if(agac_sev_2_r[28] == 1)begin          //sayi negatif ise
                if( |agac_sev_2_r[12:0] == 1 )begin                 //sayi, tamsayisinden küçükse
                        kesik_r <= agac_sev_2_r[28:13] + 1;
                end
                else begin
                        kesik_r <= agac_sev_2_r[28:13];
                end
        end
        else begin
                kesik_r <= agac_sev_2_r[28:13];
        end
        //
        if(agac_sev_2_i[28] == 1)begin          //sayi negatif ise
                if( |agac_sev_2_i[12:0] == 1 )begin                 //sayi, tamsayisinden küçükse
                        kesik_i <= agac_sev_2_i[28:13] + 1;
                end
                else begin
                        kesik_i <= agac_sev_2_i[28:13];
                end
        end
        else begin
                kesik_i <= agac_sev_2_i[28:13];
        end
end
//enterpolasyon sonrasi cikis
always@(posedge clk1, posedge reset)begin
        if(reset == 1)begin
                ent_out_r <= 0;
                ent_out_i <= 0;
        end
        else begin
                case(hb3_mod)
                        2 : begin
                                ent_out_r <= kesik_i;
                                ent_out_i <= kesik_r;
                        end
                        6 : begin
                                ent_out_r <= kesik_i;
                                ent_out_i <= kesik_r;
                        end
                        default : begin
                                ent_out_r <= kesik_r;
                                ent_out_i <= kesik_i;
                        end
                endcase
        end
end
//cikis secicisi
always@(posedge clk2, posedge reset)begin
        if(reset == 1)begin
                dout_r <= 0;
                dout_i <= 0;
        end
        else begin
                if(bir_say == 0)begin
                        dout_r <= ent_out_r;
                        dout_i <= ent_out_i;
                end
                else begin
                        dout_r <= FF_r[8];
                        dout_i <= FF_i[8];
                end
        end
end
//bire kadar sayan sayici
always@(posedge clk2, posedge reset)begin
        if(reset == 1)begin
                bir_say <= 0;
        end
        else begin
                bir_say <= ~bir_say;
        end
end
endmodule
```

## V.4. Verilog Code of TestBench (85 dB Design)

```
`timescale 1ns / 1ps
//`include "/work/kits/lf/1.8.0/PDK_LF150i_V1_8_0/digital/verilog/LF150DI_HS_F_V1_5_typical_conditional.v"
module Test_Suzgec_Uclu;
		// Inputs
		reg clk250;
		reg clk500;
		reg clk1;
		reg clk2;
		reg reset;
		reg [1:0] hb1_mod;
		reg [2:0] hb2_mod;
		reg [2:0] hb3_mod;
		reg signed [15:0] din_r, din_i;
		// Outputs
		wire signed[15:0] dout_1_r;
		wire signed[15:0] dout_1_i;
		wire signed[15:0] dout_2_r;
		wire signed[15:0] dout_2_i;
		wire signed[15:0] dout_3_r;
		wire signed[15:0] dout_3_i;
		// Sampled Outputs
		reg signed[15:0] dout_1_s;
		reg signed[15:0] dout_2_s;
		reg signed[15:0] dout_3_s;
		// parameters
		parameter period = 8.000;		//4.000 : 250MHz freakansli clock giriyor
		// file
		integer file;
		// SDF
//		initial $sdf_annotate("Suzgec_Uclu_syn_typ.sdf",uut);
		// Instantiate the Unit Under Test (UUT)
		Suzgec_Uclu uut (
				.clk250(clk250),
				.clk500(clk500),
				.clk1(clk1),
				.clk2(clk2),
				.reset(reset),
				.hb1_mod(hb1_mod),
				.hb2_mod(hb2_mod),
				.hb3_mod(hb3_mod),
				.din_r(din_r),
				.din_i(din_i),
				.dout_1_r(dout_1_r),
				.dout_1_i(dout_1_i),
				.dout_2_r(dout_2_r),
				.dout_2_i(dout_2_i),
				.dout_3_r(dout_3_r),
				.dout_3_i(dout_3_i)
		);
		initial begin
				//open file (fire)
				file = $fopen("sim_sonuc.m","w");
				$fwrite(file,"x = NaN;\n");
				$fwrite(file,"X = NaN;\n");
				$fwrite(file,"dout_3=[\n");

				// Initialize Inputs
				clk250 = 1;
				clk500 = 1;
				clk1 = 1;
				clk2 = 1;
				reset = 0;
				hb1_mod = 3;
				hb2_mod = 1;
				hb3_mod = 1;
				din_r = 0;
				din_i = 0;
				//reset
				#period reset = 1;
				//reset
				#(9*period/16) reset = 0;
```

```
            #period   din_r     =         0         ;
            #period   din_r     =         7341      ;
            #period   din_r     =         14220     ;
            #period   din_r     =         20204     ;
            #period   din_r     =         24916     ;
            #period   din_r     =         28059     ;
            #period   din_r     =         29436     ;
            #period   din_r     =         28959     ;
            #period   din_r     =         26660     ;
            #period   din_r     =         22682     ;
            #period   din_r     =         17276     ;
            #period   din_r     =         10782     ;
            #period   din_r     =         3610      ;
            #period   din_r     =         -3790     ;
            #period   din_r     =         -10951    ;
            #period   din_r     =         -17422    ;
            …                   …                   …
            #period   din_r     =         -22566    ;
            #period   din_r     =         -26582    ;
            #period   din_r     =         -28925    ;
            #period   din_r     =         -29446    ;
            #period   din_r     =         -28114    ;
            #period   din_r     =         -25012    ;
            #period   din_r     =         -20335    ;
            #period   din_r     =         -14378    ;
            #period   din_r     =         -7516     ;
            #period   din_r     =         -181      ;
            #period   din_r     =         7166      ;
            #period   din_r     =         14061     ;
            #period   din_r     =         20072     ;
            #period   din_r     =         24818     ;
            #period   din_r     =         28003     ;
            #period   din_r     =         29424     ;
            #period   din_r     =         28993     ;
            #period   din_r     =         26737     ;
            #period   din_r     =         22797     ;
            #period   din_r     =         17422     ;
            #period   din_r     =         10951     ;
            #period   din_r     =         3790      ;
            #period   din_r     =         -3610     ;
            #period   din_r     =         -10782    ;
            #period   din_r     =         -17276    ;
            #period   din_r     =         -22682    ;
            #period   din_r     =         -26660    ;
            #period   din_r     =         -28959    ;
            #period   din_r     =         -29436    ;
            #period   din_r     =         -28059    ;
            #period   din_r     =         -24916    ;
            #period   din_r     =         -20204    ;
            #period   din_r     =         -14220    ;
            #period   din_r     =         -7341     ;

            #100      $fwrite(file,"]';");
                                $fclose(file);
                                $stop;
        end

        always begin        #(period/2)         clk250 = ~clk250;   end
        always begin        #(period/4)         clk500 = ~clk500;   end
        always begin        #(period/8)         clk1 = ~clk1;                 end
        always begin        #(period/16)        clk2 = ~clk2;                 end
        always @ (negedge clk500) begin
            #(period/8)
            dout_1_s <= dout_1_r;
        end
        always @ (negedge clk1) begin
            #(period/16)
            dout_2_s <= dout_2_r;
        end
        always @ (negedge clk2) begin
            #(period/32)
            dout_3_s <= dout_3_r;
            $fwrite(file,"%d %d\n",dout_3_r, dout_3_i);
        end
endmodule
```

## V.5. Output file generated by Verilog TestBench

```
x = NaN;
X = NaN;
dout_3=[
     x      x
     x      x
     x      x
     x      x
     x      x
     x      x
     x      x
     x      x
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
     0      0
…          …
  9628 -23351
 18568 -18568
 20701  -8509
 14705      0
  3791   1716
 -5095  -5095
 -6766 -16035
     0 -24816
 10604 -25625
 18441 -18441
 18164  -7427
  9865      0
 -1732   -566
 -9403  -9403
 -8686 -20719
     0 -28002
 10915 -26288
 17153 -17153
 14483  -5876
  4402      0
 -7149  -2814
-13119 -13119
-10058 -24098]';
```

## V.6. Verilog Code of Testbench (Complete Digital System with 99 dB Design)

```
`timescale 1ns / 1ps
module TEST_CHIP_TOP;
        // SPI Inputs
        reg sclk;
        reg por;
        reg fsync;
        reg sdin;
        //SPI inputs from TRIMs
        reg [0:255] dataR_from_TRIM;
        reg [0:255] A_out_2;
        // SPI Outputs
        wire sdout;
        //SPI outputs to TRIMs
        wire [0:255] A_in;
        wire [0:255] A_clk;
        // parallel data Inputs
        reg clkin;
        reg signed [15:0] din_r, din_i;
        // data to DAC
        wire clkout_r;
        wire clkout_i;
        wire [30:0] MSBout_r;
        wire [30:0] MSBout_i;
        wire [10:0] LSBout_r;
        wire [10:0] LSBout_i;
        //Test Registers
        reg [4:0]Test_MSBbin;
        reg [15:0]TESTout;
        //parallel data Sampled Outputs
        //reg signed[15:0] dout_3_s;
        // parameters
        parameter period = 8000;        //4.000 : 250MHz freakansli clock giriyor
        //starting signal
        reg start_clock;
        // integer & file
        integer i = 0, j = 0, k = 0;
        integer file_sim_input;
        integer file_sim_sonuc;
        integer file_LA_csv;
        // SPI_CB WORD
        reg [23:0]WORD;
        // SPI_CB WORD definition
        reg [(12*8)-1:0] comment = "INITIAL";
        // TESTBENCH state
        reg [(12*8)-1:0] comment_tb = "INITIAL";
        // SPI_CB parallel read data
        reg [7:0] data_read;
        // Instantiate the Unit Under Test 1
        CHIP_TOP uut (
        .sclk       (sclk),
        .por        (por),
        .fsync      (fsync),
        .sdin       (sdin),
        .dataR_from_TRIM   (dataR_from_TRIM),
        .A_out_2    (A_out_2),
        .clkin             (clkin),
        .din_r             (din_r),
        .din_i             (din_i),
        .sdout             (sdout),
        .A_in              (A_in),
        .A_clk             (A_clk),
        .clkout_r   (clkout_r),
        .clkout_i   (clkout_i),
        .MSBout_r          (MSBout_r),
        .MSBout_i(MSBout_i),
        .LSBout_r (LSBout_r),
        .LSBout_i (LSBout_i)
        );
        //SPI signals
        initial begin
                // Initialize Inputs
                sclk = 0;
```

```
por = 1;
fsync = 0;
sdin = 0;
dataR_from_TRIM = 0;
A_out_2 = 0;
clkin = 0;
din_r = 0;
din_i = 0;
start_clock = 0;
WORD = 0;
data_read = 0;
//open file (fire)
file_LA_csv = $fopen("test_input.csv","w");
//header
$fwrite(file_LA_csv,"\"My Bus 1\"\n");
$fwrite(file_LA_csv,"\"Pod 6[7:0]\"\n");
$fwrite(file_LA_csv,"*Init Start\n");
$fwrite(file_LA_csv,"*Init End\n");
$fwrite(file_LA_csv,"*Main Start\n");
$fwrite(file_LA_csv,"FF\n");
// Wait 100 ns for global reset to finish
#100;
por = 0;
#100;
por = 1;
#100;
fsync = 1;
#100;
fsync = 0;
#100;
//TRIM testi
comment_tb = "TRIM TEST";
//yazilacak CMD
SPI_CB_TASK_24("write", 11'h001, 24'hACABA0);
//okunacak CMD
SPI_CB_TASK_24("read", 11'h001, 24'h000000);
//Fuse blow testi
comment_tb = "FuseBlow TEST";
//yazilacak CMD part 1
SPI_CB_TASK_24("write", 11'h002, 24'h800000);
//Fuse blow command
SPI_CB_TASK("fuseblow", 11'h003, 8'h00);
//yazilacak CMD
SPI_CB_TASK("write", 11'h003, 8'h00);
//okunacak CMD
SPI_CB_TASK_24("read", 11'h003, 24'h000000);

// RAMDAC yazma testi
comment_tb = "RAMDAC TEST";
//enabling RAMDAC
//yazilacak CMD
SPI_CB_TASK_16("write", 11'd301, 16'h1F00);
//writing input 1
//yazilacak CMD
SPI_CB_TASK_16("write", 11'd374, 16'h0001);
//yazilacak CMD
SPI_CB_TASK_16("write", 11'd375, 16'hA000);
# 10 clkin = 0;
# 10 clkin = 1;
# 10 clkin = 0;
//writing input 2
//yazilacak CMD
SPI_CB_TASK_16("write", 11'd374, 16'h0002);
//yazilacak CMD
SPI_CB_TASK_16("write", 11'd375, 16'hB000);
# 10 clkin = 0;
# 10 clkin = 1;
# 10 clkin = 0;
//writing input 3
//yazilacak CMD
SPI_CB_TASK_16("write", 11'd374, 16'h0003);
//yazilacak CMD
SPI_CB_TASK_16("write", 11'd375, 16'hC000);
# 10 clkin = 0;
# 10 clkin = 1;
# 10 clkin = 0;
//writing input 4
//yazilacak CMD
SPI_CB_TASK_16("write", 11'd374, 16'h0004);
//yazilacak CMD
SPI_CB_TASK_16("write", 11'd375, 16'hD000);
# 10 clkin = 0;
# 10 clkin = 1;
# 10 clkin = 0;
//disabling writing RAMDAC
//yazilacak CMD
SPI_CB_TASK_16("write", 11'd301, 16'h0F00);
//sampling RAMDAC
//yazilacak CMD
SPI_CB_TASK("write", 11'd376, 8'h00);
//yazilacak CMD
SPI_CB_TASK("write", 11'd377, 8'h00);
//reading RAMDAC data 1
//okunacak CMD
SPI_CB_TASK_16("read", 11'd376, 16'h0000);
//okunacak CMD
SPI_CB_TASK_16("read", 11'd377, 16'h0000);
# 10 clkin = 0;
# 10 clkin = 1;
# 10 clkin = 0;
//sampling RAMDAC
//yazilacak CMD
SPI_CB_TASK("write", 11'd376, 8'h00);
//yazilacak CMD
SPI_CB_TASK("write", 11'd377, 8'h00);
//reading RAMDAC data 2
//okunacak CMD
SPI_CB_TASK_16("read", 11'd376, 16'h0000);
//okunacak CMD
SPI_CB_TASK_16("read", 11'd377, 16'h0000);
# 10 clkin = 0;
# 10 clkin = 1;
# 10 clkin = 0;
//sampling RAMDAC
//yazilacak CMD
SPI_CB_TASK("write", 11'd376, 8'h00);
//yazilacak CMD
SPI_CB_TASK("write", 11'd377, 8'h00);
//reading RAMDAC data 3
//okunacak CMD
SPI_CB_TASK_16("read", 11'd376, 16'h0000);
//okunacak CMD
SPI_CB_TASK_16("read", 11'd377, 16'h0000);
# 10 clkin = 0;
# 10 clkin = 1;
# 10 clkin = 0;
//sampling RAMDAC
//yazilacak CMD
SPI_CB_TASK("write", 11'd376, 8'h00);
//yazilacak CMD
SPI_CB_TASK("write", 11'd377, 8'h00);
//reading RAMDAC data 4
//okunacak CMD
SPI_CB_TASK_16("read", 11'd376, 16'h0000);
//okunacak CMD
SPI_CB_TASK_16("read", 11'd377, 16'h0000);
# 10 clkin = 0;
# 10 clkin = 1;
# 10 clkin = 0;
//bekliyoruz
#1000
//open file (fire)
file_sim_input = $fopen("sim_input.txt","r");
file_sim_sonuc = $fopen("sim_sonuc.m","w");
$fwrite(file_sim_sonuc,"x = NaN;\n");
$fwrite(file_sim_sonuc,"X = NaN;\n");
$fwrite(file_sim_sonuc,"TESTout=[\n");
#100;
//por before filter operation (for clock equalization)
#100;
```

```
		por = 0;
		#100;
		por = 1;
		#100;
		//end of por
		// mode yazma testi
		comment_tb = "Mode TEST";
		SPI_CB_TASK_16("write", 11'd301, 16'h0300);			//0300: 8x 0000: no int.
		//Suzgec Testi
		comment_tb = "Filter TEST";
		start_clock = 1;
		for(j=0;j<256;j=j+1)begin

			i = $fscanf(file_sim_input,"%d\n",din_r);

			#(period);				// no.int: period/8			2x: period/4
	4x: period/2		8x: period

		end
		#100	$fwrite(file_sim_sonuc,"]';");
				$fclose(file_sim_input);
				$fclose(file_sim_sonuc);
				$stop;
		//reading Filter outputs
		comment_tb = "FilterOut TEST";
		//sampling Filter 1 outputs
		//yazilacak CMD
		SPI_CB_TASK("write", 11'd378, 8'h00);
		//yazilacak CMD
		SPI_CB_TASK("write", 11'd379, 8'h00);
		//reading Filter 1 outputs
		//okunacak CMD
		SPI_CB_TASK_16("read", 11'd378, 16'h0000);
		//okunacak CMD
		SPI_CB_TASK_16("read", 11'd379, 16'h0000);
		//sampling Filter 2 outputs
		//yazilacak CMD
		SPI_CB_TASK("write", 11'd380, 8'h00);
		//yazilacak CMD
		SPI_CB_TASK("write", 11'd381, 8'h00);
		//reading Filter 2 outputs
		//okunacak CMD
		SPI_CB_TASK_16("read", 11'd380, 16'h0000);
		//okunacak CMD
		SPI_CB_TASK_16("read", 11'd381, 16'h0000);
		//sampling Filter 3 outputs
		//yazilacak CMD
		SPI_CB_TASK("write", 11'd382, 8'h00);
		//yazilacak CMD
		SPI_CB_TASK("write", 11'd383, 8'h00);
		//reading Filter 3 outputs
		//okunacak CMD
		SPI_CB_TASK_16("read", 11'd382, 16'h0000);
		//okunacak CMD
		SPI_CB_TASK_16("read", 11'd383, 16'h0000);
		$fwrite(file_LA_csv,"*Main End\n");
		$fclose(file_LA_csv);
		$stop;
end
initial begin
		@(posedge start_clock)
			repeat(512*16)begin
				#(period/16)		clkin = ~clkin;
			end
end
initial begin
		Test_MSBbin = 0;
		TESTout = 0;
end
//Ther2bin converter
always @ (negedge clkin) begin
		#(period/8)
		Test_MSBbin = 0;
		for(k=0;k<=30;k=k+1)begin
			Test_MSBbin = Test_MSBbin + MSBout_r[k];
```

```
				end
				TESTout = {Test_MSBbin,  LSBout_r};
				//ekledik
				$fwrite(file_sim_sonuc,"%d\n",TESTout);
				//ekledik
			end
			task SPI_CB_TASK_16;
				input [(12*8)-1:0] command_string;
				input [10:0] addr;
				input [15:0] data;
				begin
					//MSB part
					SPI_CB_TASK(command_string, addr, data[15:8]);
					//LSB part
					SPI_CB_TASK(command_string, addr, data[7:0]);
				end
			endtask
			task SPI_CB_TASK_24;
				input [(12*8)-1:0] command_string;
				input [10:0] addr;
				input [23:0] data;
				begin
					//MSB part
					SPI_CB_TASK(command_string, addr, data[23:16]);
					//MID part
					SPI_CB_TASK(command_string, addr, data[15:8]);
					//LSB part
					SPI_CB_TASK(command_string, addr, data[7:0]);
				end
			endtask
			task SPI_CB_TASK;
				input [(12*8)-1:0] command_string;
				input [10:0] addr;
				input [7:0] data;
				begin
					case(command_string)
						"write" : begin
							comment = "write";
							WORD[23:19] = 5'b00100;
						end
						"read" : begin
							comment = "read";
							WORD[23:19] = 5'b00101;
						end
						"fuseblow" : begin
							comment = "fuseblow";
							WORD[23:19] = 5'b00110;
						end
						"reserved" : begin
							comment = "reserved";
							WORD[23:19] = 5'b00000;
						end
					endcase
					WORD[18:8] = addr;
					WORD[7:0] = data;
					for (i=0; i<24; i = i+1)begin
						sdin = WORD[23-i];
						#100;
						sclk = 1;
						#100;
						sclk = 0;
						if( (i >= 16) && (command_string == "read") )		data_read[23-i] = sdout;

					end
					#100;
					fsync = 1;
					#100;
					fsync = 0;
					#100;
					comment = "done";
				end
			endtask
		endmodule
```

## V.7. Verilog Code of Clock Divider

```
`timescale 1ns/1ps
module ClkDiv4Suzgec(   clk_in,
                        por,
                        mod_mode,
                        int_mode,
                        clk4,
                        clk3,
                        clk2,
                        clk1,
                        clk0
                        );
        input clk_in;
        input por;
        input mod_mode;
        input [1:0]int_mode;
        output clk4;
        output reg clk3;
        output reg clk2;
        output reg clk1;
        output reg clk0;

        reg pre_clk3;
        reg pre_clk2;
        reg pre_clk1;
        reg pre_clk0;
        reg FF3;
        reg FF21;
        reg FF22;
        reg FF11;
        reg FF12;
        reg FF13;
        reg FF14;
        reg FF01;
        reg FF02;
        reg FF03;
        reg FF04;
        reg FF05;
        reg FF06;
        reg FF07;
        reg FF08;

        assign clk4 = clk_in;

        //output muxes without reg
        always@(*)begin
                if(mod_mode == 0)begin                  //normal mode
                        clk3 = clk_in;
                end
                else begin                                              //PM or RT mode
                        clk3 = pre_clk3;
                end
        end
        //output muxes with reg (mod_mode)
        always@(posedge clk_in or negedge por)begin
                if(por == 0)begin
                        pre_clk3 <= 0;
                        pre_clk2 <= 0;
                        pre_clk1 <= 0;
                        pre_clk0 <= 0;
                end
                else begin
                        pre_clk3 <= FF3;
                        if(mod_mode == 0)begin                  //normal mode
                                pre_clk2 <= FF3;
                                pre_clk1 <= FF22;
                                pre_clk0 <= FF14;
                        end
                        else begin                                              //PM or RT mode
                                pre_clk2 <= ~FF21;
                                pre_clk1 <= ~FF13;
                                pre_clk0 <= ~FF07;
                        end
```

```
		end
end
//output muxes with reg (int_mode)
always@(posedge clk_in or negedge por)begin
		if(por == 0)begin
				clk2 <= 0;
				clk1 <= 0;
				clk0 <= 0;
		end
		else begin
				case(int_mode)
						0 : begin			//no int
								clk2 <= 0;
								clk1 <= 0;
								clk0 <= 0;
						end
						1 : begin			//2x int
								clk2 <= pre_clk2;
								clk1 <= 0;
								clk0 <= 0;
						end
						2 : begin			//2x int
								clk2 <= pre_clk2;
								clk1 <= pre_clk1;
								clk0 <= 0;
						end
						3 : begin			//2x int
								clk2 <= pre_clk2;
								clk1 <= pre_clk1;
								clk0 <= pre_clk0;
						end
				endcase
		end
end
// clk/2 generation
always@(posedge clk_in or negedge por)begin
		if(por == 0)begin
				FF3 <= 0;
		end
		else begin
				FF3 <= ~FF3;
		end
end
// clk/4 generation
always@(posedge clk_in or negedge por)begin
		if(por == 0)begin
				FF21 <= 0;
				FF22 <= 0;
		end
		else begin
				FF21 <= FF22;
				FF22 <= ~FF21;
		end
end
// clk/8 generation
always@(posedge clk_in or negedge por)begin
		if(por == 0)begin
				FF11 <= 0;
				FF12 <= 0;
				FF13 <= 0;
				FF14 <= 0;
		end
		else begin
				FF11 <= FF14;
				FF12 <= FF11;
				FF13 <= FF12;
				FF14 <= ~FF13;
		end
end
// clk/16 generation
always@(posedge clk_in or negedge por)begin
		if(por == 0)begin
				FF01 <= 0;
				FF02 <= 0;
				FF03 <= 0;
				FF04 <= 0;
				FF05 <= 0;
				FF06 <= 0;
				FF07 <= 0;
				FF08 <= 0;
			end
			else begin
				FF01 <= FF08;
				FF02 <= FF01;
				FF03 <= FF02;
				FF04 <= FF03;
				FF05 <= FF04;
				FF06 <= FF05;
				FF07 <= FF06;
				FF08 <= ~FF07;
			end
		end


	endmodule
```

## V.8. Verilog Code of SPI_CB

```
`timescale 1ns / 1ps
module SPI_CB(
                //inputs
                sclk,
                por,
                fsync,
                sdin,
                dataR,

                //outputs
                sdout,
                resetRegOut,
                sdoutEn,
                write,
                dataW,
                CE,
                BlowFuse);

parameter L_ARRAY = 512;    // 512Adres * 16 bit = 1024 adres * 8 bit
parameter L_ADDR = 9;                   //equation must be satisfied: 2**L_ADDR = L_ARRAY
parameter L_DATA = 8;                   //data length comes after command & address
integer i;

input   sclk;               //saat işareti girişi.
input   por;                //power on reset.
input   fsync;              //fsync kontrol işareti girişi.
input   sdin;               //seri data girişi (user'dan)
input [0:L_ARRAY-1] dataR;              //adrese ozel okunan data (cip icinden)
output sdout;               //seri data çıkışı (user'a)
output resetRegOut;
output reg sdoutEn;
output reg write;                       //ortak yazma istegi
output dataW;                       //ortak seri data cikisi (cip icine)
output reg [0:L_ARRAY-1] CE;            //adrese ozel enable
output reg BlowFuse;

wire    reset;          //reset işareti
reg [5:0]   counter;            //counter.
reg [4:0]   Command;
reg [L_ADDR-1:0]    Addr;
reg resetReg;           //Reset Registers commandi geldiginde iner (0 enable)
reg pre_CE;

assign reset = por & (~fsync);
assign dataW = sdin;
assign sdout = dataR[Addr];
assign resetRegOut = por & resetReg;

//command'e gore yapilacak islem
always @ (posedge sclk or negedge por) begin
        if(por == 0) begin
                resetReg <= 1;
                sdoutEn <= 1;
                BlowFuse <= 0;
                write <= 0;
        end
        else begin
                if(counter == 15)begin
                        //if(Command == 5'b00000)
                                                        //Noop
                        if(Command == 5'b00001)     resetReg <= 0;      else resetReg <= 1;
                        //Reset DAC registers
                        if(Command == 5'b00010)     sdoutEn <= 1;
                                                        //Enable sdout
                        if(Command == 5'b00011)     sdoutEn <= 0;
                                                                //Hi-Z sdout
                        //if(Command == 5'b00111)
                                                        //Reserved for now
                end
                if(Command == 5'b00100)     write <= 1;     else write <= 0;
                            //write data from serial input or parallel fuse data to registers
```

```
			if(Command == 5'b00110)		BlowFuse <= 1;		else BlowFuse <= 0;
					//Blow Fuse
		end
end

//fsync 0 iken 24 clk boyunca sdin'den data alinip SR'a yazilir.
always @ (posedge sclk or negedge por) begin
		if(por == 0) begin
			Command <= 0;
			Addr <= 0;
		end
		else begin
			if (counter < 5) begin							//ilk 5 clk command
				Command <= {Command[3:0], sdin};
			end
			else if(counter < 16)begin							//sonraki 11 clk adres
				Addr <= {Addr[L_ADDR-2:0], sdin};
			end
		end
end

//
always @ (posedge sclk or negedge reset) begin
		if(reset == 0) begin
			pre_CE <= 0;
		end
		else begin
			if(counter == 15)begin
		//command'in son clk'unda
				if( (Command == 5'b00100) || (Command == 5'b00101) || (Command == 5'b00110) )begin
			// yaz, oku ya da yak demisse
					pre_CE <= 1;
				end
				else begin
					pre_CE <= 0;
				end
			end
			else if(counter == (15 + L_DATA))begin
		//son clk'ta				//data bit sayisi artsa bunu da arttiricaz
				pre_CE <= 0;
			end
		end
end

//CE demux
always@(*)begin
		for(i=0; i<L_ARRAY; i=i+1)begin
			if(i == Addr)begin
				CE[i] = pre_CE;			// data bekleniyor
			end
			else begin
				CE[i] = 0;
			end
		end
end

//fsync 0 iken her clk'ta counter 1 arttirilir
always @ (posedge sclk or negedge reset) begin			//reset olmasinin sebebi fsync 1 oldugunda clk olmadan
sifirlamak istememiz
		if(reset == 0) begin
			counter <= 0;
		end
		else begin
			if(counter < (16 + L_DATA))begin				//data bit sayisi artarsa bunu da arttiricaz
				counter <= counter + 1;
			end
		end
end

endmodule
```

## V.9. Verilog Code of RAMDAC

```
`timescale 1ns / 1ps
module RAMDAC_BANK(                                     sclk,
                                                                SRclk_r,
                                                                resetRegOut,
                                                                dataW,
                                                                write,
                                                                RAMDAC_w_enable,
                                                                CE_374,
                                                                CE_376,
                                                                dataR_374,
                                                                dataR_376,
                                                                dout_r,                         //data to
                                                                clk_out
                                                                );

input sclk;
input SRclk_r;
input resetRegOut;
input dataW;
input write;
input RAMDAC_w_enable;
input CE_374;
input CE_376;
output dataR_374;
output dataR_376;
output [15:0]dout_r;
output clk_out;

wire [15:0] din_r, din_i;
assign clk_out = SRclk_r;

FILTER_REG reg374(
.sclk                   (sclk),
.por                    (resetRegOut),
.dataW      (dataW),
.write      (write),
.CE                     (CE_374),
.dataR      (dataR_374),
.pardout    (din_r)
);
RAMDAC_SR ramdac_sr(
.clk_r                  (SRclk_r),
.resetRegOut            (resetRegOut),
.din_r                  (din_r),
.RAMDAC_w_enable                    (RAMDAC_w_enable),
.dout_r                 (dout_r)
);
FILTER_END reg376(
.sclk                   (sclk),
.por                    (resetRegOut),
.doutF      (dout_r),
.write      (write),
.CE                     (CE_376),
.dataR      (dataR_376)
);
endmodule

`timescale 1ns / 1ps

module RAMDAC_SR(
                                                        clk_r,
                                                        resetRegOut,
                                                        din_r,
                                                        RAMDAC_w_enable,
                                                        dout_r
);

input clk_r;
input resetRegOut;
input [15:0]din_r;
input RAMDAC_w_enable;
output [15:0] dout_r;
```

```
integer i;
parameter L_RAMDAC = 128;                                   //default can be 4 or 128
reg [15:0]RAMDAC_r[0:L_RAMDAC-1];
assign dout_r = RAMDAC_r[L_RAMDAC-1];

//real part
always@(posedge clk_r, negedge resetRegOut)
begin
        if(resetRegOut == 0)begin
                for( i=0; i<=L_RAMDAC-1; i=i+1 )begin
                        RAMDAC_r[i] <= 0;
                end
        end
        else begin
                if(RAMDAC_w_enable == 1)begin
                        RAMDAC_r[0] <= din_r;
                end
                else begin
                        RAMDAC_r[0] <= RAMDAC_r[L_RAMDAC-1];
                end
                for( i=1; i<=L_RAMDAC-1; i=i+1 )begin
                        RAMDAC_r[i] <= RAMDAC_r[i-1];
                end
        end
end
endmodule
```

## V.10. Verilog Code of Binary to Thermometer Encode with Modulation

```
`timescale 1ns / 1ps
module Bin2TherWmod (        clk_RAMDAC,              // RAMDAC
                             clk_suzgec,              // suzgec
                             clk4,
                             por,
                             mode,
                             RAMDAC_enable,
                             data_from_RAMDAC,
                             data_from_suzgec,
                             MSBout,
                             LSBout,
                             clk_out
                             );
        input clk_RAMDAC;
        input clk_suzgec;
        input clk4;
        input por;
        input [1:0] mode;    //00: normal       01: normal10: RZ          11: mod
        input RAMDAC_enable;
        input [15:0] data_from_RAMDAC;
        input [15:0] data_from_suzgec;
        output [30:0] MSBout;
        output [10:0] LSBout;
        output clk_out;

        wire clk3;
        reg [15:0] int_data_RAMDAC;
        reg [15:0] int_data_suzgec;
        reg [4:0] int_MSBin;
        reg [30:0] Ther;
        reg [30:0] PMTher;
        reg [30:0] RZTher;
        reg [30:0] mux_1_out;
        reg [30:0] mux_2_out;
        reg [10:0] int_LSBin;
        reg [10:0] LSB_sev_1;
        reg [10:0] PMLSB;
        reg [10:0] RZLSB;
        reg [10:0] mux_1_LSBout;
        reg [10:0] mux_2_LSBout;
        reg bir_say;
        reg bir_say_2;
        reg bir_say_3;

        integer i;

        assign clk_out = clk4;
        assign MSBout = mux_2_out;
        assign LSBout = mux_2_LSBout;
        assign clk3 = RAMDAC_enable ? clk_RAMDAC : clk_suzgec;

        //giris orneklenemsi
        always@(posedge clk3 or negedge por)begin
                if(por == 0)begin
                        int_data_RAMDAC <= 0;
                        int_data_suzgec <= 0;
                end
                else begin
                        int_data_RAMDAC <= data_from_RAMDAC;
                        int_data_suzgec <= data_from_suzgec;
                end
        end
        //input muxes
        always@(posedge clk3 or negedge por)begin
                if(por == 0)begin
                        int_MSBin <= 0;
                        int_LSBin <= 0;
                end
                else begin
                        if(RAMDAC_enable == 1)begin
        //RAMDAC_r_enable
                                int_MSBin <= {~int_data_RAMDAC[15], int_data_RAMDAC[14:11]};
```

```
				int_LSBin <= int_data_RAMDAC[10:0];
			end
			else begin
				int_MSBin <= {~int_data_suzgec[15], int_data_suzgec[14:11]};
				int_LSBin <= int_data_suzgec[10:0];
			end
		end
end
//Thermometer converter MSB
always@(posedge clk3 or negedge por)begin
		if(por == 0)begin
			Ther <= 0;
		end
		else begin
			for(i=0;i<=30;i=i+1)begin
				if(i < int_MSBin)begin
					Ther[i] <= 1;
				end
				else begin
					Ther[i] <= 0;
				end
			end
		end
end
//plus minus thermometer MSB
always@(posedge clk4 or negedge por)begin
		if(por == 0)begin
			PMTher <= 0;
		end
		else begin
			if(bir_say == 0)begin
				PMTher <= Ther;
			end
			else begin
				PMTher <= ~Ther;
			end
		end
end
//RTZ MSB
always@(posedge clk4 or negedge por)begin
		if(por == 0)begin
			RZTher <= 0;
		end
		else begin
			if(bir_say_2 == 0)begin
				RZTher <= Ther;
			end
			else begin
				RZTher <= {16'hFFFF, 15'h0000};
			end
		end
end
//Mux 1 MSB
always@(posedge clk4, negedge por)begin
		if(por == 0)begin
			mux_1_out <= 0;
		end
		else begin
			if(mode[0] == 0)begin
				mux_1_out <= RZTher;		//zero mode
			end
			else begin
				mux_1_out <= PMTher;		//mod mode
			end
		end
end
//Mux 2 MSB
always@(posedge clk4, negedge por)begin
		if(por == 0)begin
			mux_2_out <= 0;
		end
		else begin
			if(mode[1] == 0)begin			//normal mode
				mux_2_out <= Ther;
			end
```

```
				else begin						//zero or mod mode
					mux_2_out <= mux_1_out;
				end
		end
end
//Delay LSB 1
always@(posedge clk3 or negedge por)begin
		if(por == 0)begin
				LSB_sev_1 <= 0;
		end
		else begin
				LSB_sev_1 <= int_LSBin;
		end
end
//plus minus LSB
always@(posedge clk4 or negedge por)begin
		if(por == 0)begin
				PMLSB <= 0;
		end
		else begin
				if(bir_say_3 == 0)begin
					PMLSB <= LSB_sev_1;
				end
				else begin
					PMLSB <= ~LSB_sev_1;
				end
		end
end
//RTZ LSB
always@(posedge clk4 or negedge por)begin
		if(por == 0)begin
				RZLSB <= 0;
		end
		else begin
				if(bir_say_3 == 0)begin
					RZLSB <= LSB_sev_1;
				end
				else begin
					RZLSB <= 0;
				end
		end
end
//Mux 1 LSB
always@(posedge clk4, negedge por)begin
		if(por == 0)begin
				mux_1_LSBout <= 0;
		end
		else begin
				if(mode[0] == 0)begin
					mux_1_LSBout <= RZLSB;		//zero mode
				end
				else begin
					mux_1_LSBout <= PMLSB;		//mod mode
				end
		end
end
//Mux 2 LSB
always@(posedge clk4, negedge por)begin
		if(por == 0)begin
				mux_2_LSBout <= 0;
		end
		else begin
				if(mode[1] == 0)begin				//normal mode
					mux_2_LSBout <= LSB_sev_1;
				end
				else begin						//zero or mod mode
					mux_2_LSBout <= mux_1_LSBout;
				end
		end
end
//bire kadar sayan sayici
always@(posedge clk4, negedge por)begin
		if(por == 0)begin
				bir_say <= 0;
		end
```

```
				else begin
					bir_say <= ~bir_say;
				end
		end
		//bire kadar sayan sayici 2
		always@(posedge clk4, negedge por)begin
				if(por == 0)begin
					bir_say_2 <= 0;
				end
				else begin
					bir_say_2 <= ~bir_say_2;
				end
		end
		//bire kadar sayan sayici 3
		always@(posedge clk4, negedge por)begin
				if(por == 0)begin
					bir_say_3 <= 0;
				end
				else begin
					bir_say_3 <= ~bir_say_3;
				end
		end
endmodule
```

## CURRICULUM VITAE

**Name Surname:** Gürer Özbek

**Place and Date of Birth:** Istanbul, 19.09.1987

**Address:** Kültür district, Sevgi street, A5/5, Ulus, Istanbul

**E-Mail:** ozbekgu@itu.edu.tr

**B.Sc. 1:** FPGA Controlled Logistics Robot (ITU, 2009)

**B.Sc. 2:** Direction of Arrival Estimation of Acoustic Signals (ITU, 2010)

**Professional Experience and Rewards:**

- **R.A. and T.A. –** ITU Faculty of Electrical and Electronics Eng. (11/2010-…)
- **Junior Designer** – ITU VLSI LABs. (06/2009-09/2010)
- **Intern –** Aselsan REHİS System Engineering (08/2008-09/2008)
- **Intern –** ST Microelectronics Istanbul Office (06/2008 – 08/2008)
- **Intern –** ITU DSP LAB (06/2007-07/2007)
- **1st Prize Award** from ITU Faculty of Electrical and Electronics Eng.

**List of Publications:**

- Aksin D., **Ozbek G.** and Maloberti F.: Multi-Rate Segmented Time-Interleaved Current Steering DAC with Unity-Elements Sharing. *IEEE International Symposium on Circuits and Systems, ISCAS 2010*, Paris, May 30-June 2 2010, pp. 3353-3356.

**PUBLICATIONS/PRESENTATIONS ON THE THESIS**

- **Ozbek G.**, Karaali Ö.K. and Küyel T.: Verification of an Interpolation and Modulation System Used at Digital to Analog Converters. *Symposium on Embedded Systems and Applications, GÖMSİS 2012,* November 29-30, 2012 Istanbul, Turkey