Msc Project

Msc Electronics

Rahul Singh Thakur

Matriculation number: s1776200

Project Report

**An Event-Based Asynchronous Processing System**

August 2018

# MSc project Mission Statement

## Event-Based Asynchronous Processing System


**Student:** **Rahul Singh Thakur**

**Supervisor: Dr. Alister Hamilton**

### Background and Project Aim:

Synchronous data converters use uniform sampling for data conversion and signal processing which uses a lot of power. Event based approach is asynchronous that samples the data non-uniformly thus saving dynamic power for ADC and DSP [1]. This can benefit the applications that requires low power source or even can be used for energy harvesting. An example is biomedical devices or wearable gadgets.

The goal of this project can be characterized into 3 parts. First, conversion of an analogue signal coming from a source (music signal) to an event signals. Second, process the event signals using asynchronous event processor (low pass filters). Third, converting the processed signal back to analogue signal as output (speakers). If the time is allowed in the end, this whole circuit could be designed and developed on a PCB.

### Preparatory Task:

- Research background knowledge provided by project supervisor as well as from other resources.
- Understanding the principle and working of widely used analogue to digital and digital to analogue converters.
- To understand the challenges in the process to convert analogue to event signals and find possible solutions.
- To understand and research about asynchronous processors and how it can be implemented on FPGA.

### Main Task:

- Implementing analogue to events converter.
- Find a possible solution to develop asynchronous event processor.

- To figure out how to design analogue delay line used for implementing a low pass filter.
- Implementing event-to-analogue signals on simulations or on PCB
- Integrate the above modules and check for final test.
- If possible design the system on a PCB.

**Background Knowledge:**

- Cadence (Simulations)
- FPGA architecture (if event processor is implemented using FPGA)
- MATLAB (for designing low-pass FIR filter)

**Reminder:**

Asynchronous circuits as well as analogue-to-event and event-to-analogue conversion techniques should be studied, and relevant information should be gathered to implement the above challenge.

**References Material:**

[1]. Luiz Carlos Gouveia, T. J. K. a. A. H., 2011. An Asynchronous Spike Event Coding Scheme for. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS,* Volume 58, pp. 791-799.

[2]. Rybakov, A. S., 2016. The Use of Asynchronous Sigma-Delta Modulation. *Avtomatika i Vychislitel'naya Tekhnika,* Volume 50, pp. 385-396.

[3]. Tsividis, Y., 2010. Event-Driven Data Acquisition and. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS,* Volume 57, pp. 577-581.

The supervisor and student are satisfied that this project is suitable for performance and assessment in accordance with the guidelines of the course documentation.

Signed:

Student: _____

Supervisor: _____

Date: _____

# Abstract

Power is one of the major concerns for designers and there are continuous efforts being made to reduce power consumption. Due to the absence of clock in the event driven system, power consumption can be reduced. These techniques can be used for wearable, implantable or digestible biomedical devices that are powered by a tiny battery that cannot be replaced or charged often.

This project describes an event-driven data acquisition along with digital signal processing that can potentially offer significant energy and bandwidth savings.

The methodology described in this report does not require a clock for sampling time, it is implemented using level-crossing sampling in amplitude of the analogue signal which generates non-uniform sampling points. The asynchronous method used here generates non-uniform samples, thus eliminating the use of the clock. An FIR filter is designed through an asynchronous processor. A delay that is required for the functioning of the FIR filter is designed and implemented using analogue components.

# Declaration of Originality

I declare that this thesis is my original work except where stated. This thesis has never been submitted for any degree or examination to any other University.

……

# Statement of Achievement

This project has helped me in gaining insight about asynchronous processors and their advantages over widely used synchronous processors. Asynchronous processor was successfully implemented on an field programmable gate array (FPGA) which functions as a low pass finite impulse response filter (FIR). Power dissipation is one of the concerns for every designer, and in this project, I was able to reduce the power consumption of the overall system, which at the end was **0.504W.** This power can be further reduced and is discussed in project further work.

This project also made me aware of the digital side of cadence virtuoso tool, where I designed the analogue components using Verilog-A thus, increasing my knowledge. I have never used so many components (analogue components) before on a breadboard but, in this project, I used more than 400 components on the breadboard and successfully designed an analogue delay line dealing with all the stray losses.

Along with the technical achievements, I also learned time managements skills which helped me to complete the project in given time.

# Contents

# List of Figures

# List of tables

# List of code snippets

# Glossary

S.No. : Serial Number
ADC : Analogue to digital converter
DAC : Digital to analogue converter
DSP : Digital signal processing
AEC : Analogue to event converter
EAC : Event to analogue converter
DEC : Digital to event converter
EDC : Event to digital converter
ADM : Asynchronous delta modulator
CT : Continuous time
DT : Discrete time
CMOS : Complementary metal oxide semiconductor
FPGA : Field programmable gate array
IIR : Infinite impulse response
FIR : Finite impulse response
LPF : Low pass filter
HPF : High pass filter
BPF : Band pass filter
BSF : Band stop filter
ASF : All pass filter
EDA : electronic design automation
RC : Resistor-capacitor
PCB : Printed circuit board
HDL : Hardware descriptive language
SoC : System on chip
BBD : Bucket bridge device
Op-amp : Operational amplifier
IO : Input-output
CAB : Configurable analogue blocks
Event : A sample/spike which is produced when something significant occurs [2].

# List of Symbols

∑ : Summation
C̄ : Tau (delay)
∫ : Integrator
$\theta$ : Theta (time constant)
$\pi$ : Pi
$\omega$ : Omega
$\omega$c : Cut off frequency
t : Time
$f_s$ : Sampling frequency
δt : Error of time quantisation
V : Voltage
P : Power
$T_c$ : Timer
R : Resistor
I : Current
$\mu$ : micro ($10^{-6}$)
s : second
k : Kilo
f : frequency
C : Capacitance
δ1 : Passband ripple
δ2 : Stopband ripple
$\omega p$ : Passband edge frequency
$\omega s$ : Stopband edge frequency
h(n) : Filter coefficients
w(n) : Window coefficients
$h_D(n)$ : Impulse response
* : Multiplier (Verilog)
+ : Addition (Verilog)
Ω : Ohm
F : Farad
p : pico ($10^{-12}$)
n : nano ($10^{-9}$)
Hz : Hertz
Q : Charge

# Chapter 1

# Introduction

## 1.1 Motivation

We live in a world surrounded by analogue signals. These analogue signals carry information within them that needs to be processed. Processors are required to process these signals, such that the processor can carry out the manipulation of the data accordingly. For instance, while speaking on the phone, the analogue voice signal is converted to a digital signal for transmission and at the receiver it is converted back to an analogue signal and propagated through speakers or headphones. Another example would be the generation of an analogue signal by the singer at a concert. This analogue signal is then converted to a digital domain for processing the sound quality and converting it back to analogue signal through speakers for the audience. Conversion of the analogue signals to the digital domain is necessary because the analogue signal is susceptible to various forms of noise. Digital signals on the other hand are not affected as much because they are represented in binary form. Another advantage of using digital signals over analogue signals is the data compression capabilities. The bandwidth of the channel is also saved as more information can be transferred at a higher rate. Digitals signals can also be coded for secure communication.

A majority of digital signal processors sample at a constant sampling rate dictated by the Nyquist theorem [4], which states that the sampling rate must be at least twice the maximum frequency of the signal [2]. A constant sampling rate is maintained for low-frequency signals leading to the wastage of energy and transmission resources. This classical approach is not an optimum choice for a large number of applications that require the devices to function for long periods of time, powered by a tiny battery that cannot be replaced often. Wearable gadgets and health-care monitoring devices are some examples that require low energy source.

A solution to reduce the power consumption is an adaptive event-based sampling scheme which is an asynchronous approach to convert analogue signals to events. An event in event-based sampling scheme is a voltage spike (or something significant) [2] [14] which is generated when an analogue signal crosses a certain reference voltage (power will be only utilised when the signal crosses a certain voltage point, else there will be no power consumption when there is no signal). These events are then converted to digital

signals for processing the information present in it. If an asynchronous signal processor is used along with this technique, the problem of excessive power consumption can be solved to some extent.

## 1.2 Power dissipation

In electrical/electronic engineering, power dissipation is the sum of energy lost by the number of electrons flowing per second. In simple words, the current/ number of electrons flowing from a higher potential to a lower potential (voltage difference) applied across a conductor.

The general formula of the power is given by $P = V \times I$ equation 1.1

Where, P is power in Watt, V is voltage in Volts, and I is current in Amperes.

Power is an important parameter to design any system and cannot be neglected.

### Power Dissipation in FPGA

The amount of power that a chip dissipates per unit area is called its power density. There are two types of power density that are associated with a processor architecture : dynamic power density and static power density [1].

Dynamic power dissipation can be defined as the total power dissipated due to switching of transistors present on the chip. Clock speed along with transistor density play a vital role in determining the dynamic power. Transistors which are switched more rapidly dissipate more power than the ones which are switching at a slow rate. Therefore, increasing the clock speed causes transistors to switch at a higher pace which increases the power. Dynamic power is calculated by-

$$P_{DP} = f \times C \times V_{CC}^2 \quad \text{equation 1.2 [1]}$$

Where $P_{DP}$ is dynamic power, f is frequency at which the clock is operating , C is the load capacitance and $V_{CC}$ is the voltage applied.

Static power dissipation can be defined as the total power dissipated due to leakage of current. This leakage can be due to idle transistors which are not switching [1]. Another form of current leakage is wire resistances across the chip. The power dissipated through a wire to transfer a charge/ current from one

port to another also adds static power. Continuous capacitor charging and discharging also increases the static power. Static power can be calculated by equation 1.1.

## 1.3 Aim or the Project

The aim of the project is to design an asynchronous processor which can process the events generated by analogue-to-event converters. The event-based processing technique implemented by Junfei Mao in [2] is carried forward to design a new signal processing system which will be asynchronous in nature.

The whole project is divided into three parts: First a low-resolution analogue-to-event(AEC) converters where a design method will be selected and implemented to generate events. Second part is asynchronous signal processor where a low pass finite impulse response filter will be implemented. The processor designed by Junfei Mao [2] in 2016 was a synchronous process with 179 taps filter. The challenge in this part will be to reduce the number of taps of the filter and implement it asynchronously on FPGA. To do this analogue delay lines will be studied and implemented on hardware. Last event-to-analogue converters (EAC) will be studied and if time persists will be designed on hardware.

Apart from designing the system, the main objective of the project is to cut down the power dissipation of the system.

## 1.4 Report Outline

Following an introduction, this report is organised as follows:

**Chapter 2** describes the literature review of the first part analogue-to-event converters by converting analogue signals through different schemes presented in the chapter. The possible architecture is simulated in cadence.

**Chapter 3** describes about digital filters in detail. The aim of this chapter is to design a low pass finite impulse response filter. Comparisons of different window techniques is also presented in this chapter. A LPF is designed and simulated in MATLAB.

**Chapter 4** focuses on the digital part of the circuits in which an asynchronous processor is implemented in Verilog and designed on Field programmable gate array(FPGA), followed by an overall block diagram and the simulation result.

**Chapter 5** focuses on the analogue part of the circuit. Analogue delay lines were simulated on cadence and implemented on hardware.

**Chapter 6** describes the literature review of the third part of the project, event-to-analogue converters. An operational amplifier integrator is implemented on hardware.

**Chapter 7** presents the system level simulation of the project. The results from FPGA are then reverified on MATLAB. A result section in this chapter shows the achievement of the main objective of the project.

**Chapter 8** contains a conclusion where a summary of the achievements that have been accomplished is presented. The future works are also discussed to point out possible modifications and improvements that can be made to improve the performance of the system.

# Chapter 2

# Event-driven analogue to digital conversion

Analogue to digital conversion(ADC) is a process in which analogue signals are converted to digital signals. This conversion is done using two ways- synchronous ADC and asynchronous ADC [2]. Both methods have their own different architectures with different merits and drawbacks.

Comparison between synchronous(regular) and asynchronous(irregular) sampling are shown in table 1.1

*Table 1.1: Comparison of synchronous and asynchronous sampling schemes*

| | **Regular sampling** | **Irregular sampling** |
|---|---|---|
| **Conversion trigger** | clock | level crossing |
| **Amplitude** | quantized | exact value |
| **Time** | exact value | quantized |
| **SNR dependency** | number of bit | timer period |
| **Converter output** | amplitude | (amplitude, time) |

The project describes an asynchronous architecture which does not use a clock and this chapter deals with the literature review, simulations and hardware implementation of analogue-to-event converter(AEC).

## 2.1 Level – crossing sampling scheme

Level-crossing sampling scheme is often called as non-uniform sampling or irregular sampling of a signal in the time domain. A set of predefined amplitude reference levels are defined, and a signal is only sampled when it crosses one of these reference levels [2]. Thus, if the signal has a larger amplitude, more samples of the signal will be captured, and if the signal becomes ideal, no samples will be captured. This is shown in detail in figure 2.1.

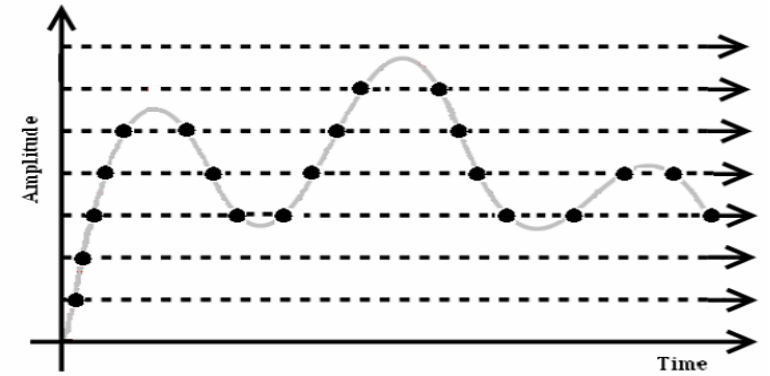



*Figure 2.1: Level crossing sampling [3]*

In the above figure, the dots represent the samples being taken at every reference they cross. The reference points taken here are equally spaced. So, whenever a signal intersects the reference level, we call that as an event(a voltage spike). A low-frequency signal will correspond to a low sampling rate whereas a high-frequency signal will correspond to a high sampling rate. This technique also avoids energy wastage (as power is not used by any clock).

## 2.2 Time Quantization

Sampling is used to make a continuous time signal discrete. A signal shown in figure 2.2, is passed through a continuous time quantiser (system without any sampling or asynchronous system), and then through a level crossing sampling scheme, as in figure 2.1. The output of that signal will be a staircase signal $x_q(t)$ as represented in figure 2.2. It is clear that, whenever a signal intersects one of the reference levels, the output takes the mid-value of the quantisation. It also generates an event which later can be used to generate a digital signal with all the information provided in the signal [2].

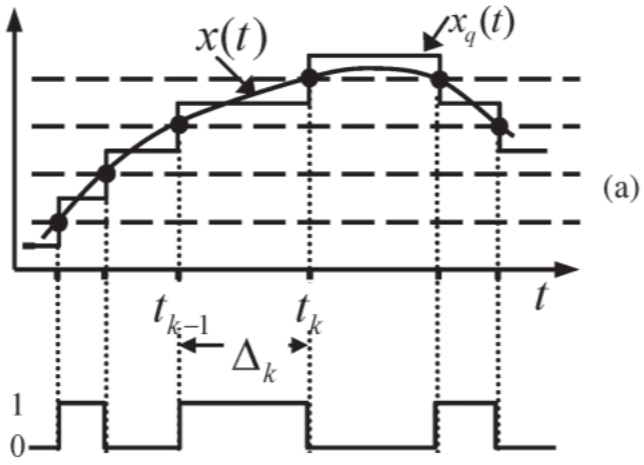



*Figure 2.2: Digital representation of a quantized signal [2]*

It can be seen in the figure 2.2 that the intervals are not equally spaced because the system has no clock of its own. The signal is being sampled asynchronously.

Compared to discrete time quantisation, continuous time quantisation has less quantisation noise. A general block diagram with spectra input signal is shown in figure 2.3.

A sinusoidal wave is used as an input to a synchronous ADC. Figure 2.3.a shows the input as a single sinusoidal wave with only one frequency in the spectrum. A sampling clock follows the Nyquist criteria [3] and samples the signal at $f_s$. In the spectrum of ADC input, there will be many frequencies(harmonics) at $kf_s \pm f_{input}$, where k is an integer, which are shown in figure 2.3.b. This results in non-linearity of the quantisation process which will lead to quantisation noise as the intermodulation product frequency will be there because of the aliasing of frequencies [4] [5] as shown in figure 2.3.c.

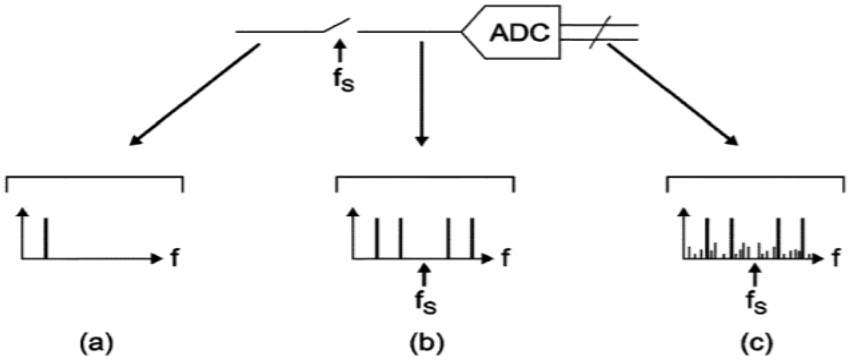



*Figure 2.3: Conventional ADC and (a) the spectrum of a sinusoidal signal; (b) the spectrum in the input of the ADC; (c) spectrum in the output of the ADC [4].*

Since there is no sampling frequency in continuous time approach, therefore it results in no quantisation noise [5].

## 2.3 Circuit Implementation

### 2.3.1 Asynchronous delta modulator

The level crossing sampling ADC was originally proposed in 1966 [6], and was called asynchronous delta modulation. Delta modulators are used to convert analogue signals to rectangular pulses. This is a type of converter which is based on a level-crossing sampling scheme in which conversion takes place whenever a reference voltage is crossed as depicted in figure 2.1**.** The dynamic range of the input signal specifies the number of reference levels. Therefore, in these converters, the dynamic range is decreased which makes the number of bits to be reduced which are allocated to the amplitude of the signal [7]. To prevent quantisation error propagation, quantisation is done for the time difference between the two consecutive discrete samples.

One such modulator is shown in figure 2.4. The demodulator of this system is shown in appendix A1**.**

Let δt be the error of time quantisation, the error in the amplitude of demodulated wave will be δV which is derived from the input signal(V) and δt with

$$\delta V_1 = \frac{dV_1}{dt}.\ \delta t \qquad \text{equation 2.1}$$

In the above equation 2.1, $V_1$ is the input of level crossing sampler, $\frac{dV_1}{dt}$ is the slope.

The quantisation noise power is given by

$$P(\delta V_1) = P(\frac{dV_1}{dt}).\ P(\delta t) \qquad \text{equation 2.2}$$

At each time instant, $V_1$ is the input analogue signal $V_{in}$ which was subtracted from a selected reference level. Therefore $\frac{dV_{in}}{dt} = \frac{dV_1}{dt}$.

The signal to noise ratio (SNR) of asynchronous delta modulator (ADM) can be written as

$$SNR_{dB} = 10.\log(3\frac{P(V_{in})}{P(\frac{dV_{in}}{dt})}) + 20.\log(\frac{1}{T_c}) \qquad \text{equation 2.3}$$

Where $P(V_{in})$ and $P(\frac{dV_{in}}{dt})$ are the power of the input signal and its derivation and $T_c$ is a timer [7]. SNR or the demodulated signal (reconstructed signal) can be increased by decreasing $T_c$ and increasing precision of time quantizer.

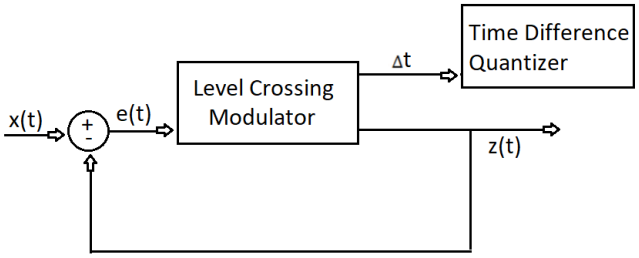



*Figure 2.4: The modulator of delta crossing [6].*

### 2.3.2 Flash converter

Another approach to achieve the level crossing sampling scheme is by using asynchronous flash ADC. This is one of the fastest methods for ADC converters. This method requires the use of voltage dividers with $2^N$ resistors and $2^{N-1}$ comparators. The inverting input of comparators is connected to the resistors which forms a voltage divider circuit (this is explained in table 2.2). The input signal is fed to the non- inverting input of the comparator. As the input signal cross a certain level of the reference voltage, the comparator output goes high ; or low when $V_{in}$ (input voltage) is less than $V_{REF}$ (reference voltage). All this together generates a digital thermometer code [8] which looks like 000, 001, 010, 011….111 etc.

After a thermometer code is generated, the signal is then passed through an encoder to get an efficient digital signal which has fewer number of bits when compared to thermometer code. Figure 2.5 represents the Flash converter with an encoder(8X3). Figure 2.6 represents a delay circuit either sample and hold or analogue delay with a feedback loop to the input [9] [10]. This circuit is a comparator which compares the output signal with the input signal to determine if the signal is UP signal(UP event) or DOWN signal(DOWN event).

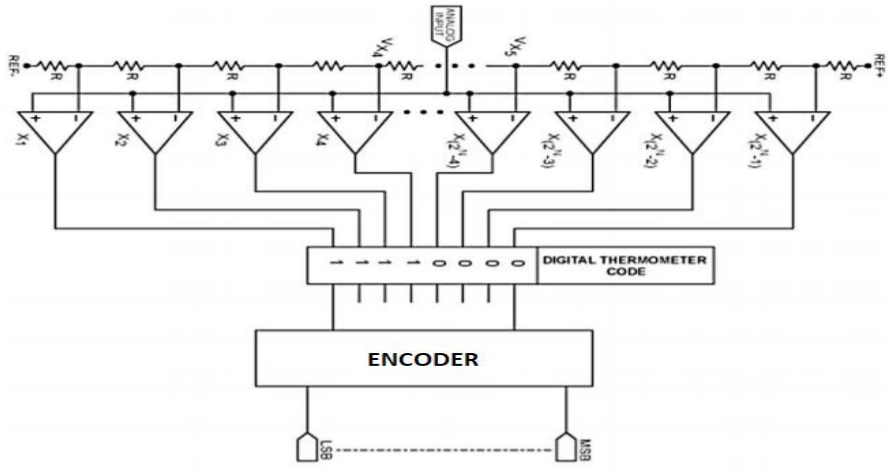



*Figure 2.5: Flash converter with 8X3 encoder [4].*

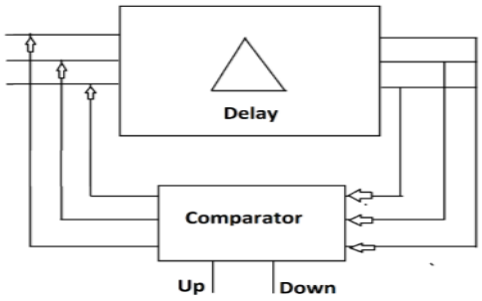



*Figure 2.6: Comparator converts coded signal to events.*

## 2.4 Circuit level simulation

### 2.4.1 Verilog-A

Verilog-A is an analogue hardware descriptive language. This language helps designers to create behavioural blocks of the circuit and define them by the use of mathematical relationships between voltage terminals and currents. Earlier C language was used, but nowadays Verilog-A is being used by many designers because of its versatilities over C [11]. It gives an advantage of an hardware descriptive language (HDL) such as Verilog, VHDL. It allows designers to work on a simulator free interface. It can be used with MATLAB and cadence for simulations.

### 2.4.2 Comparator

Comparators are the components which are used to compare the input voltage with the reference voltage. If the input voltage is greater than the reference voltage, it will assign the output as logic ‘1' else logic '0'.

Code snippet of comparator designed in cadence is shown in listing 2.2.

```
analog begin
@(cross((V(vin)-V(ref)),0) or initial_step )
begin                          //Condition for comparator
if(V(vin) >= V(ref))
result = V(vdd);

else
result = V(vss);
end                            //Assigning the result to
                               //the outport port
V(vout) <+ transition (result, delay, ttime);
```

*Listing 2.1: code snippet for comparator module.*

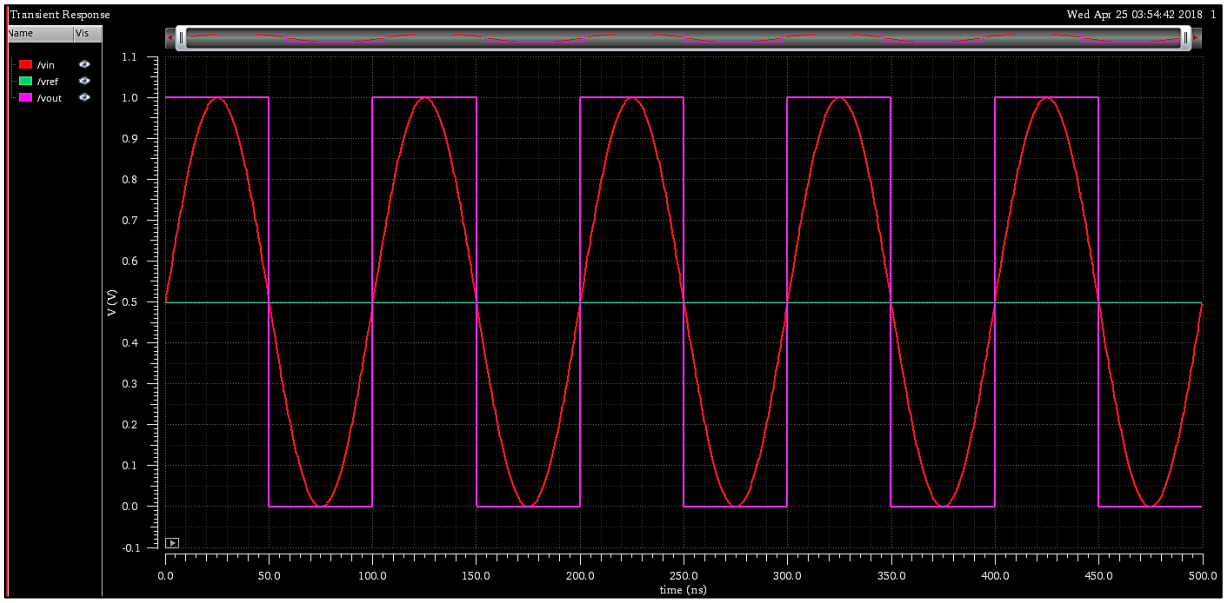



*Figure 2.7: Simulation output of comparator module.*

In figure 2.7, the input signal is a sine wave ($V_{in}$) of amplitude 0-1*V*, $V_{ref}$ if 0.5*V* and the output is a square wave (pink colour). Refer to appendix A1 for circuit diagram and code.

### 2.4.3 8-bit Flash ADC

This module is implemented on Verilog-A in cadence. Eight comparators were used as the input signal is of 3 bits (N = 3). If N = 3, therefore $2^N = 8$.

The dynamic range of ADC is 0-10*V*. A resistor chain of 8 resistors was used with the resistance of 1*kΩ*.

The voltage drop across every resistor can be calculated by:

$$V_{in} = \frac{R_2}{R_2 + R_1} \quad \text{where } R_2 = R_1 \qquad \text{equation 2.4 with respect to figure 2.8}$$

The voltages after applying the above equation are given in table 2:

*Table 2.2: Reference voltages for flash ADC calculated using equation 2.4*

| S.No. | Voltage (V) |
|---|---|
| 1 | 1.25 |
| 2 | 2.5 |
| 3 | 3.75 |
| 4 | 5 |
| 5 | 6.25 |
| 6 | 7.5 |
| 7 | 8.75 |
| 8 | 10 |

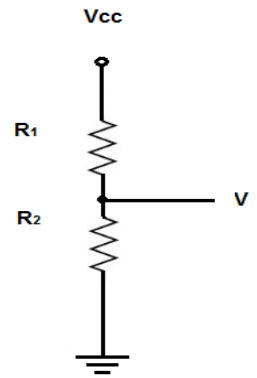



*Figure 2.8: Voltage divider*

```
                        //Main block starts here (Encoder)
 analog begin

                        //Conditions of Thermometer code or Encoder
 if (V(t7) == 10)
     begin
     dzero   = 1;
     done    = 1;
     dtwo    = 1;
 if (V(t7) == 10)
     begin
     dzero   = 1;
     done    = 1;
     dtwo    = 1;
     end

 else if (V(t6) ==10)
     begin
     dzero   = 0;
     done    = 1;
     dtwo    = 1;
     end
```

*Listing 2.2: code snippet for flash ADC*

*Table 2.3: Output of the Flash ADC*

| Reference Voltage (V) | Output |
|---|---|
| 1.25 | 000 |
| 2.5 | 001 |
| 3.75 | 010 |
| 5 | 011 |
| 6.25 | 100 |
| 7.5 | 101 |
| 8.75 | 110 |
| 10 | 111 |

The simulation output of figure 2.5 with snippet listing 2.2 is as follows:

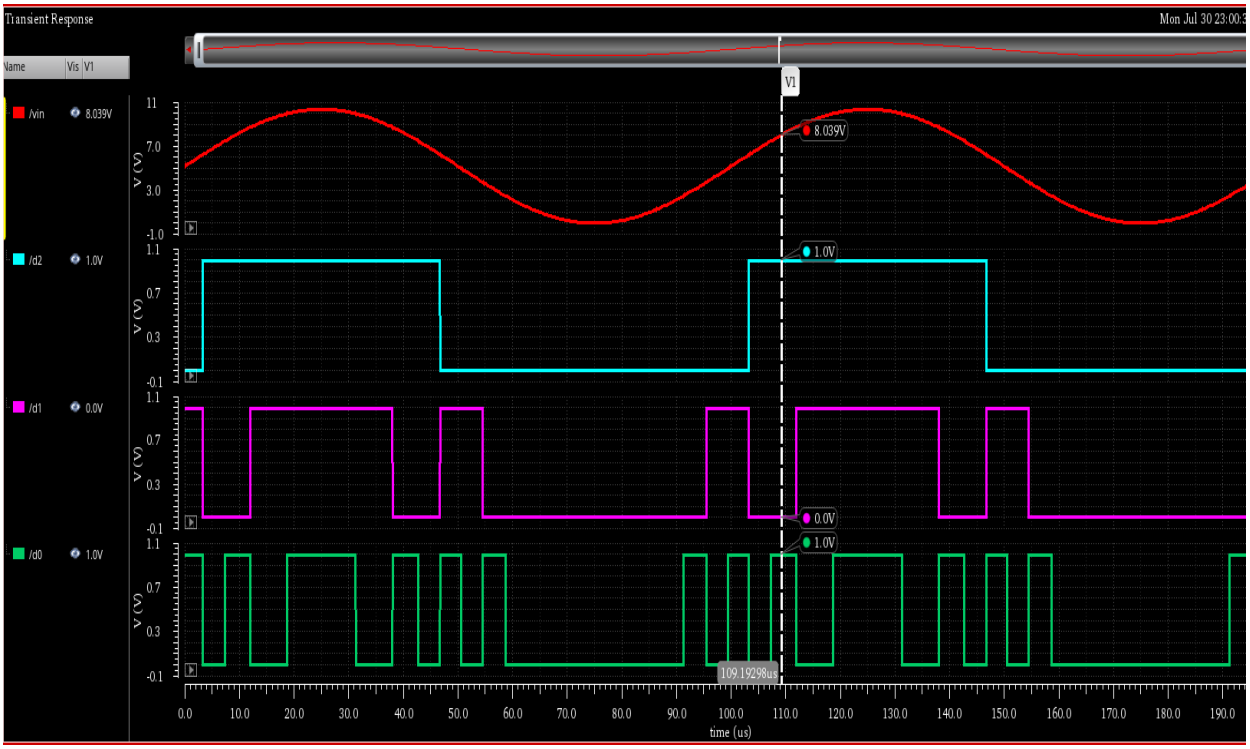



*Figure 2.9: Simulation output of sine with flash ADC and an encoder*

The input signal is a sinusoidal wave of 10*V* amplitude. The pointer is at 8.039*V* which means the output should be 101 (table 2.3). The output from the above simulation is 101 where d0, d1, d2 are the outputs of the encoder.

For more information about the source code, circuit diagrams and simulations for this section, please refer to appendix A1.

## 2.5 Event Generator

Now that the input signal has already been converted/ encoded to the 3-bit thermometer code [8], a block is required to generate events from the encoded data. For this, either a state machine or a logical block can be made.

In the figure 2.10, the input signal(x) is compared with a delayed input signal(y). This delay is generated using a small synchronous block. So, when the input signal(x) is greater than the delayed input signal(y) then an event is created which is UP in this case and likewise for the DOWN signal. When the signal is higher than the delayed signal, it is going to state S1 where the value of UP is logic high. Whenever the signal is decreasing (smaller than the delayed signal) it is going to state S2 where the value of DOWN is logic high. As S0 state is a state when the input equals to delayed input.

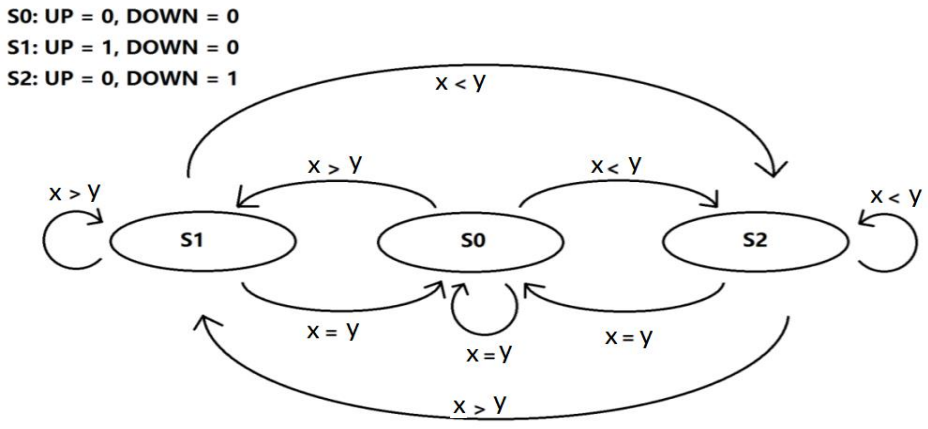



*Figure 2.10: A structure of a finite state machine with three stages [4]*

The above logic can also be implemented by if-else statements as the system is asynchronous, the use of state machines is generally not recommended.

The code snippet for the figure 2.10 is implemented using if-else statement (in Verilog) is shown below

```
always @*
begin
    if (in > Delayed_in)begin      // If input is greater
    Up_inl   = 1'b1;
    Down_inl = 1'b0;
    end
    else
    if ( in < Delayed_in)begin     // If input is smaller
    Up_inl   = 1'b0;
    Down_inl = 1'b1;
    end
    else begin                     // Default state or if
    Up_inl = 1'b0;                 // input euqals to delayed_in
    Down_inl = 1'b0;
    end
    end
                                   // Assigning the signals
assign Up_in   = Up_inl;
assign Down_in = Down_inl;
```

*Listing 2.3: Code snippet for the event generator*

In simulations, a 5*Khz* frequency signal is given as an input to FPGA, and the output of the event generator is as follows.

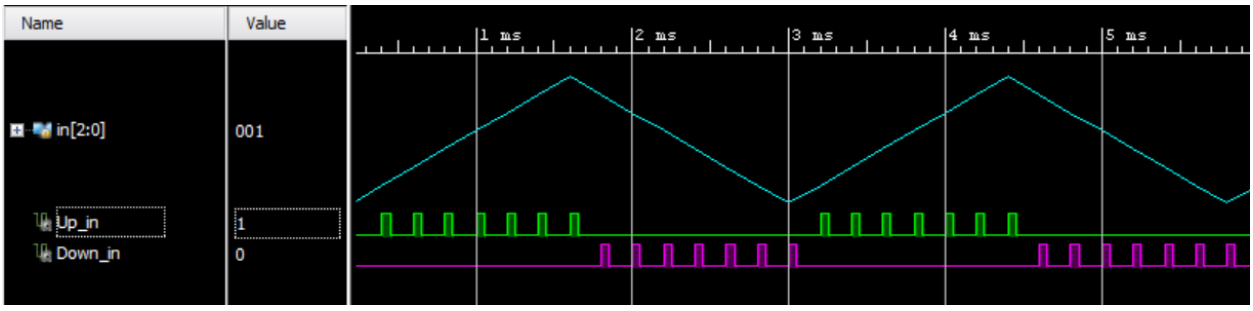



*Figure 2.11: Event generator simulation output*

The events/spikes are generated at 50$\mu s$ because the delay given in Verilog is of 50$\mu s$. So, after every 50$\mu s$, the output is generated by comparing the input signal with the delayed input signal.

## 2.6 Chapter Summary

Chapter 2 describes two different asynchronous coding schemes and their possible circuit implementations. Asynchronous flash ADC is explained in detail with its Verilog-A code and simulation results on cadence. More detailed circuit diagrams and simulations with their respective source codes are also described in the appendix A1. An event generator is explained and simulated on FPGA. Now to process these signals a signal processor is required which is discussed in the later chapters.

# Chapter 3:

# Filter

Filters are the building block of every signal processing system. The function of a filter is to block or remove the unwanted noise, random signal or to extract a particular signal or parts of the signal from a group of signals.

The block diagrams explain the basic idea of the working of a filter.

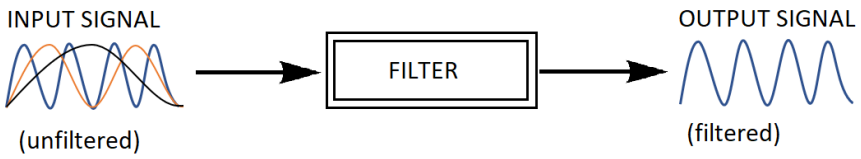



*Figure 3.1: Block diagram of a filter system*

Filters are used for two primary purposes – signal separation and signal restoration. For example, in radio frequency(RF) filters, signal separation is done by selecting only a particular frequency or set of frequencies and removing the unwanted frequencies by attenuating them. Signal restoration is used when a signal is distorted, for example, consider an audio recording recorded by poor equipment or in a noisy place (on the street or near an industry), this recorded signal can be filtered to obtain a good quality of sound. Another example of filtering is smoothing the image, enhancing or detecting the edges of an image either by suppressing the high frequencies or low frequencies present in the image.

This chapter briefly describes filters, designing techniques and simulations on MATLAB.

## 3.1 Types of filters

There are two main filters – analogue and digital filters. Both the filters are different in their implementation and designing.

### 3.1.1 Analogue filters

Analogue filters use analogue circuits and analogue equipment(components) such as resistors, op-amps, capacitors and inductors to make a filter. These filters are used to enhance the video quality, reduce the noise of a system, equalises the graphics and many more. They are designed based on the specifications of a system. They are simple to implement but hard to achieve precise results with an increase in complexity.

### 3.1.2 Digital filters

Digital filters are discrete-time, discrete-amplitude convolver [12]. They are programmable and can be made on chips which work on digital signals, for example, SoC (System on chip), DSP (digital signal processing). Digital filters are more precise, but the only condition is the signal should be a digital signal. Digital filters are programmable where the program is stored inside a processor. Therefore, it is easier to change or manipulate the digital filter because it is hardware independent. Whereas, to change an analogue filter, redesigning of the circuit is required. Digital filters are more stable than their analogue counterparts as they are not affected by the problems of active components such as drift, temperature etc.

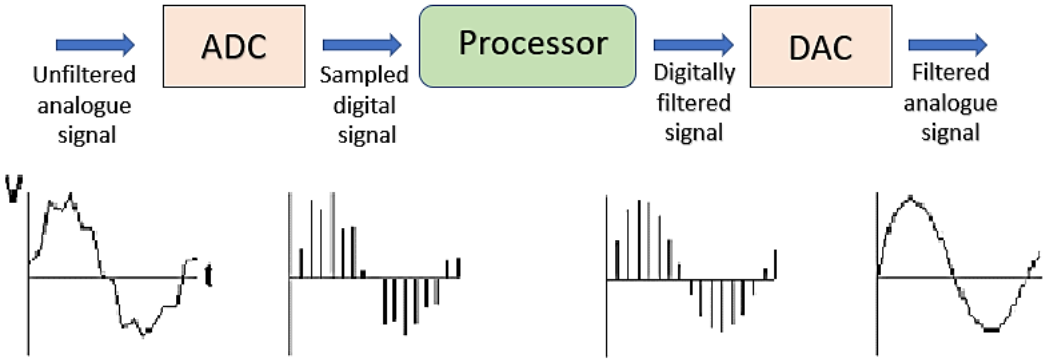



*Figure 3.2: Block diagram of a system using a digital filter*

The analogue signal must be converted to a digital signal using ADC (analogue to digital converters). The resulted binary signals are then fed into a processor which performs numeric calculations on the digital signal. These calculations, most of the times are the multiplication of the input signal with constants and addition of the products. Then, if required, this filtered output is then passed through a DAC (digital to analogue converter) to get a filtered analogue output. More detailed steps are shown figure 3.2.

There are two major types of digital filters – finite impulse response filters (FIR) and infinite impulse response digital filters (IIR).

The basic characteristics of an IIR filter are:

- They have non- linear phase characteristic
- They have low filter order and less complex circuitry
- They are prone to instability

The basic characteristics of using an FIR filter are:

- They have a linear phase characteristic
- They have high filter order and more complex circuitry
- They are stable

Advantages of using an FIR over IIR is that FIR can be made and realised on hardware more efficiently. The delays in FIR filter is often more than IIR filter which is one of the leading disadvantages of FIR filter [13].

In this project, an FIR filter is designed which is discussed in the later in the chapter.

## 3.2 Ideal Filter

Ideal filter is also called a brick wall filter because of its response. Its response is 1 in the passband and 0 otherwise for a low pass filter [14]. The diagram is shown below.

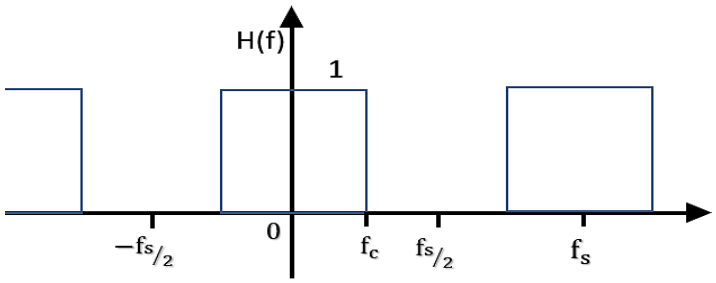



*Figure 3.3: Ideal filter*

Here $f_c$ is the cut-off frequency whereas $f_s$ is the sampling frequency.

Ideal filter is hard to construct. These are explained with the help of the equations below

Inverse Fourier transform of an ideal filter is:

$$h(\omega) = \begin{cases} 1; & |\omega| \leq \omega_c \\ 0; & \omega_c < |\omega| \leq \pi \end{cases}$$

$$h(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} h(\omega)\, e^{j\omega n} d\omega$$

$$= \frac{1}{2\pi}\left[\frac{1}{jn} e^{j\omega n}\right]_{-\omega_c}^{\omega_c}$$

$$= \frac{1}{\pi n}\frac{1}{2j}\left\{e^{j\omega_c n} - e^{-j\omega_c n}\right\}$$

$$= \frac{1}{\pi n} sin(\omega_c n)$$

$$= \frac{\omega_c}{\pi}\frac{sin(\omega_c n)}{\omega_c n} \qquad \text{equation 3.1 [4]}$$

According to the above equation 3.1, the impulse response, h(n) is 0 when n = N X $2\pi\omega_c$, except when n = 0 where h(n) = $\omega_c/\pi$ from l'Hôpital's rule) [14]. This implies, there is a delay of an infinite length before the output is obtained. Plus, for values n <0 for which h(n) ≠ 0. The filter is not causal, i.e. it requires future values to determine the present values. All these issues make it ideal filter which is almost impossible to achieve in the real world.

## 3.3 Practical Filter

A practical filter is different from the ideal filter mentioned above. It has ripples both in the passband and stopband because the frequency response is not equal to zero which is explained in detail in figure 3.4. The range between the passband edge and stopband edge is defined as transition band. $\omega_p$ is the passband edge where the filter response drops below to cut-off (-3*dB*). $\omega_s$ is the stopband edge where the response drops to stopband height [14]. Reducing the width of the transition band (δ1, δ2) can improve the efficiency of the filter but will lead to a more complex design.

In the following figure 3.4–

$\delta 1$ – Passband ripple

$\delta 2$ – Stopband ripple

$\omega_p$ – Passband edge frequency

$\omega_s$ – Stopband edge frequency

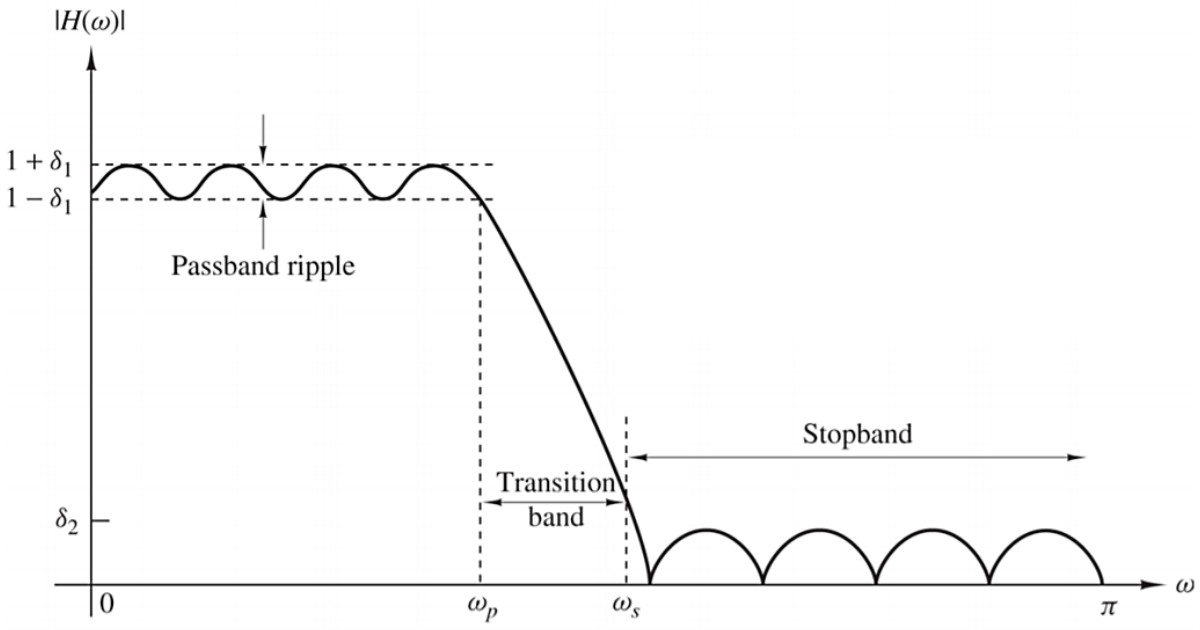



*Figure 3.4: Magnitude characteristics of a low pass filter [14]*

## 3.4 FIR filter

In DSP, an FIR is a filter whose impulse response is of a finite period. They are majorly executed on software. There are various kinds of filters-

- Low pass filter (LPF)
- High pass filter (HPF)
- Band pass filter (BPF)
- Band stop filter (BSF)

A LPF allows only low frequency of signals to pass and block the high frequencies from the signal. This type is filter often used for audio signals to control the frequencies in a sound.

High pass filter is precisely opposite of LPF. It blocks the low frequencies (frequency which is less than the cut-off frequency) and allows high frequencies to pass through it. An example of HPF is to amplify the high frequency of a signal using audio amplifiers.

BPF is a type of filter which allows a particular set of frequencies of the signal to pass through it. An example of this filter can be an audio filter used in telephones to limit the signal from 250*Hz*-5*kHz*.

BSF is also called as reject filter because of its nature. Opposite of BPF, BSF rejects/ stops a particular window of the signal and allows the rest of the frequencies to pass through it. A notch filter is a perfect example of a BSF.

The designing of an FIR filter is based on the approximation of the ideal filter. The design process depends on specifications and implementation of the filter. There are three methods to design an FIR filter, but window method is used mostly oversampling frequency method and optimal filter design method.

### 3.4.1 Structure of FIR

The building blocks of FIR filter are

- Multipliers : $y(n) = h_N \times x(n)$
- Adders : $y(n) = x_1(n) + x_2(n)$
- Series of delays : $y(n) = x(n-1)$

All these together generates the output.

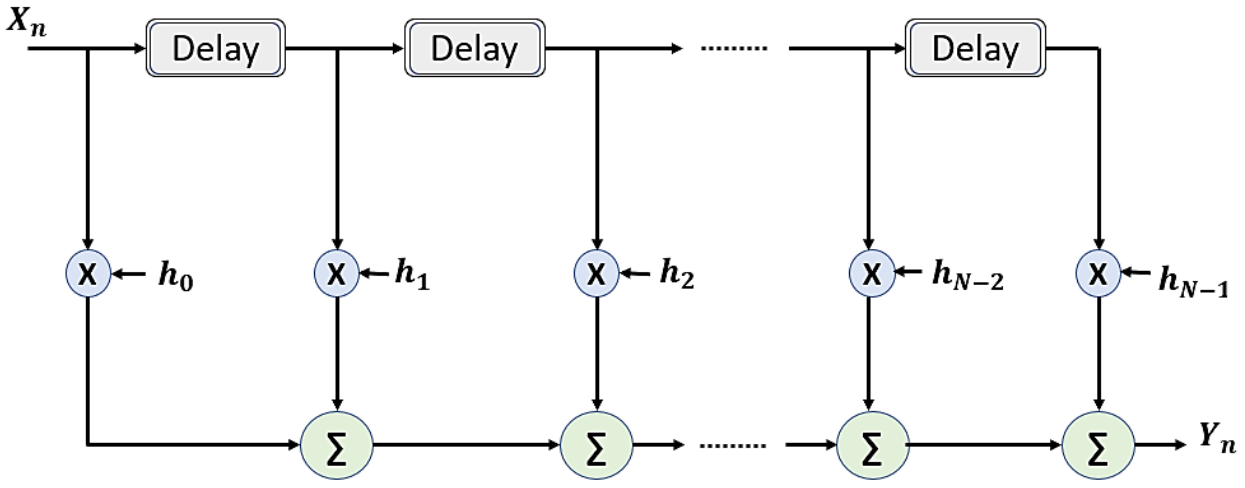



*Figure 3.5: Logical Structure of FIR filter*

The relationship between x(n) and y(n) is shown below:

$$y(n) = h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) + h(3)x(n-3) + \ldots\ldots\ldots + h(N-1)x(n-N-1)$$

$$= \sum_{m=0}^{N-1} h(m)x(n-m) \qquad \text{equation 3.2 [14]}$$

$$\cong (h*x)(n)$$

Where "*" is a convolution operator.

The above equation 3.2 can also be written as-

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k\tau) \qquad \text{equation 3.3 [14]}$$

where $\tau$ is the delay line and h(k) is a coefficient.

### 3.4.2 Window method

When an FIR filter is designed using the window method, the ideal frequency response $h_D(\omega)$ of a filter is defined. Once it has been obtained, the inverse ideal response $h_D(n)$ is calculated. Then the next task is to select a window which best suits the specifications of the system *δ1* and *δ1* as shown in figure 3.4.

Table 3.1 shows the ideal response of different types of filter. Table 3.2 illustrates different types of windows and their mathematical definitions. Once the window function is chosen based on specifications, then the filter coefficients "N" is calculated. This is done by multiplying the coefficients of window function w(n) with the ideal impulse response.

In Table 3.3 below, illustrates various window functions and their corresponding properties to enable in the design of a filter.

Once the appropriate window function has been determined by the designer based on the filter specification, then the required number of filter coefficients 'N' can then be calculated. The final procedure of the design process is to determine the actual filter coefficients h(n) by multiplying the coefficients of the window function w(n) with the corresponding ideal impulse response $h_D(n)$. This is shown in the equations below [14] [15].

Frequency response of filter $h(n) = w(n).\ h_D(n)$ equation 3.4

Transfer function of filter $h(e^{j\omega}) = \sum_{n=0}^{N} h(n).\ e^{-jn\omega}$ equation 3.5

Z transform of filter $h(z) = \sum_{n=0}^{N} h(n).\ z^{-n}$ equation 3.6

The following tables shown below shows different $h_D[n]$, w[n] and a window function.

*Table 3.1: Ideal response of FIR filter types [15] [16].*

| Filter type | $h_D[n],\ n \neq 0$ | $h_D[n],\ n = 1$ |
|---|---|---|
| Low-pass | $2F_c \frac{\sin(n\Omega_c)}{n\Omega_c}$ | $2F_c$ |
| High-pass | $1 - 2F_c \frac{\sin(n\Omega_c)}{n\Omega_c}$ | $1 - 2F_c$ |
| Band-pass | $2F_2 \frac{\sin(n\Omega_2)}{n\Omega_2} - 2F_1 \frac{\sin(n\Omega_1)}{n\Omega_1}$ | $2F_2 - 2F_1$ |
| Band-stop | $1 - \left[2F_2 \frac{\sin(n\Omega_1)}{n\Omega_1} - 2F_1 \frac{\sin(n\Omega_2)}{n\Omega_2}\right]$ | $1 - [2F_2 - 2F_1]$ |

*Table 3.2: Mathematical definition of various windows [15] [16].*

| Name of window function $w(n)$ | Mathematical definition |
|---|---|
| Rectangular | 1 |
| Hanning | $0.5 - 0.5\cos\left[\frac{2\pi n}{N-1}\right]$ |
| Hamming | $0.54 - 0.46\cos\left[\frac{2\pi n}{N-1}\right]$ |
| Blackman | $0.42 - 0.5\cos\left[\frac{2\pi n}{N-1}\right] + 0.08\cos\left[\frac{2\pi n}{N-1}\right]$ |
| Kaiser | $\frac{I_0\left[\beta\sqrt{1-\left(\frac{\lvert 2n-N+1\rvert}{N-1}\right)^2}\right]}{-I_o(\beta)}$ Where, $I_0(x) = \sum_{k=0}^{\infty}\left(\frac{x^k}{2^k k!}\right)^2$ |

*Table 3.3: Window function and their unique properties [14]*

| Name of window function $w[n]$ | Transition width ΔF in (Hz), (normalised) | Pass-band ripple $A_p$ in (dB) | Ripple $\delta_p$, $\delta_s$ | Side-lobe level in (dB) | Stop-band attenuation $A_s$ in (dB) |
|---|---|---|---|---|---|
| Rectangular | 0.9/N | 0.741 | 0.089 | -13 | 21 |
| Hanning | 3.1/N | 0.0546 | 0.063 | -31 | 44 |
| Hamming | 3.3/N | 0.0194 | 0.0022 | -41 | 53 |
| Blackman | 5.5/N | 0.0017 | 0.000196 | -57 | 74 |
| Kaiser β=4.54 | 2.93/N | 0.0274 | | | 50 |
| β=5.65 | 3.63/N | 0.00867 | | | 60 |
| β=6.76 | 4.32/N | 0.00275 | | | 70 |
| β=8.96 | 5.71/N | 0.000275 | | | 90 |

Although there are so many windows available to design an FIR filter, yet, Kaiser window is chosen in this project because of its advantages over the other windows.

- It has a good stopband attenuation
- When compared to other windows given in table 3.3 and table 3.2, Kaiser window requires minimum conceptual effort.
- The window length does not affect the passband and stopband ripple size as much as it does in other windows [17]. So, by keeping the N (window length) constant, it is easier to shape the filter (passband and stopband ripples). This is a considerable advantage over other windows.

In addition to the above reasons, it can also be seen from the work of Anurag and Rainy in "Low Power Reconfigurable FIR Filter Based on Window Techniques for On-Chip Network" [18], the Kaiser window has the minimum power consumption over other window functions.

## 3.5 Kaiser Window

Kaiser window is defined as-

$$w(n) = \frac{I_o\left(\beta\sqrt{1-\left(\frac{n-N/2}{N/2}\right)^2}\right)}{I_o(\beta)} \qquad \text{where } 0 \leq n \leq N \qquad \text{equation 3.7 [19]}$$

$I_o$ is the zeroth-order Bessel function of the first kind which is given by

$$I_o = 1 + \sum_{k=1}^{\infty} \frac{\left[\left(\frac{x}{2}\right)^k\right]^2}{k!} \qquad \text{equation 3.8 [14]}$$

The value of $\beta$ is calculated using the parameter $\alpha(\text{attenuation})$ dB. The value of $\beta$ is shown below;

$$\beta = \begin{cases} 0.1102(\alpha - 8.7), & \alpha > 50 \\ 0.5842(\alpha - 21)^{0.4} + 0.07886(\alpha - 21), & 50 \geq \alpha \geq 21 \\ 0, & \alpha < 21 \end{cases} \qquad \text{equation 3.9 [19]}$$

and $\alpha = -20\log_{10}(\delta)$

This project uses a Kaiser window to design an FIR filter which is discussed in the next section of this chapter.

## 3.6 Filter design

FIR filter designed in this project has following specifications: -

- Sampling Frequency = 20*kHz*
- Passband edge = 3*kHz*
- Stopband edge = 6*kHz*
- Stopband attenuation = -40*dB*
- Design method = Kaiser window
- Beta value ($\beta$) = 3.39 calculated from equation 3.9

The following simulations are done on MATLAB. MATLAB is a programming platform used by millions of scientists and engineers to simulate, verify, build and test circuits or algorithms.

Code snippet to generate a Kaiser window filter is shown in Listing 3.1

```
c = kaiserord([3000 6000],[1 0],[0.89 0.01],20000,'cell');
bcell = fir1(c{:});
hfvt = fvtool(bcell,1);
legend(hfvt,'kaiser')
```

*Listing 3.1: Snippet to generate a Kaiser window in MATLAB*

### I Magnitude Response

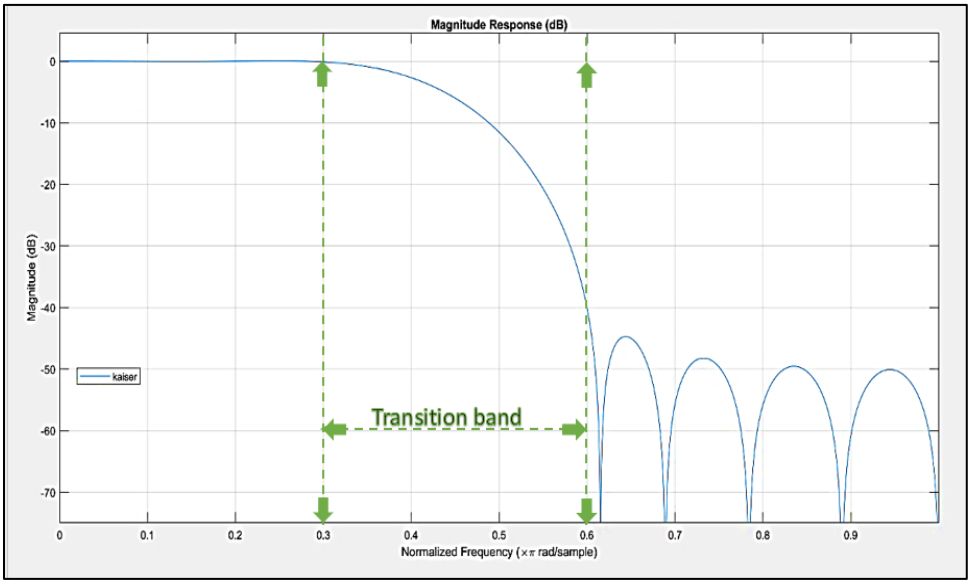



*Figure 3.6: Magnitude response of FIR low pass filter*

According to the specifications mentioned above the FIR filter is designed with a passband edge at 3*kHz* (or 0.3 X *fs*/2 (normalised frequency here is 10k)). Stopband edge is at 6*kHz* which meets the 3rd criteria. The stopband attenuation is -40*dB*, and it can be seen that all the ripples are greater than -45*dB*.

## II Phase response

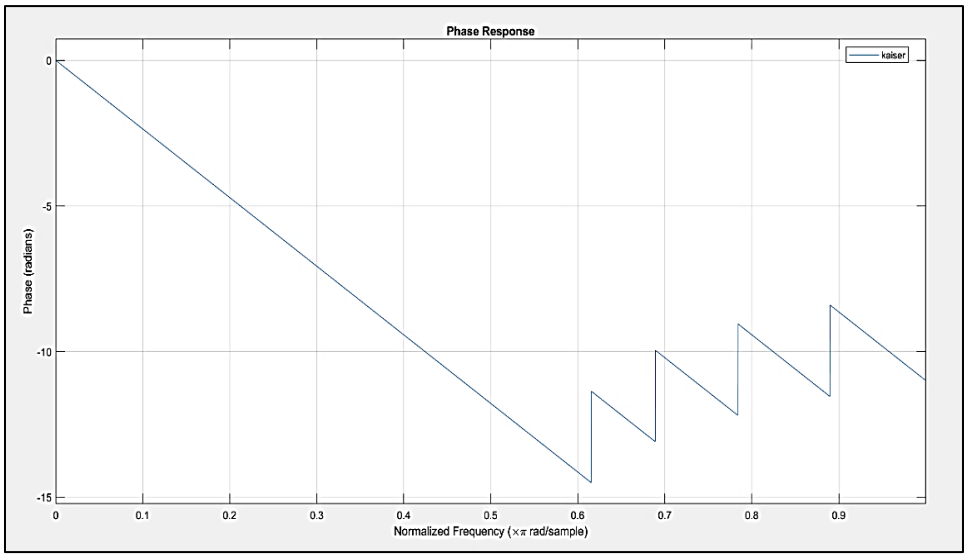



*Figure 3.7: Phase response of the filter*

The above figure 3.7 shows that the phase of the filter is linear.

## III Poles and Zeros

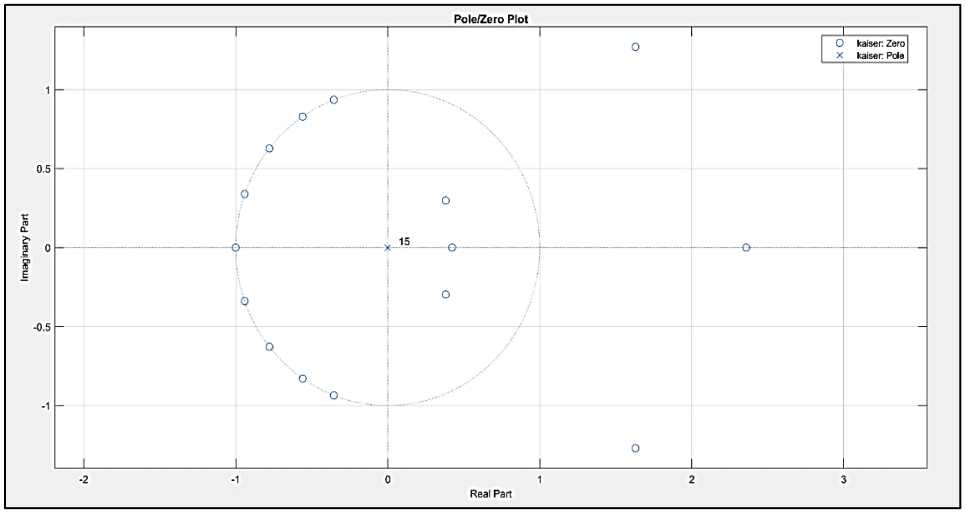



*Figure 3.8: Poles and zeros of the filter*

The filter is stable as there is only one pole which is at the centre/origin . The rest of the 15 circles in figure 3.8 is the zeros.

## IV Impulse response

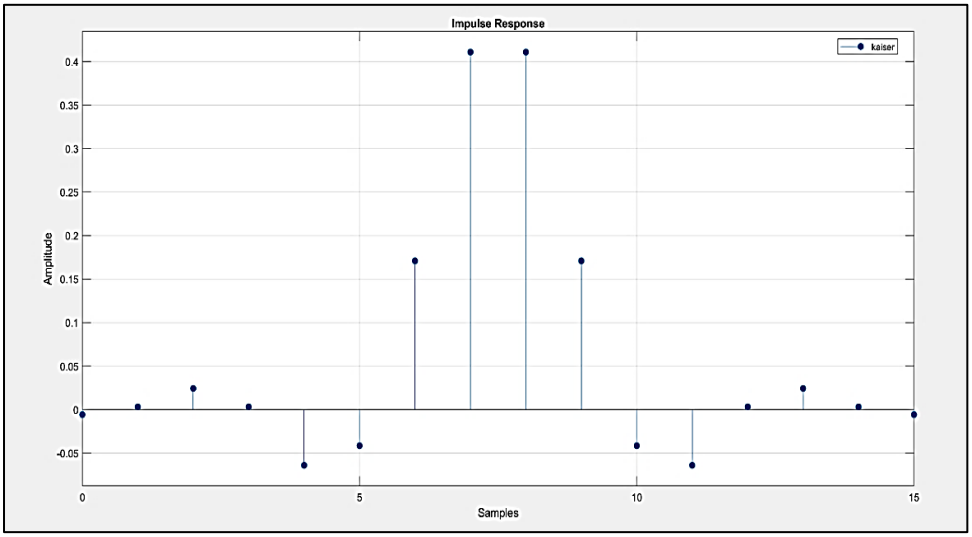



*Figure 3.9: Impulse Response of the Filter*

The impulse response of the filter gives the coefficients of the filter.

## V Filter coefficients

These are the 16 filter coefficients generated through MATLAB (impulse response). These filters are then converted to their nearest three decimal places and used later in the project as coefficients ($C_K$ in FPGA). This is shown in table 3.4

*Table 3.4: Revised filter coefficients*

| S.No. | Filter Coefficients (h(n)) | New Filter Coefficients (h(n)) |
|---|---|---|
| 1 | -0.00580182 | -0.006 |
| 2 | 0.003140629 | 0.003 |
| 3 | 0.02430697 | 0.024 |
| 4 | 0.003195267 | 0.003 |
| 5 | -0.064017181 | -0.064 |
| 6 | -0.041473844 | -0.041 |
| 7 | 0.170880665 | 0.171 |
| 8 | 0.410851226 | 0.411 |
| 9 | 0.410851226 | 0.411 |
| 10 | 0.170880665 | 0.171 |
| 11 | -0.041473844 | -0.041 |
| 12 | -0.064017181 | -0.064 |
| 13 | 0.003195267 | 0.003 |
| 14 | 0.02430697 | 0.024 |
| 15 | 0.003140629 | 0.003 |
| 16 | -0.00580182 | -0.006 |

## 3.7 Simulations in MATLAB

w(n) of the window is calculated from MATLAB.

Code snippet to generate a Kaiser window w(n) is shown in Listing 3.2

```
H = sigwin. kaiser (16,3.384);
win = generate(H)
wvtool(H)
hfvt = fvtool(win,1);
```

*Listing 3.2: Kaiser window with beta 3.384*

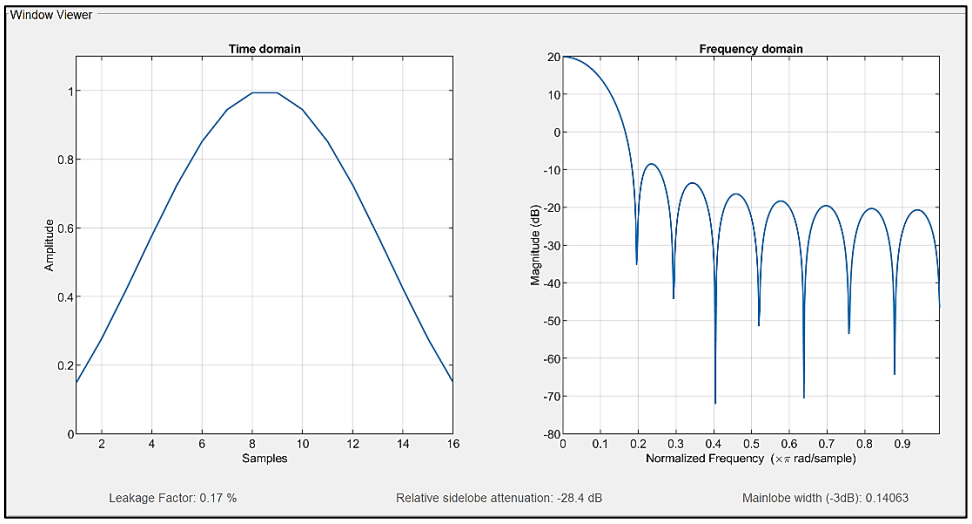



*Figure 3.10: Kaiser window with Beta 3.384*

The coefficients generated through MATLAB for the above window are as follows: -

**Kaiser window Coefficients**

*Table 3.5: Kaiser window coefficients w(n) with beta 3.384*

| S.No. | Coefficients |
|---|---|
| 1 | 0.1486 |
| 2 | 0.2755 |
| 3 | 0.422 |
| 4 | 0.5764 |
| 5 | 0.7244 |
| 6 | 0.8515 |
| 7 | 0.9445 |
| 8 | 0.9937 |

| S.No. | Coefficients |
|---|---|
| 9 | 0.9937 |
| 10 | 0.9445 |
| 11 | 0.8515 |
| 12 | 0.7244 |
| 13 | 0.5764 |
| 14 | 0.422 |
| 15 | 0.2755 |
| 16 | 0.1486 |

To design a filter h(n), $h_D$ and w(n) are required which is calculated from MATLAB and shown above.

To check if the filter is working correctly with the new filter coefficients $h_N$ (refer to table 3.4), h(n) is calculated by multiplying the window coefficients(w(n)( refer to table 3.5)) with the new filter coefficients $h_{DN}$. Using equation 3.4 $h(n) = w(n) \times h_N$

*Table 3.6: New filter coefficients generated using equation 3.4*

| S.No. | Coefficients |
|---|---|
| 1 | -0.0008916 |
| 2 | 0.0008265 |
| 3 | 0.010128 |
| 4 | 0.0017292 |
| 5 | -0.0463616 |
| 6 | -0.0349115 |
| 7 | 0.1615095 |
| 8 | 0.4084107 |

| S.No. | Coefficients |
|---|---|
| 9 | 0.4084107 |
| 10 | 0.1615095 |
| 11 | -0.0349115 |
| 12 | -0.0463616 |
| 13 | 0.0017292 |
| 14 | 0.010128 |
| 15 | 0.0008265 |
| 16 | -0.0008916 |

The filter generated by these filter coefficients (table 3.6) is shown in the figure below:

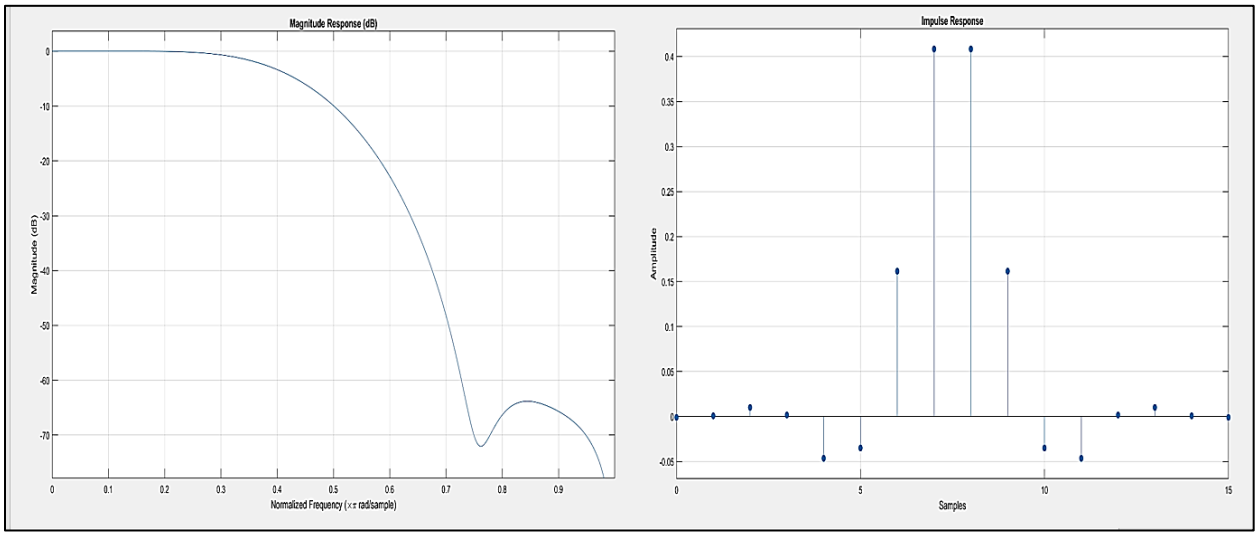



*Figure 3.11: Low pass filter response with new coefficients*

The magnitude response and the filter response of the newly generated filter are the similar to that of the original filter designed in figure 3.8 and 3.9**.** This means the filter meets the above-required specifications with the new coefficients.

### 3.7.1 LPF with different types of input signals

**a**) A sine wave with frequency (2*$\pi$*0.1*n) in passed through the filter, where n is 0 < n < 32. To check the magnitude response of the input, refer to appendix A2.

As it is a 2*kHz* signal, it should pass through the designed low pass filter. The output magnitude response of the FIR low pass filter is shown in figure 3.12

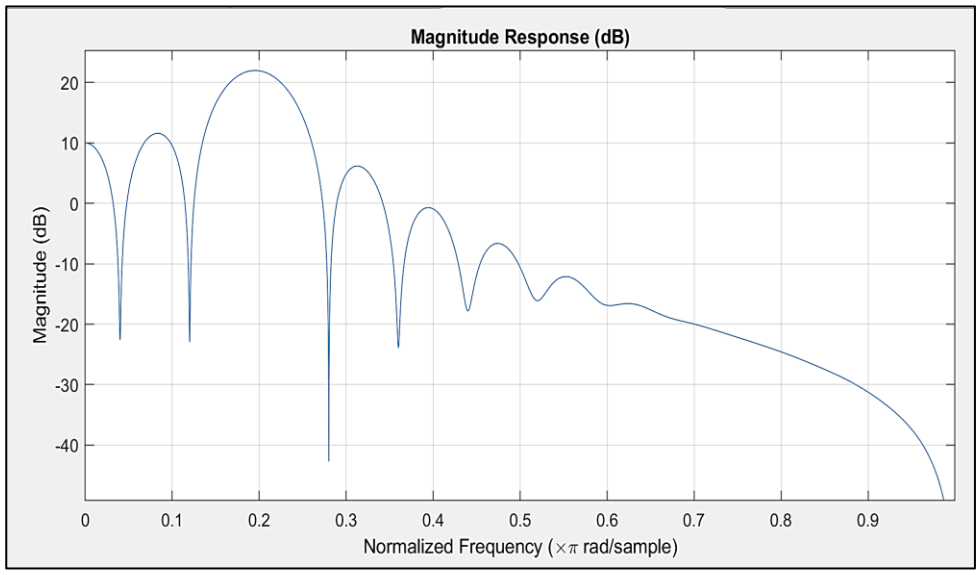



*Figure 3.12: Magnitude response of input sine wave of 2kHz*

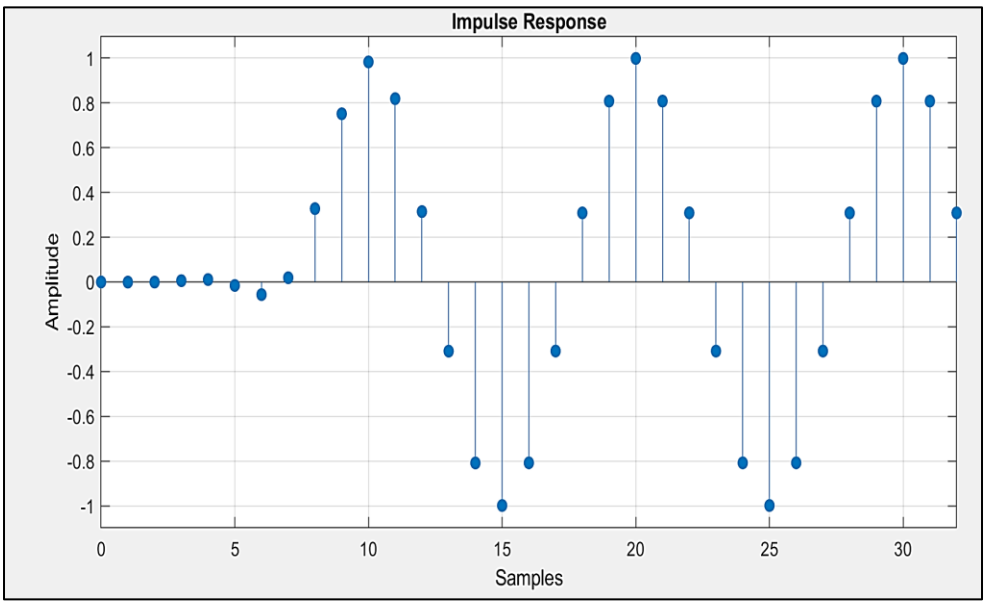



*Figure 3.13: Impulse response of the 2kHz sine wave*

From the above two figures (figure 3.12, figure 3.13) it can be concluded that the input signal is passing through the designed filter. As expected, there is no attenuation for the input signal because it was below the cut-off frequency.

**b**) A combination of different signals (high-frequency signal and low-frequency signal) is given as an input to the designed filter.

*mm = cos(2*π*0.4*n) + sin(2*π*0.1*n) + cos(2*π*0.35*n) + sin(2*π*0.15*n)* where 0 < n < 32

Here two low-frequency signals and two high-frequency signals are passed through the filter.

Figure 3.14,3.15 shows the responses of the input signal.

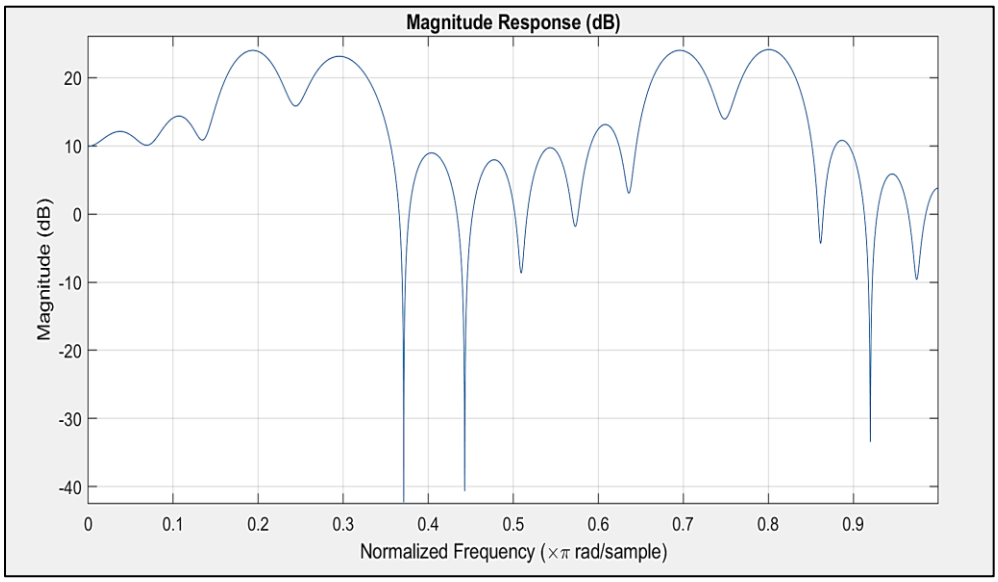



*Figure 3.14: Magnitude response of the mixed input signal*

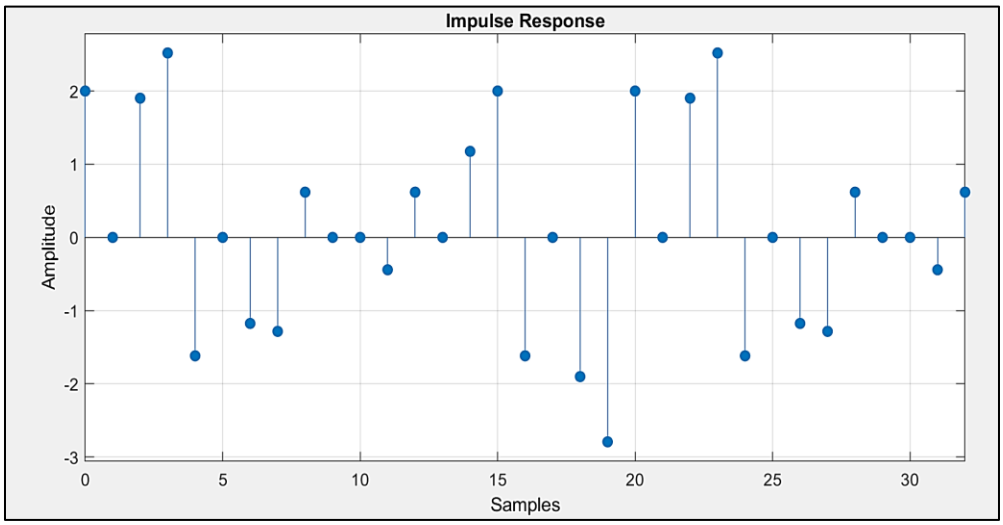



*Figure 3.15: Impulse response of the mixed input signal*

From figure 3.14, the input signal has four peaks, 0.2, 0.3, 0.7 and 0.9 (2*kHz*, 3*kHz*, 7*kHz* and 9*kHz*). The filter is expected to pass 2*kHz* and 3*kHz* signal and block 7*kHz* and 9*kHz* signal.

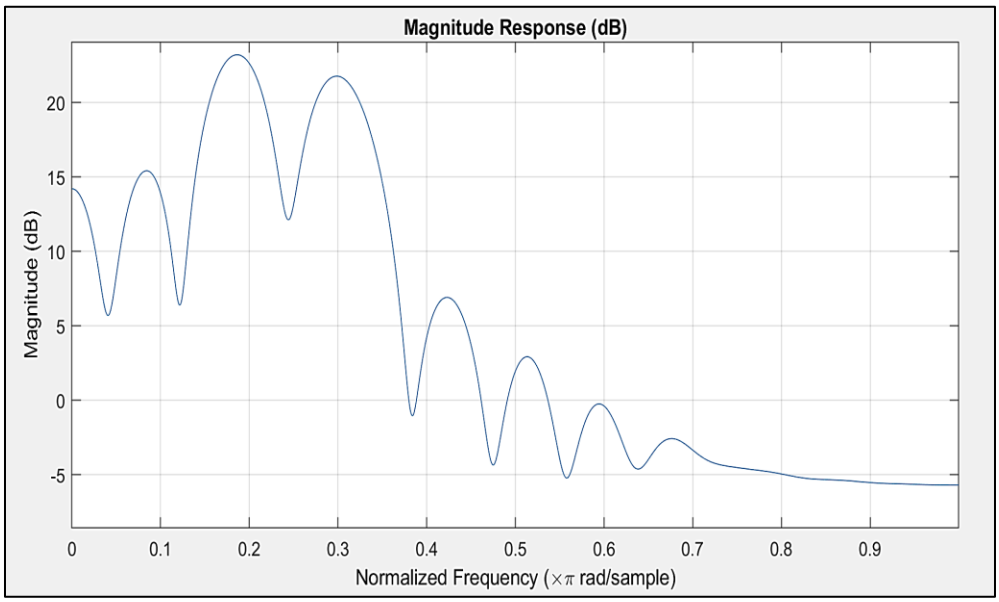



*Figure 3.16: Magnitude response of filter when the mixed input is passed (refer figure 3.14 for input)*

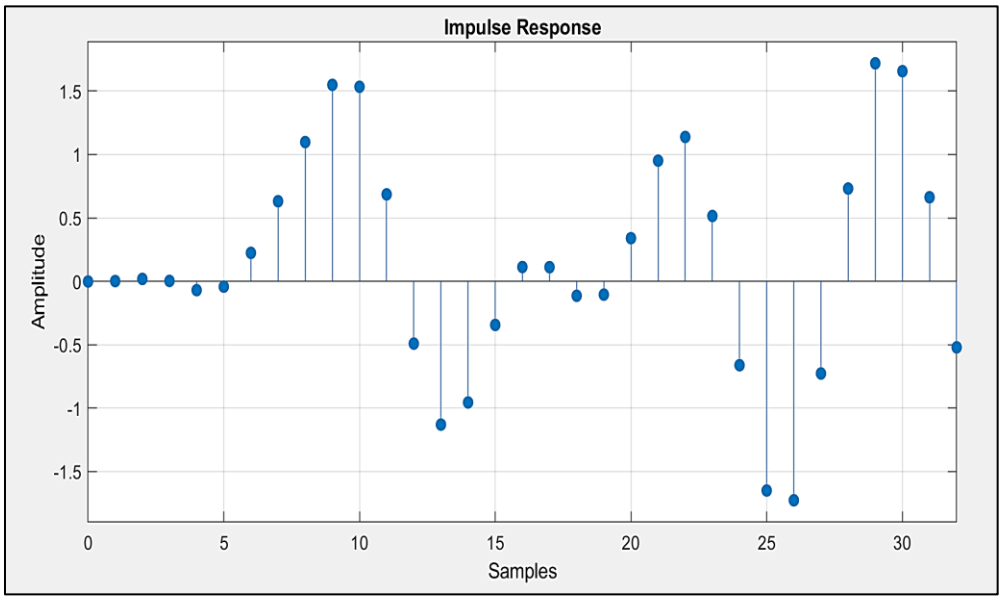



*Figure 3.17: Impulse response of the LPF filter*

As expected the two signals which have peaked at 2*kHz* and 3*kHz* were passed through the filter and the filter blocks/attenuates the signals which have peaked at 7*kHz* and 9*kHz*.

By comparing the impulse response (figure 3.15 and figure 3.17), it can be clearly seen that the signal with higher frequencies are attenuated.

The low pass filter is designed successfully in MATLAB.

## 3.8 Summary

Chapter 3 briefly describes about the filters. Sections 3.1 and 3.2 discuss the general filters, their design, and responses. FIR filters are discussed in detail with various windows available to design a filter. Kaiser window is chosen over other windows because of its advantages over other window functions which are discussed in 3.4.2.

A low pass filter is successfully designed in MATLAB with the project specifications. The coefficients of this filter were multiplied with the coefficients of Kaiser window function to generate a new filter to check/verify the magnitude and impulse response of the generated filter. The new filter generated was similar to the previous filter, thus stating that the coefficients are correct and are used later in the project in FPGA. The filter was again verified by giving different input signals (different frequencies), and the filter passed all the tests done on MATLAB.

# Chapter 4

# Asynchronous Processors

The first asynchronous processor was built in 1989 by Alain Martin, a professor at Caltech. It had a 16-bit data path and was capable of functioning as a normal(regular) processor except interrupts and exceptions [18][19]. Sequential processors which are not governed by a clock are termed as asynchronous processors. They are made up of asynchronous blocks/ circuits. Unlike synchronous processors, the communication between these blocks are not governed by a central/ global clock. They are governed by the blocks itself which detect whether a task is completed or not. In this way, each block/ task can take up as much time as it needs to perform its function. This is a huge advantage over the conventional processors (synchronous processors). In synchronous processors, each task must wait or start after a particular clock period which governs the whole system (by calculating the worst case and critical paths). Figure 4.1 shows the comparison between the two processors pipelining.

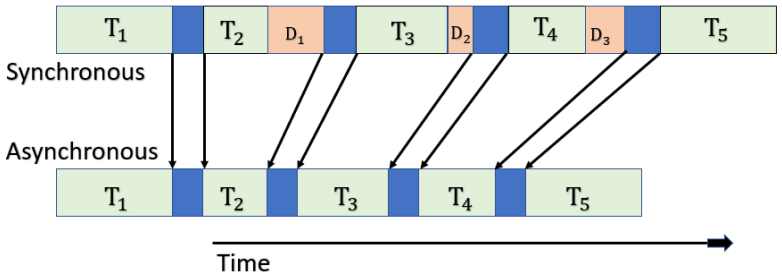



*Figure 4.1: Synchronous and asynchronous pipelining*

In figure 4.1, The synchronous processor has calculated the worst time for the tasks which is of T1 or T5 and set a clock period according to the maximum time taken by the tasks. The T2 task requires less computation and less time to perform its function. However, the problem arises when task T3 wants to compute, it has to wait for D1 time (delay) as the next clock cycle will occur after a whole period. Asynchronous processors overcome this waiting delay. It can be seen that in an asynchronous processor the tasks T1, T2, T3, T4 and T5 took their respective time to compute and are not governed by any clock and delays. It not only increases the performance of the system but also increases the efficiency.

The chapter briefly describes asynchronous processors, design, implementation and analogue delays (to implement FIR filters asynchronously on FPGA).

## 4.1 Advantages of asynchronous processors over synchronous processors

This project aims to design an FIR filter using an asynchronous processor because of its advantages over the synchronous processors, some of them are listed below-

- Higher performance
    - No central clock
    - No local delay
    - Process data at the rate of the environment (figure 4.1)
- Better power efficiency
    - Inactive components are in a standby state (no power utilisation)
    - Only active components utilise the power
    - No glitches, no power dissipation [18]
- Smaller chip size
- Less high-frequency electromagnetic interference (EMI) components [18]
- Circuit speed

Apart from the advantages, asynchronous processors have some disadvantages also. These include-

- More complex design
- Tough to troubleshoot the error
- The area might get doubled (addition of circuitry to check if the task is completed and to check it)
- Fewer people are trained to design/work on asynchronous processors
- Lack of electronic design automation (EDA) [20] tools available to design asynchronous circuits and processors

## 4.2 Asynchronous design styles

Asynchronous processors can be designed in two styles [20] [21]-

1) Bundle-data encoding
2) Dual-rail encoding


*Figure 4.2: Asynchronous design styles- a) Bundle-data. b) Dual-rail [20]*

### 4.2.1 Bundle-Data

Bundle-data is a method to encode in asynchronous circuit design, where one bit (either logic high or logic low) is represented by one wire only, and a separate wire represents the completion(acknowledge) of the data. Figure 4.2.a shows the general encoding structure of bundle-data. Bundle-data encoding is also termed as single-rail encoding.

One way to implement asynchronous circuits with bundled-data is shown in figure 4.2 [22]. The delay elements guarantee the timing of data operations. Therefore, the whole performance of the processor depends on the delay. The same data-path is used to represent the one bit (of the signal) and completion of the signal. The data path used in bundle-data is similar to synchronous design. This is one of the advantages which bundle-data has over dual-rail, where one bit of the signal is represented by two separate wires [20][22] and a completion wire to acknowledge the data. The power dissipation and the circuit are also smaller and lower than other implementations.

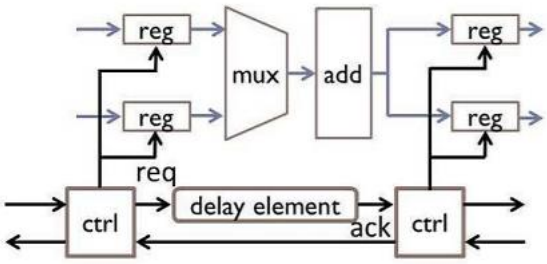



*Figure 4.3: Circuit structure of asynchronous circuits with [22]*

### 4.2.2 Dual-rail

Dual-rail is a method of encoding the data in asynchronous design, where one bit is represented by two wires and an acknowledgement wire which confirms the completion of the data [22]. One wire represents the logic high, and one wire represents logic low. The general structure of the dual-rail encoding structure is shown in figure 4.2.b. Dual-rail encoding design is also called as delay insensitive [20]. Timing information is embedded with the data bits, so individual delay variations do not affect much to its functioning.

More information about asynchronous processors, designs and circuit diagrams are summarised in [20] [18] [17].

## 4.3 Field programmable gate array

Field programmable gate array (FPGA) is a reconfigurable data-path architecture device [20]. It is widely used nowadays because of its advantage of lower design cost and flexibility to change the circuitry. FPGA's offer high throughput when area and power are taken into consideration.

FPGA that is used in this project is by Xilinx (Digilent Basys 3 Artix-7 board) [22]. The hardware descriptive language used in this FPGA is Verilog. This section aims to design a 16 tap FIR low-pass filter that was implemented on MATLAB in section 3.6.

### 4.3.1 FIR design in FPGA

Designing an filter requires four basic parts-

I) Coefficients
II) Multiplication
III) Addition
IV) Delay

**I Coefficients**

The coefficients obtained from MATLAB in section 3.6 and Table 3.4 are used in this section. These coefficients H(n) are converted to integers as FPGA requires the coefficients to be integers [3]. The approach used here is to multiply the double values with $2^x$ and then round off the product.

$$C_{k\ integer} = \text{round}( H_{D\ double} X\ 2^x ) \qquad \text{equation 4.1}$$

In this project x = 10, The following new filter coefficients are shown in table 4.1.

*Table 4.1 : MATLAB filter coefficients(H(n)) and revised filter coefficients($C_k$) calculated using equation 4.1*

| S.No. | MATLAB Filter Coefficients (H(n)) | New Filter Coefficients ($C_k$) | | S.No. | MATLAB Filter Coefficients (H(n)) | New Filter Coefficients ($C_k$) |
|---|---|---|---|---|---|---|
| 1 | -0.006 | -6 | | 9 | 0.411 | 421 |
| 2 | 0.003 | 3 | | 10 | 0.171 | 175 |
| 3 | 0.024 | 25 | | 11 | -0.041 | -42 |
| 4 | 0.003 | 3 | | 12 | -0.064 | -66 |
| 5 | -0.064 | -66 | | 13 | 0.003 | 3 |
| 6 | -0.041 | -42 | | 14 | 0.024 | 25 |
| 7 | 0.171 | 175 | | 15 | 0.003 | 3 |
| 8 | 0.411 | 421 | | 16 | -0.006 | -6 |

In table 4.1, the value of half of the coefficients are same i.e.

$C_k = C_{17-k}$ , $C_1 = C_{16}$ , $C_2 = C_{15}$ ..... $C_8 = C_9$ which means 16 coefficients can be simplified to 8 coefficients and similarly the number of multipliers can be reduced to 8.

**II Multiplication**

In Verilog "*" operator is for multiplication, i.e. Product (a*b) = a X b;

The coefficient $C_K$ is multiplied with the input signal and delayed input signal (refer to figure 3.5). Here input signal specified in the project is 3 bit, and coefficient $C_K$ is 10 bits (9 bits + 1 sign bit) so, the multiplier designed in this section is of 13 bits (12 bits + 1 sign bit).

It is clear from the table 4.1 that some of the coefficients are negative, and some of the coefficients are positive. Therefore, two separate tasks were created in the Verilog for multiplications (positive multiplication and negative multiplication).

Positive task (positive multiplication) code snippet is shown in listing 4.1.

```
                                        //Positive multiplication
task positive;
input [2:0] temp_input;                 //Temporary input
input [9:0] coeff;                      //Positive Coefficient
output [12:0] temp_output;              //Temporary output of multiplication
    begin
        temp_output = temp_input * coeff;  //Multiplication
    end
endtask
```

*Listing 4.1: Code snippet for positive multiplication*

Simulation results of the above snippet are in section 4.4.

The same multiplier(signed) is also designed using Baugh Wooley algorithm [23]. Refer to appendix A3 for the source code and simulations.

**III Adder**

In Verilog "+" operator is used for addition, i.e. Sum (a + b) = a + b;

The implementation of an adder is a cascaded full adder are placed in parallel to each other, and this algorithm is called a ripple carry adder [24]. A ripple carry adder is an adder in which the carry out (cout) of each adder is the carry in (cin) of the succeeding adder.

Code snippet for a full adder is shown in listing 4.2. Figure 4.4 shows a block diagram of a 4-bit ripple carry adder.

```
input a, b, cin;                                    //Input signal
output s, cout;                                     //Output signal
    assign s = a^b^cin;                             //Sum
    assign cout = (a&b) | (b&cin) | (a&cin);        //Carry
```

*Listing 4.2: Code snippet for full adder*

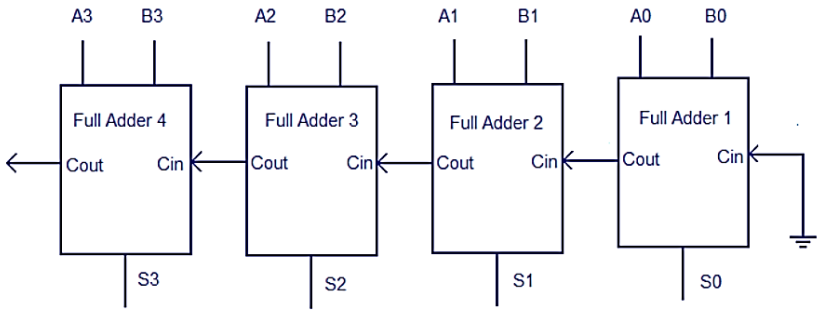



*Figure 4.4: 4-bit ripple carry adder [24]*

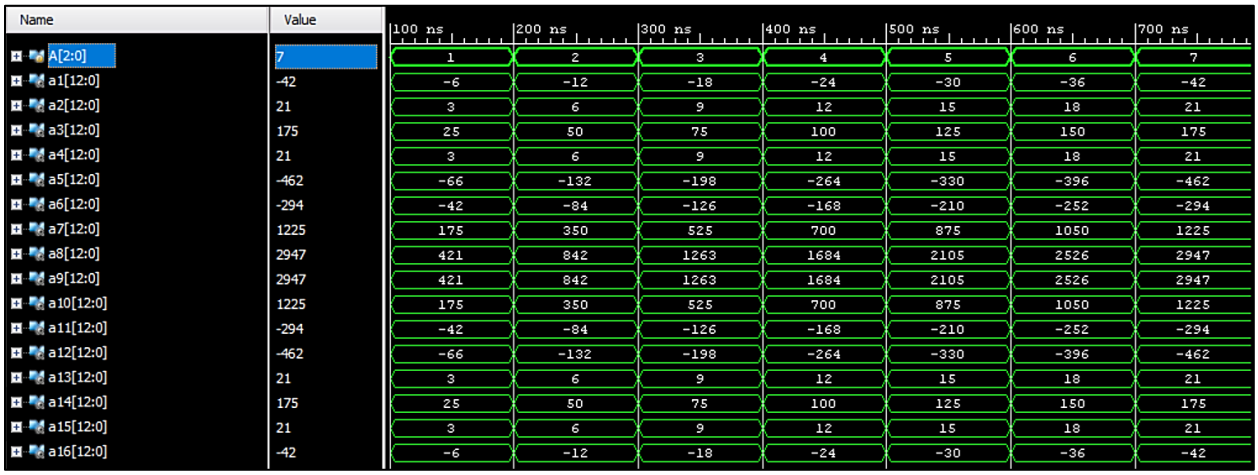
In this project several adders are used to add the products of the coefficients. Simulation results are in the next section 4.4.

**IV Delay**

The delay approach used in the project is asynchronous. The filter implemented on MATLAB and FPGA is a 16-tap filter (16 coefficients) so, a total of 16 asynchronous delay blocks are implemented on a breadboard using analogue components. The delay required is calculated by $1/f_s$ which is 50$\mu s$ ($f_s$ in the project is 20*kHz* (defined in section 3.6)). This section is discussed in detail in chapter 5.

## 4.4 Simulations in FPGA

The simulation result of multiplier with the coefficients is shown in figure 4.5.

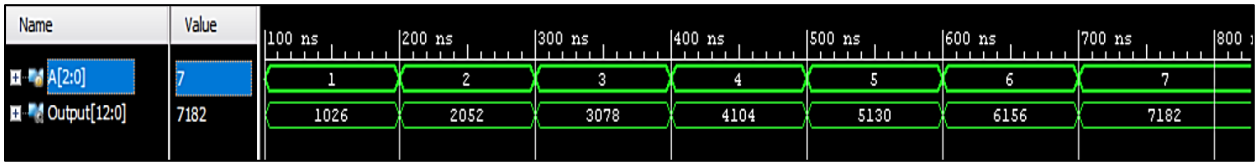


| Name | Value | 100 ns | 200 ns | 300 ns | 400 ns | 500 ns | 600 ns | 700 ns |
|---|---|---|---|---|---|---|---|---|
| A[2:0] | 7 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| a1[12:0] | -42 | -6 | -12 | -18 | -24 | -30 | -36 | -42 |
| a2[12:0] | 21 | 3 | 6 | 9 | 12 | 15 | 18 | 21 |
| a3[12:0] | 175 | 25 | 50 | 75 | 100 | 125 | 150 | 175 |
| a4[12:0] | 21 | 3 | 6 | 9 | 12 | 15 | 18 | 21 |
| a5[12:0] | -462 | -66 | -132 | -198 | -264 | -330 | -396 | -462 |
| a6[12:0] | -294 | -42 | -84 | -126 | -168 | -210 | -252 | -294 |
| a7[12:0] | 1225 | 175 | 350 | 525 | 700 | 875 | 1050 | 1225 |
| a8[12:0] | 2947 | 421 | 842 | 1263 | 1684 | 2105 | 2526 | 2947 |
| a9[12:0] | 2947 | 421 | 842 | 1263 | 1684 | 2105 | 2526 | 2947 |
| a10[12:0] | 1225 | 175 | 350 | 525 | 700 | 875 | 1050 | 1225 |
| a11[12:0] | -294 | -42 | -84 | -126 | -168 | -210 | -252 | -294 |
| a12[12:0] | -462 | -66 | -132 | -198 | -264 | -330 | -396 | -462 |
| a13[12:0] | 21 | 3 | 6 | 9 | 12 | 15 | 18 | 21 |
| a14[12:0] | 175 | 25 | 50 | 75 | 100 | 125 | 150 | 175 |
| a15[12:0] | 21 | 3 | 6 | 9 | 12 | 15 | 18 | 21 |
| a16[12:0] | -42 | -6 | -12 | -18 | -24 | -30 | -36 | -42 |

*Figure 4.5: Simulation result of multiplier with coefficients*

Figure 4.5 shows the simulation results of the multiplier with the coefficients. The output of the multiplication is correct (refer table 4.2).

Figure 4.6 shows the simulation result of addition of products generated by multiplier in the above figure 4.5.



| Name | Value | 100 ns | 200 ns | 300 ns | 400 ns | 500 ns | 600 ns | 700 ns | 800 |
|---|---|---|---|---|---|---|---|---|---|
| A[2:0] | 7 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
| Output[12:0] | 7182 | 1026 | 2052 | 3078 | 4104 | 5130 | 6156 | 7182 | |

*Figure 4.6: The output of addition of products from multiplier*

Table 4.2 verifies the output generated by FPGA with the manual calculations of multiplication and addition.

*Table 4.2 : Manual calculations for multiplication (input by coefficients) and addition of products*

| **Coefficients $C_k$** | **Input** | | | |
|---|---|---|---|---|
| | **1** | **2** | **3** | **7** |
| -6 | -6 | -12 | -18 | -42 |
| 3 | 3 | 6 | 9 | 21 |
| 25 | 25 | 50 | 75 | 175 |
| 3 | 3 | 6 | 9 | 21 |
| -66 | -66 | -132 | -198 | -462 |
| -42 | -42 | -84 | -126 | -294 |
| 175 | 175 | 350 | 525 | 1225 |
| 421 | 421 | 842 | 1263 | 2947 |
| 421 | 421 | 842 | 1263 | 2947 |
| 175 | 175 | 350 | 525 | 1225 |
| -42 | -42 | -84 | -126 | -294 |
| -66 | -66 | -132 | -198 | -462 |
| 3 | 3 | 6 | 9 | 21 |
| 25 | 25 | 50 | 75 | 175 |
| 3 | 3 | 6 | 9 | 21 |
| -6 | -6 | -12 | -18 | -42 |
| **OUTPUT** | **1026** | **2052** | **3078** | **7182** |

The following table 4.2 confirms that the logic and implementation of multiplication and addition designed in FPGA is correct and working.

## 4.5 Summary

The chapter describes asynchronous processors. Advantages of using asynchronous processors over synchronous processors are explained. It briefs about the possible design encoding schemes present for asynchronous processors and how it can be used on FPGA's. Three out of four requirements to design an FIR filter are achieved in this chapter. The fourth requirement which is delays is designed and implemented in the next chapter- Analogue delay line.

# Chapter 5

# Analogue delay line

An analogue delay line is a cascaded network of electrical components, where a time difference (delay) or a phase change between the output signal and the input signal is created by each component. In 1920's Francis Hubbard made the first analogue delay line with a network of an inductor-capacitor ladder and named it as an artificial transmission line [22]. Later coaxial cables, piezoelectric transducers and quartz glass were used to generate a delay from 10 to 10000 milliseconds delay [22]. A cascaded series of resistor-capacitor (RC circuits [23]) are also used to generate a significant delay.

Analogue delay lines have their advantages when used with signal processing. For example, an oscilloscope uses an analogue delay line to allow the wave observation just before it triggers the event [22]. The phase alternating line (PAL)- the colour encoding system used in televisions uses an analogue delay line for scan lining [22]the video. For quite a long time, rotating magnetic drums were used as delays other than storing the data [23].

The chapter briefly describes analogue delays, designing, simulations and circuit hardware implementation on breadboard [25].

## 5.1 Delay techniques

### 5.1.1 Bucket bridge devices

The bucket bridge devices often called as BBD was invented around 1968-69 by F. Sangster and K. Tee [22]. BBDs got their unique name because of similarity in their functionality with a bunch of people standing in a line, passing buckets of water to fight a fire. BBDs were the first delay line of a kind which consists of a series of capacitors and Mosfet/ transistors to carry analogue signals. Mosfet are used as a switch to pass the charge in the capacitors to the next stage using a clock. Using this technique, the delayed output signal is discrete in time [22]. A simple circuit diagram of a 5 stage BBD is shown in figure 5.1.

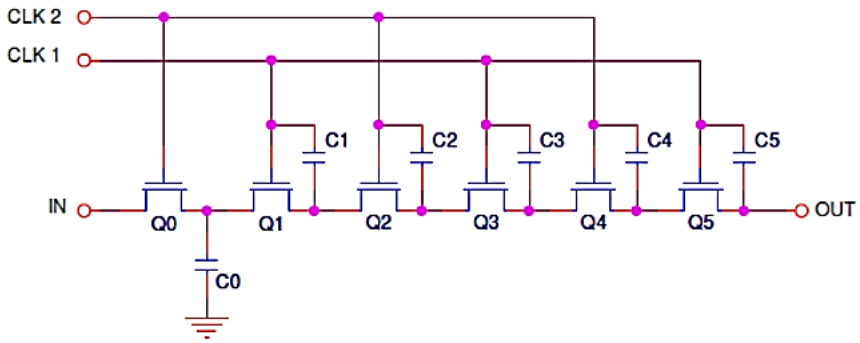



*Figure 5.1: A simple BBD delay line governed by external clock*

Mosfet and capacitor together act as a sample-and-hold circuit. So, when CLK 2 is high, the input is gated through Q0 to C0 capacitor. At the next clock, C0 is isolated, and the last voltage applied at C0 is stored. As CLK 1 goes high, the charge stored in C0 is gated to the capacitor C1. So, the process goes on till the last sample-and-hold-circuit.

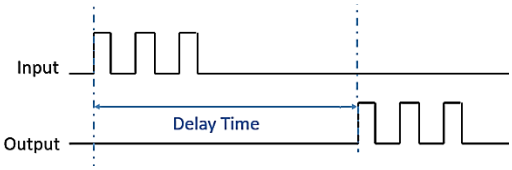



*Figure 5.2: Input and output signal when passed through the above circuit figure 5.1*

More information, circuit diagrams and better explanations about BBDs are given in [25] and [26].

### 5.1.2 Delay using All pass filter

Another approach to generate a delay is using all pass filters. All pass filter (APF) is a type of filter which allows all the frequencies to pass through it without changing the gain of the input. ALF's are generally used in musical instruments as a phaser to mix the output signal with the input signal [27]. Schematic of an APF is shown in figure 5.3.

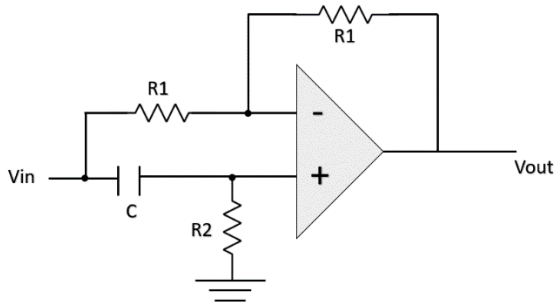



*Figure 5.3: Schematic of an All pass filter [28].*

The primary purpose of APF is to add a delay (phase shift) to the response of the circuit. The phase change determines how much delay will an input signal experience by passing through this circuit. Resistor(R2) and capacitor(C) allow the adjustment of the delay at a particular frequency [28]. R1 resistors act as a voltage divider in the circuit (refer section 2.4.3 for voltage divider working). The schematic shown in the figure 5.3 is a first order all pass filter with a unity gain.

The transfer function of APF is given by equation 5.1

$$H(s) = \frac{s - \frac{1}{R2C}}{s + \frac{1}{R2C}} \qquad \text{equation 5.1 [27]}$$

Magnitude of the filter is given by $|H(i\omega)| = 1$ equation 5.3 [24]

Phase of the filter is given by $\angle H(i\omega) = \arctan(-\omega R2C) - \arctan(\omega R2C)$ equation 5.3 [24]

APF as a pure delay function is given by Laplace transform-

$$e^{-sT} \qquad \text{equation 5.4 [24]}$$

where *T* is the delay and *s* is complex frequency (iω). Equation 5.4 can be approximated using Padé approximant[24]

$$e^{-sT} = \frac{e^{-sT/2}}{e^{sT/2}} \approx \frac{1 - sT/2}{1 + sT/2} \qquad \text{equation 5.5 [24]}$$

Therefore, by setting R2C = T/2 the transfer function H(s) can be recovered.

### 5.1.3 Other methods to generate delay

Analogue delays can be implemented by using other techniques as well such as cascading CMOS inverters one after the other in series. Figure 5.4 shows the general structure of this technique and it is well defined in [2]. Another method to generate a delay is using RC network and an amplifier as a comparator. In this technique the RC determines the delay and comparator compares the input signal from the RC circuit with a reference voltage and generates the output accordingly. This technique is explained in detail in [29]. The simulations and circuit diagram for this technique is shown in section 5.2.

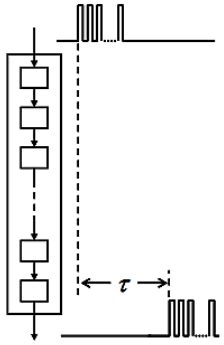



*Figure 5.4: Cascaded CMOS to generate a delay [2] .*

The delay in the project is implemented using APF rather than BBD's because BBD's use an external clock to function the delays.

## 5.2 Circuit level simulation

Delay generation using amplifier as a comparator

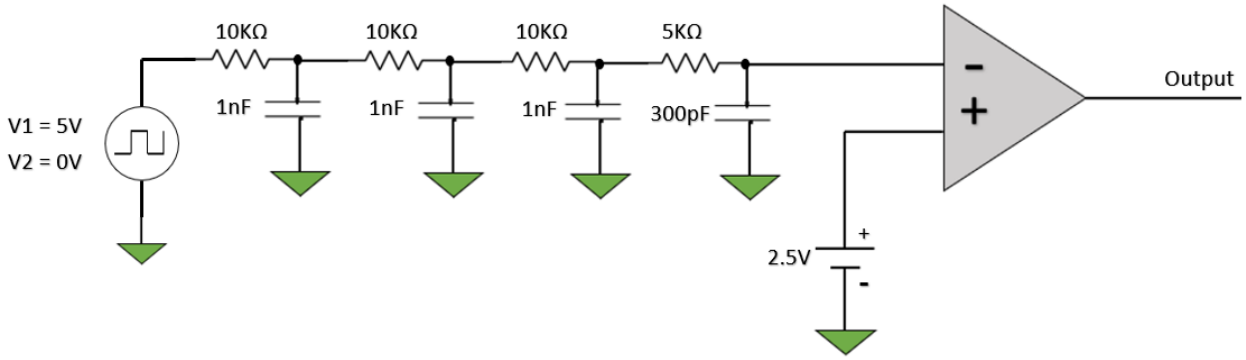



*Figure 5.5: Schematic of an amplifier used as a comparator to generate a delay*

The positive voltage (+Vdd) of the amplifier is +5$V$, and the negative voltage (Vss) of the amplifier is connected to the ground. RC network chosen in the above schematic is such that there will be enough slew rate to get a delay 50$\mu s$ (refer section 4.3.1.IV). Amplifier works as a comparator, whenever the input voltage (V1 or V2) is greater than the Vref (reference voltage 2.5$V$) the output of the comparator will be +Vdd (5$V$) else the output of the comparator will be Vss (0$V$). The input signal is Vpulse (square wave), and the delayed output signal is also a square wave. Refer to appendix A1 for a clear schematic of figure 5.5.

Figure 5.6 shows the simulation of the above circuit.

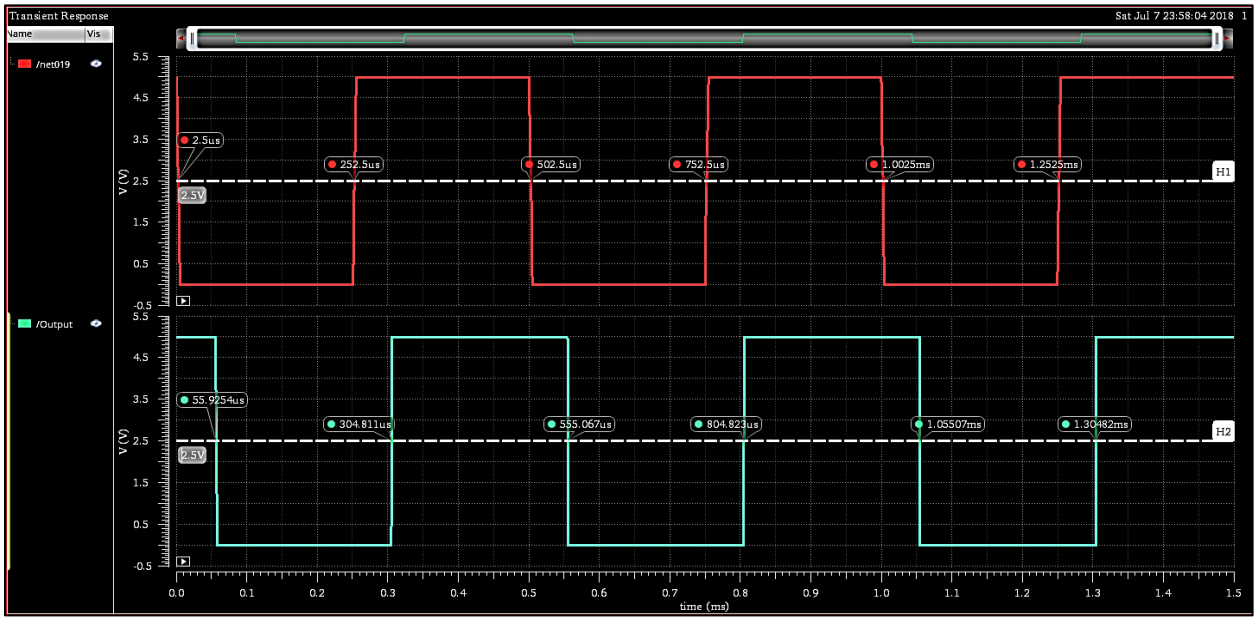



*Figure 5.6: Simulation of the schematic shown in figure 5.5*

The simulation results show that there is a delay of 52$\mu s$. A perfect delay (50$\mu s$) can be achieved by changing the value of R and C.

The same RC series circuit (along with one more RC pair (1$k\Omega$ resistance and 100$pF$ capacitance)) was tested with the comparator designed in Verilog-a in section 2.4.2. The simulation results shown in figure 5.7 confirms that the delay between the input and the output signal is 55$\mu s$. Refer to the appendix A1 for the schematic diagram.

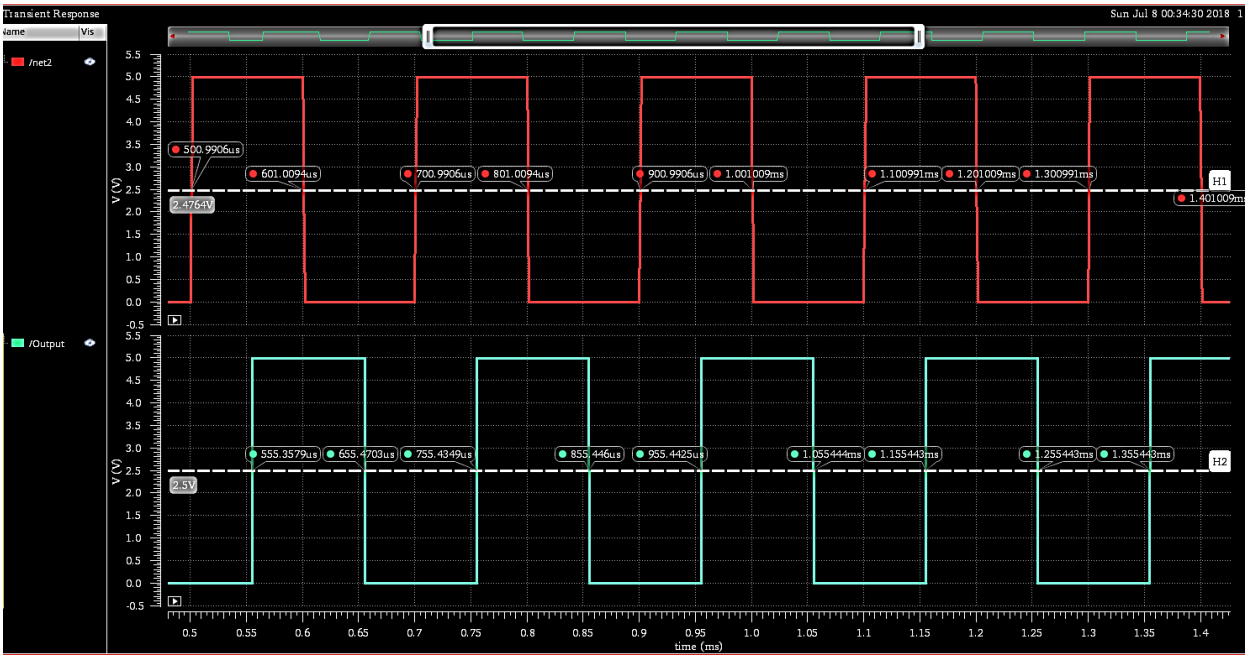



*Figure 5.7: Delay simulation of the comparator circuit designed in section 2.4.2.*

## 5.3 Hardware implementation

An all pass filter is implemented on a breadboard [30]. APF is used instead of the circuit designed in simulations because of the number of components used by APF are half than that of the circuit shown in figure 5.5.

The values of the resistors, capacitors used to implement 16 delay blocks are, R1 = 10$k\Omega$, R2 = 3$k\Omega$ and C= 10$nF$. Amplifier used in the project are LM741 [31] and LM348M [32]. NAND Gate IC CD4011BE [33] is also used in the project to make a phase shift of 180° so that the delay can be calculated clearly and to get a sharper square wave output without any slewing of the signal. The schematic of the circuit will be the same as shown in figure 5.3.

From equation 5.5 RC or R2C = T/2 where T = 50$\mu s$.

Therefore, R2 is chosen to be 3$K$ and C is chosen to be 10$nF$.

3K X 10n = 30$\mu s$. (R2 is chosen 3k$\Omega$ to deal with stray losses of the breadboard)

Input signal given to the APF was a square wave from the signal generator at 2$kHz$ frequency. The hardware implementation is shown in figure 5.8.

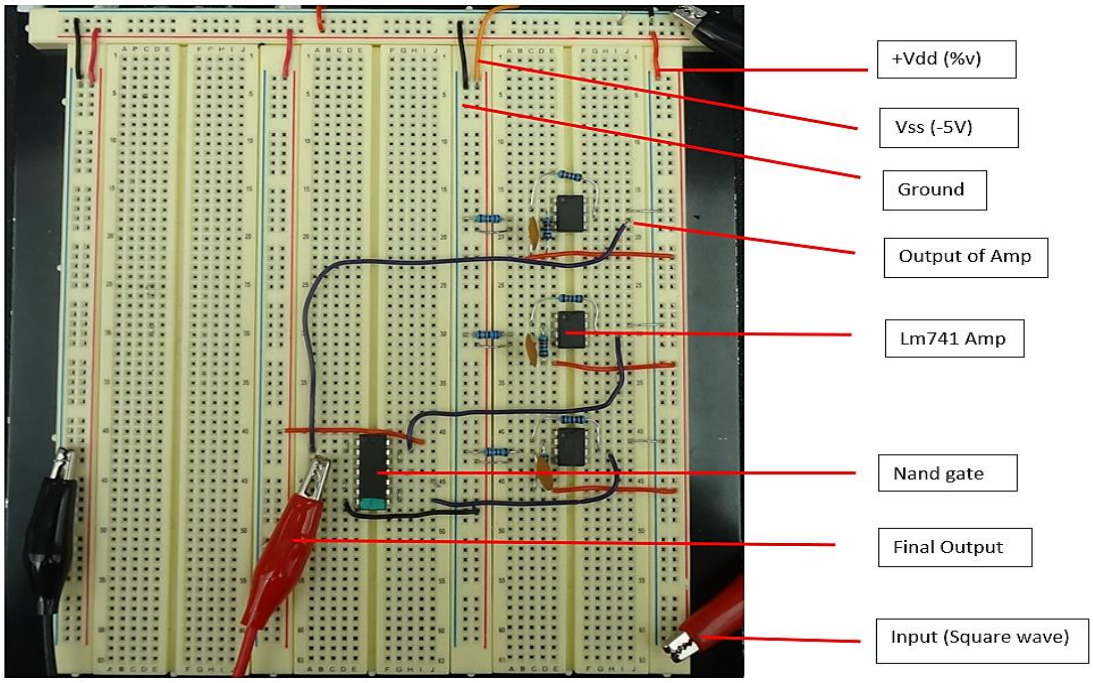



*Figure 5.8: Hardware implementation of 1 delay block on breadboard.*

16 such cascaded delay blocks are made on two breadboards where the output of 1st delay block is the input of the 2nd delay block and so on till the 16th delay block. Refer to appendix A4 for the overall hardware implementation (circuit implementation on breadboard) of the whole delay block. The output when checked on oscilloscope confirms a delay of 50$\mu s$. This is shown in figure 5.9.

The expected result was 60$\mu s$ but because of the losses explained in 5.3.1 The output is 50$\mu s$.

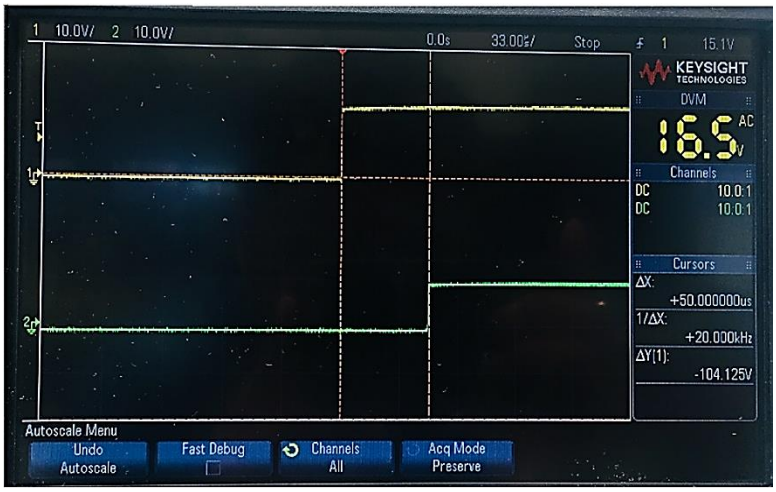



*Figure 5.9 : Output of the figure5.8 on oscilloscope*

### 5.3.1 Analogue delay implementation with FPGA

The analogue delay is successfully implemented on breadboard using a signal generator. Now the delay block should show the same results when implemented using FPGA. For this a test counter is made in Verilog at 2*khz* frequency and is given as an input to the analogue line. The output of delay block (figure 5.10) when viewed on the oscilloscope shows the delay to be 53.4$\mu s$.

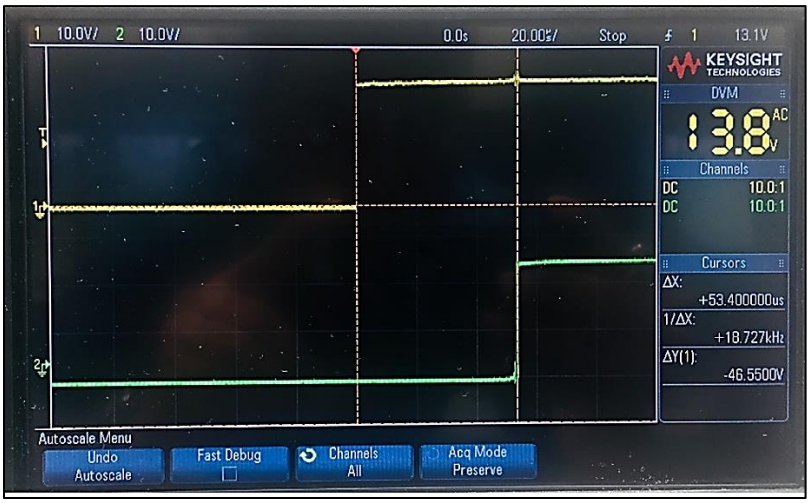



*Figure 5.10: Delay block output when the input is given from FPGA*

In figure 5.10, the upper signal (yellow) is the input signal from FPGA. The signal which is green in colour is the output signal generated after the 1st delay block. Simulation results of 16th delay block is shown in appendix A4.

The delay after 16th delay block (when the input is from signal generator) is 1.16*ms*. And the delay after 16th delay block (when the input signal is from FPGA) is 1.18*ms*. The possible reasons of the slight change in the delay are due to these reasons-

The whole delay blocks (16 delay blocks) are made on three different breadboards.

- A breadboard is not 100% efficient/ correct as there are many internal stray capacitances.
- Long wires are used to connect the output of every delay block to an Input-output (IO) block which will have their own resistances. Plus, the resistance of the breadboard.
- The external and internal noise of the whole system.

- The stray capacitance from the jumper wires connected from FPGA.
- The components connected on the breadboard are placed close to each other and the overhead wires used to connect the system cause parasitic capacitances which affect the performance of the circuit.

The delay after the $16^{th}$ block is greater than the expected delay. It is not bad for the system and will not affect the working of the system as such. The delay should not be less than T (time period of the sampling wave which is $50\mu s$. If the delay is less than $50\mu s$ then the system will not read the signal correctly and will generate the wrong output.

## 5.4 Summary

Chapter 5 briefly describes about different techniques to achieve analogue delays. Bucket bridge devices are used at most of the places (mixing the input signal with delayed output signal in tuning guitar [25]) but it requires the use of the external clock.

All pass filter can be used to generate a delay by changing the phase of the signal and adjusting the values of Resistor and Capacitor. The delay circuit was first designed and simulated with an amplifier as a comparator to generate a significant delay. The designed all pass filter circuit was then implemented on breadboard and tested successfully with the FPGA.

# Chapter 6

# Digital to analogue conversion

Digital to analogue conversion (DAC) is a process to convert the digital code back to analogue signals. DAC is the reverse process of analogue to digital conversion (ADC). DAC is used in televisions and mobile phones to convert the digital video codes (data) to analogue video signals which are later displayed on the screen [34] . The general structure of a DAC is shown in figure 6.1.

This chapter describes the designing and hardware implementations of digital to analogue converters and an event to analogue converters (EAC).

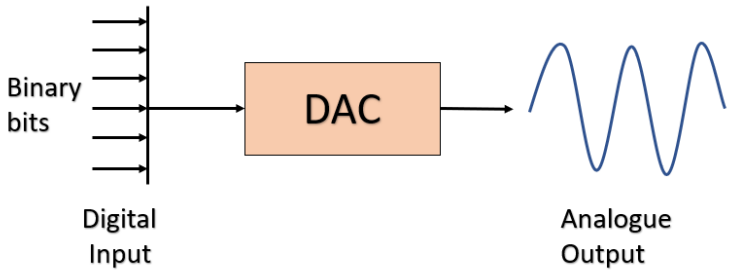



*Figure 6.1: General structure of a digital-to-analogue converter*

## 6.1 Types of digital to analogue converter

There are many types of DAC's available in the market some of them are listed below-

1) Using summing amplifier DAC
2) Using R-2R ladder DAC
3) Thermometer coded DAC
4) Operational amplifier integrator

In the project Operational amplifier are used as an integrator to convert the digital signals to analogue signals. For more detail about other techniques refer [34] [35].

## 6.2 Operation amplifier integrator

An operational amplifier integrator is an integrating circuit which integrates the input voltage with respect to time. It produces the output voltage which is proportional to the integral of the input voltage [36]. The circuit consists of an amplifier, a capacitor and a resistor. The schematic of an op-amp integrator is shown in figure 6.2.

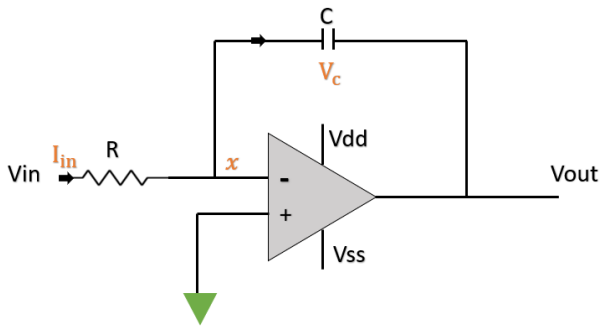



*Figure 6.2: Operational amplifier integrator [29]*

If a square wave is given as an input to the integrator, then the capacitor will charge and discharge with the changing input. This will generate a sawtooth [36] waveform. The output is affected by RC time constant ($\tau$). This type of circuit is also called as ramp generator [36] which is shown in figure 6.3.

The voltage across the capacitance is given by

$$V_c = Q/C \qquad \text{equation 6.1}$$

Where $V_c$ is voltage across capacitor, Q is charge held by the capacitor and C is the capacitance.

$V_c = V_x - Vout \quad => \quad 0 - Vout$

Therefore, $\frac{dVout}{dt} = \frac{dQ}{Cdt} = \frac{1}{C}\frac{dQ}{dt}$ equation 6.2

Since x = 0 in the starting then $I_{in}$ will be given by

$$I_{in} = \frac{Vin-0}{R} = \frac{Vin}{R}$$ equation 6.3

Let current flowing through feedback C be $I_f$ then

$$I_f = C\frac{dVout}{dt} = C\frac{dQ}{dt} = \frac{dVout.C}{dt}$$ equation 6.4

The ideal voltage output is given by

$$Vout = \frac{-1}{RC}\int_0^t Vin\ dt$$ equation 6.5 [36]

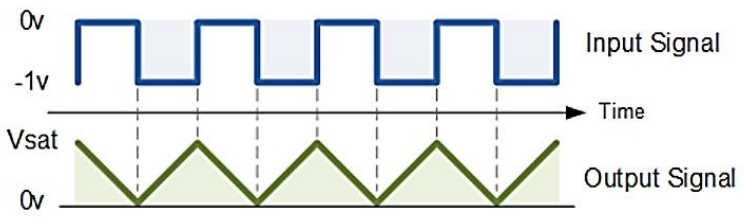



*Figure 6.3: Output of an integrator when a square wave is given an input [29].*

## 6.3 Hardware implementation

The operational amplifier as an integrator is made on breadboard. The value of R and C chosen is 5$k\Omega$ and 10$nF$.

Therefore, RC = 50$\mu s$

The output (ramp signal) generated by the integrator is passed through a low pass filter to get a sinusoidal wave back. LPF is designed using R and C in series with the same resistance and capacitance. The implementation of the circuit is shown in figure 6.4.

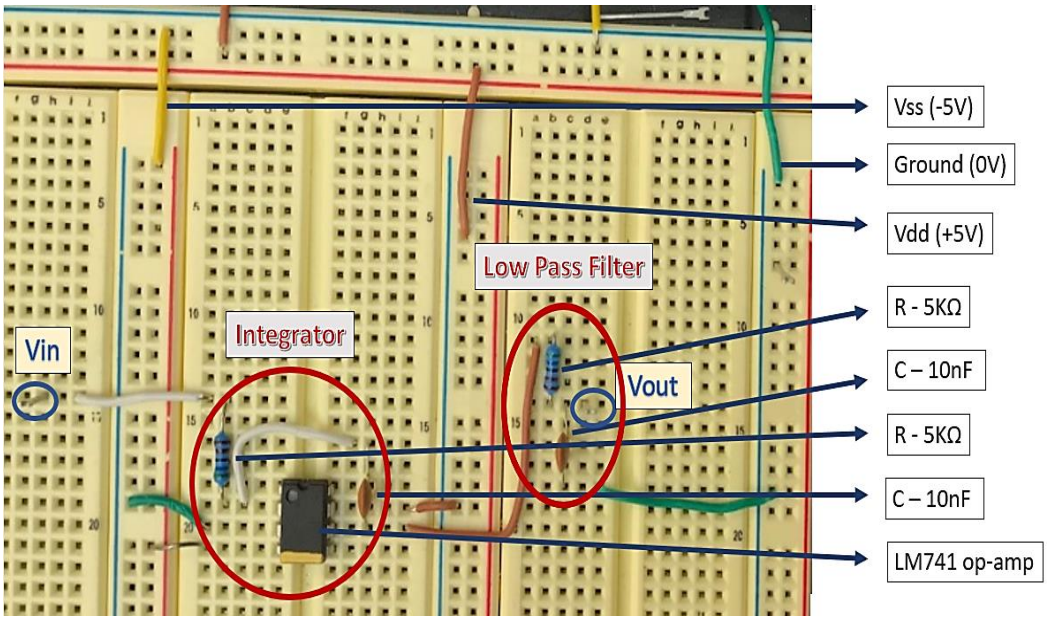



*Figure 6.4: Integrator circuit with a low pass filter.*

The input (Vin) is given a square wave and the output is observed on oscilloscope which is shown in figure 6.5.


*Figure 6.5: Output of the circuit implemented on breadboard shown in figure 6.4*

There are some distortions in the input signal from the signal generator that is why the output signal (sinusoidal wave) is a little bit distorted in between, otherwise, the integrator and the low pass filter are working perfectly fine.

A digital to analogue converter is successfully made.

## 6.4 Event to analogue conversion

Event to analogue conversion (EAC) is a process to convert events back to analogue signal. Refer to section 7.1 for simulation results of digital-to-event.

The UP and DOWN signals (events) generated from the FPGA are of same amplitude which is 3.3*V* shown in figure 6.6. These amplitudes are then converted to a new continuous-time-mod delta representation [4] ,shown in figure 6.7.

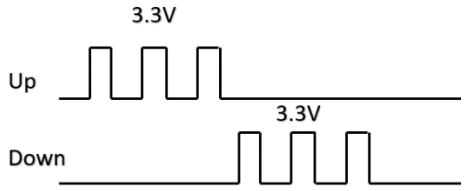



*Figure 6.6: Up and down signals generated from FPGA.*

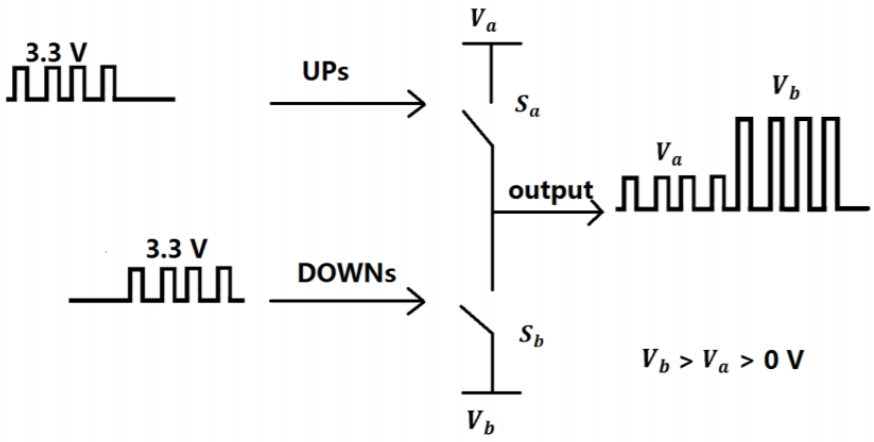



*Figure 6.7: new continuous-time-mod delta representation of up and down signals [4].*

In figure 6.7 two analogue switches are used to generate a new continuous time-mod data. Whenever an UP event (signal) occurs, switch $S_a$ will be closed, and voltage $V_a$ will pass through the circuit to generate the signal of $V_a$ volts which will be equal to the width of the up event. Whenever a DOWN event will occur, switch $S_b$ will be closed, and voltage $V_b$ will pass through the circuit to generate the signal of $V_b$ volts which will be equal to the width of the down event. When no event occurs (no UP or DOWN event), both the switches will be open, and no output will be generated.

The hardware implementation of this circuit along with the successful results are shown by Junfei Mao in [4].

## 6.5 Summary

This chapter discusses about digital to analogue conversion. Types of DAC were discussed. The operational amplifier as integration is studied in detail along its implementation. Integrator was successfully made on the breadboard and tested with a square wave to get a sinusoidal wave as output.

Event to analogue converters was studied briefly. As the amplitude (voltage) of events (UP and DOWN) is the same 3.3*V* (voltage from FPGA), it will be hard to distinguish between the two events. To overcome this problem, events were then represented by a new time-delta-mod scheme. These new representations of signals are then converted to analogue signals[3].

# Chapter 7

## System level simulation

The overall block diagram of the system is presented in figure 7.1.

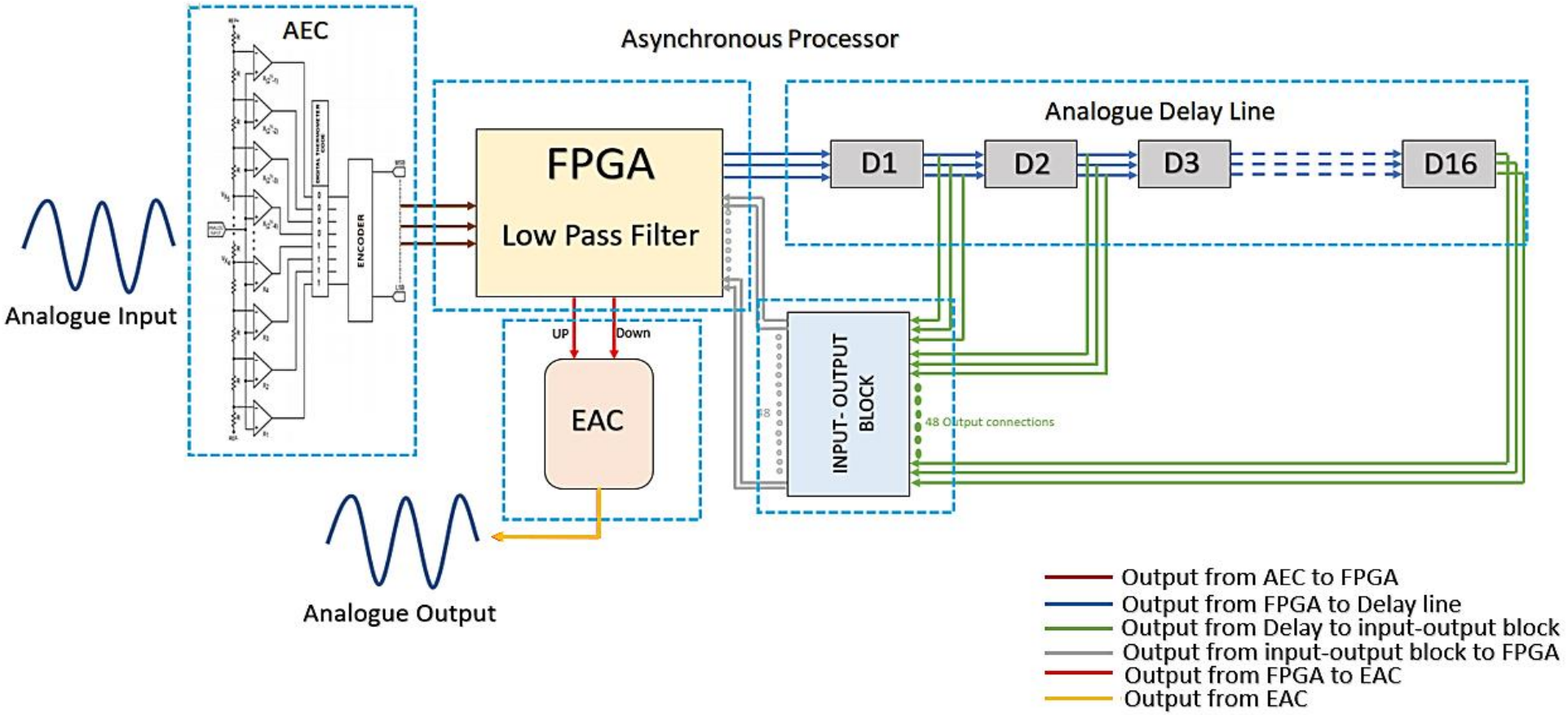



*Figure 7.1: Block diagram of the whole system*

In the above block diagram, the analogue signal is first converted to a digital signal using asynchronous flash analogue-to-digital converter (ADC). The output from the analogue to digital converter is a 3-bit digital code signal which is again converted to UP-DOWN events that is explained in detail in section 2.5.

Inside the field programmable gate array (FPGA) (Basys 3 Digilent board used in this project), these events (UP-DOWN) are then processed through an asynchronous event-to-digital converter, the simulation and process are explained in next section of this chapter.

After the signal is converted successfully to its digital state (3-bit), it is processed through a low pass FIR filter designed in section 3.6 and 4.3. This low pass filter is implemented using the coefficients generated using MATLAB. These coefficients are then multiplied with the delayed input signal.

The 3-bit signal from FPGA is passed through the cascaded delay blocks. The 3-bit input to the delay block is delayed using the technique explained in chapter 5. The output of every delay block (after getting delayed by 50$\mu s$) is then connected to an input-output block which is also designed on the breadboard. There is a total of 48 output signals from the 16 delay blocks.

The input-output block receives a total of 16 set of 3-bit signals, at different amount of delays. These signals are then connected back to FPGA for asynchronous processing. The output of the low pass filter is then converted to an event (UP, DOWN signals) by comparing the output with the delayed output. Simulation of this is also shown in next section of this chapter. These events are then transferred to an event-to-analogue converter.

Event-to-analogue converter (EAC) receives 2-bits of signals – UP and DOWN signal. These signals are then converted back to an analogue signal which is explained in chapter 6 and [3].

## 7.1 Digitisation

### 7.1.1 Event-to-digital conversion

The events received from the event-to-analogue converter(EAC) are processed through an asynchronous converter designed in FPGA. The aim of the processor is to decode the UP-DOWN events to a meaningful digital code.

Code snippet shown in listing 7.1 shows the conversion technique used in the project.

```
reg [2:0] Signal = 0;

    always @*                       //Asynchronous conversion block
    begin
        if (Up_inl == 1'b1)         //Checking UP Signal
             Signal = Signal + 1;   //logic
        else
        if (Down_inl ==1'b1)        //Checking DOWN signal
            Signal = Signal - 1;
    end
        assign Signal_in = Signal;  //Assigning the output
```

*Listing 7.1: cone snippet for UP-DOWN to digital code converter.*

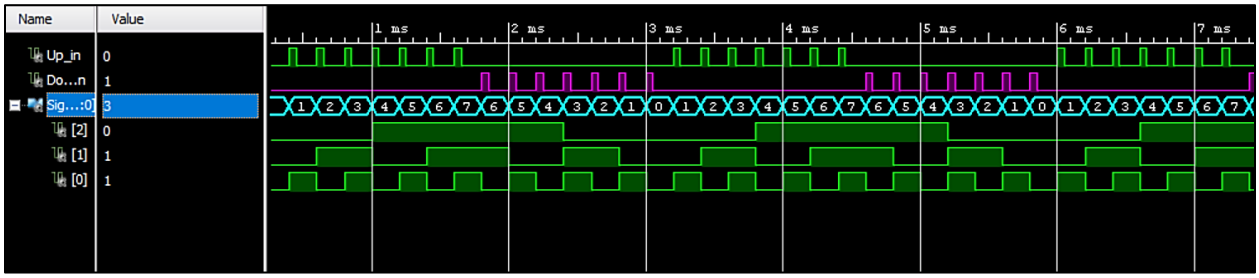



*Figure 7.2: UP-DOWN events converted to digital signals.*

After The successful conversion to digital code, this digital signal is then processed by asynchronous processor.

### 7.1.2 Digital-to-event conversion

Digital-to-event conversion(DEC) is opposite to event-to-digital conversion(EDC) where a digital output after the FIR filter is converted back to events (UP, DOWN signals). These events are then processed by event-to-analogue converter(EAC) to generate an analogue output which is explained in section 6.4.

DEC is done by comparing the output by FIR filter with a delayed output of the same filter. To do the DEC part a comparator module is designed which is synchronous.

Code snippet and simulation for this part is shown in listing 7.2 and figure 7.3.

```
always@ ( Signal)                  //Signal = Output of FIR filter
begin
DelayedOutput = #50000 Signal;     //Logic
end

always @*
begin
if ( Signal > DelayedOutput)       // if Matlab output is greater than delayed output
upl = 1'b1;                        // Assign UP =1
else
if ( Signal < DelayedOutput)       // if MATLAB output is smaller than delayed output
downl = 1'b1;                      // Assign DOWN =1
else
begin                              //else Assign both as 0
upl = 1'b0;
downl = 1'b0;
end
end
```

*Listing 7.2: Conversion of digital code-to-events.*

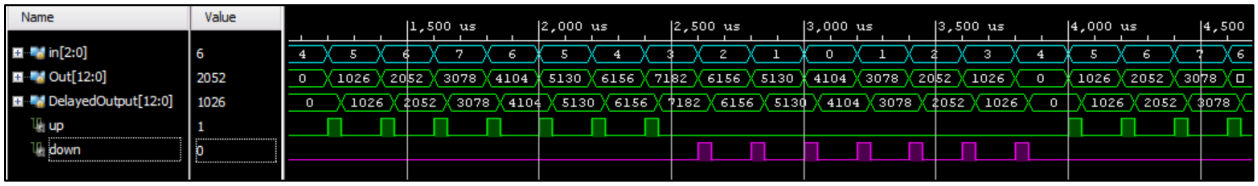



*Figure 7.3: Simulation of code listing 7.2.*

In the above figure 7.3, Out is the output of the FIR filter and DelayedOutput is the delayed output of the same FIR filter, delayed by 50$\mu s$. After the comparison UP-DOWN signals are generated which are represented in the above figure. Events are successfully generated.

The aim of project was to design an asynchronous processor in FPGA with analogue delays. The following figures shows the simulations of the FIR low pass filter output generated from FPGA and tested on MATLAB.

## 7.2 Simulations on FPGA

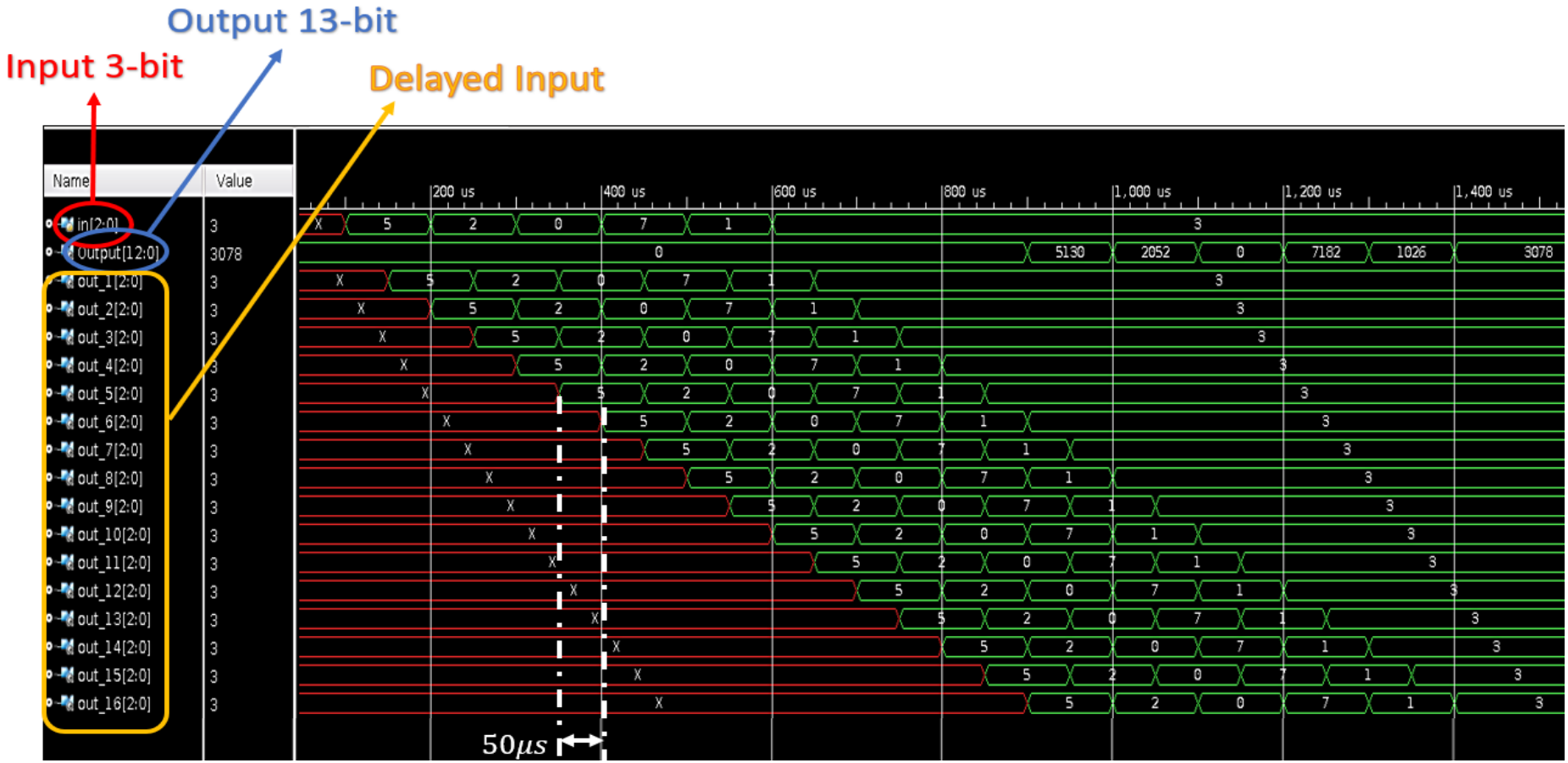



*Figure 7.4: Simulation output of delay.*

Figure 7.4 shows the simulation results of FPGA with delay. Due to the limitation of FPGA (discussed in next section), this is the simulation generated by Vivado-synthesis tool and not the actual delay line. But the actual delay line will work exactly like this as the output shown above. Refer to figure 5.10 which shows the delay of $\sim 50\mu s$.

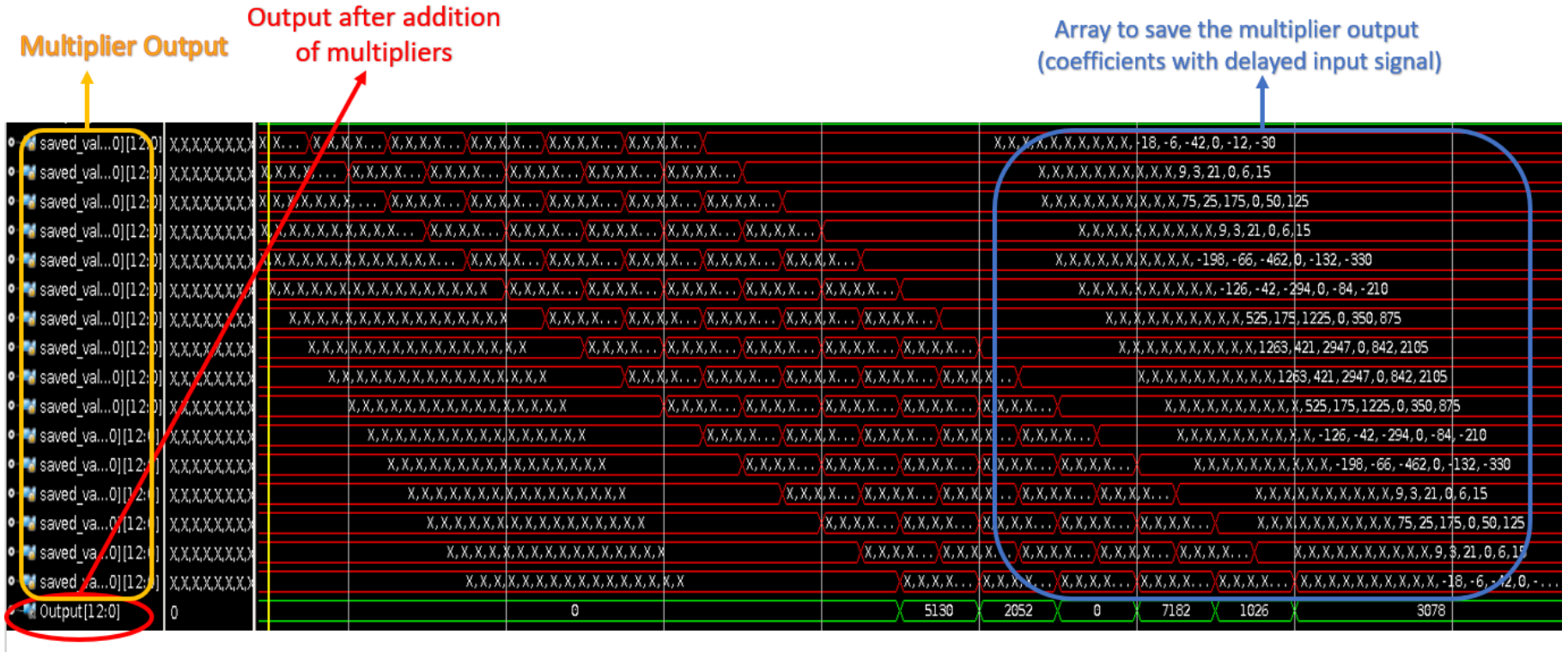



*Figure 7.5: Simulation of multiplication of coefficients of low pass filter with input and delayed input signal stored in an array.*

In figure 7.5, the multiplier output is saved in an array of Verilog registers of the same width as that of the output signal. saved_value_0, saved_value_1, …. saved_value_15 are the 16 Verilog registers used to store the multiplication output with changing inputs (delayed input signals). The index of these registers is incremented after every time a value is stored and later used for addition (adding the same array index) to generate output.

The code snippet is presented in listing 7.3.

```
                                                    //Time Stamping starts here
always @(out_1)
    begin
        negative (out_1, c0, semi_product[0]);          //Negative multiplication
        saved_value_0[integer(s0)] = semi_product[0];   //Storing value in array
        s0 = s0 + 1;                                    //incrementing array index
    end

always @(out_2)
    begin
        positive (out_2, c1, semi_product[1]);          //Positive multiplication
        saved_value_1[integer(s1)] = semi_product[1];   //Storing value in array
        s1 = s1 + 1;                                    //incrementing array index
    end
```

*Listing 7.3: Code snippet for storing the multiplication output in arrays.*

## 7.3 Simulations of FPGA output on MATLAB

### 7.3.1 Input signal is oscillating between 3'b001 and 3'b010 at 2.5KHz

18 inputs were given to FPGA at 2.5*kHz* frequency. The output of the input signal is presented in figure 7.6.

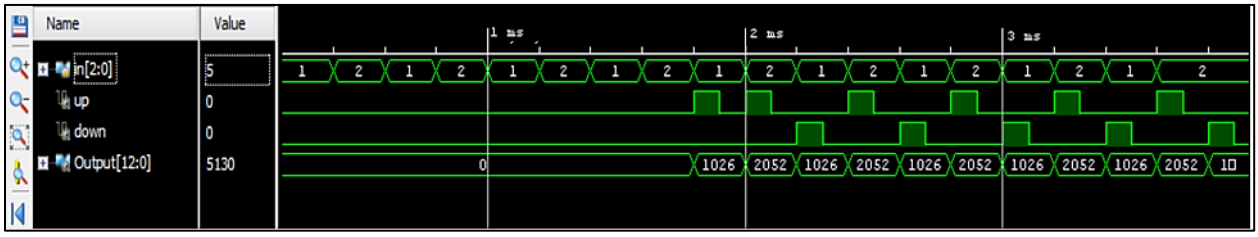



*Figure 7.6: Output of 2.5Khz signal after FIR low pass filter.*

The output of the FIR filter after the addition is 1026 and 2052. This output is then converted to new filter output (by dividing the output with 1024, refer section 4.3.1). This is done manually but, it is a lot easier to do it on hardware (shifting to right).

After dividing, the new output is 1.001953 and 2.003906.

18 such inputs are then interpolated by 4 to match the sampling frequency (refer section 3.6) and passed through the filter designed in MATLAB. Shown below are the MATLAB simulations.

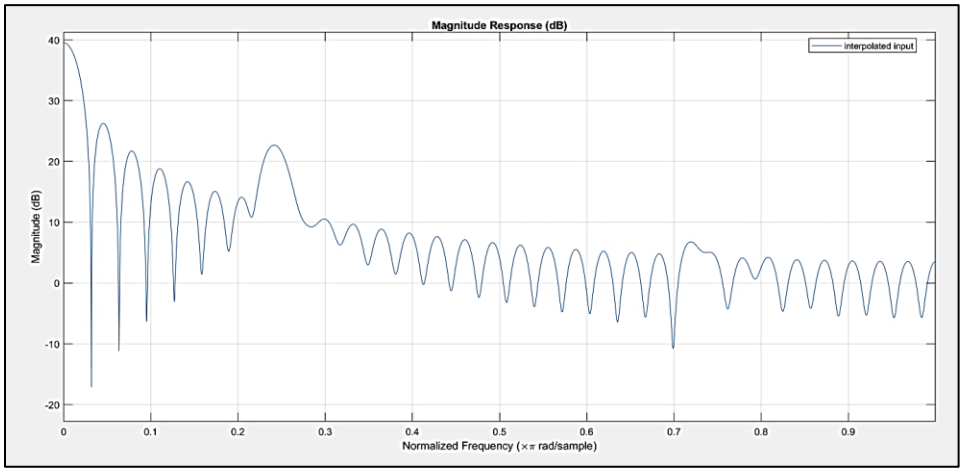



*Figure 7.7: 2.5kHz signal (interpolated input signal)*

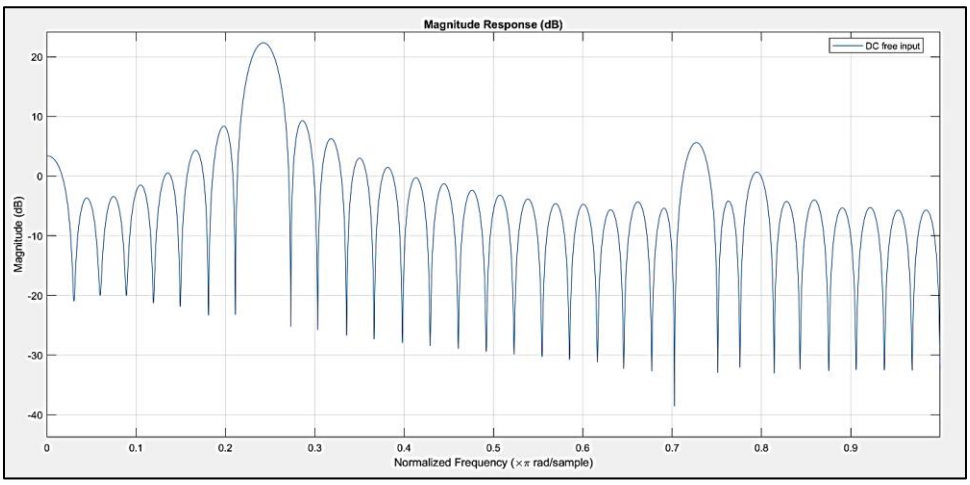



*Figure7.8: DC free input signal.*

Figure 7.7 is an interpolated input signal at 2.5*kHz* frequency. It has a peak at 0.25 (normalised frequency taken here is 10*kHz*). It can be seen that the input signal has a DC component with the signal. This is because the output from FPGA has a positive mean whereas there were no DC components when the waves generated in MATLAB in section 3.6 and 3.7. This can be corrected by taking the mean average and subtracting the original values with the mean average value. This technique is called as moving average filter [34].

The output after applying a moving average filter or removing the DC components from the input signal is shown in figure 7.8. The processed signal in figure 7.8 is similar to that of figure 7.7 with a peak at 0.25 and another at 0.75 (after 1 period) except DC components.

The signal should pass after passing through a Kaiser low pass filter as the stop band designed of the filter designed is 6*kHz* or 0.6(in the above graph).

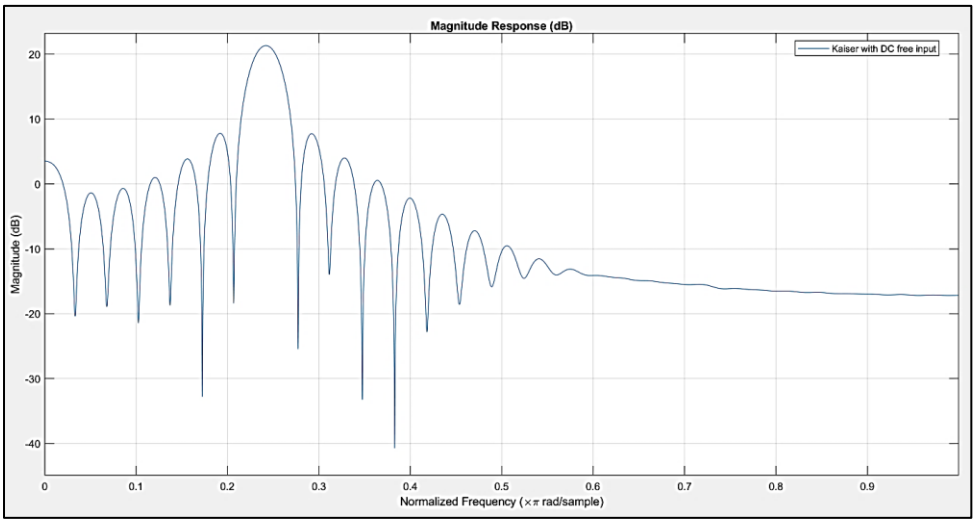



*Figure 7.9: The output of 2.5kHz DC free input signal.*

As expected, the signal passes through the low pass filter.

Refer to appendix A2 for impulse responses of these simulations.

### 7.3.2 Two input signal of different frequency 2.5*kHz* and 7.5*kHz* are given input to FPGA

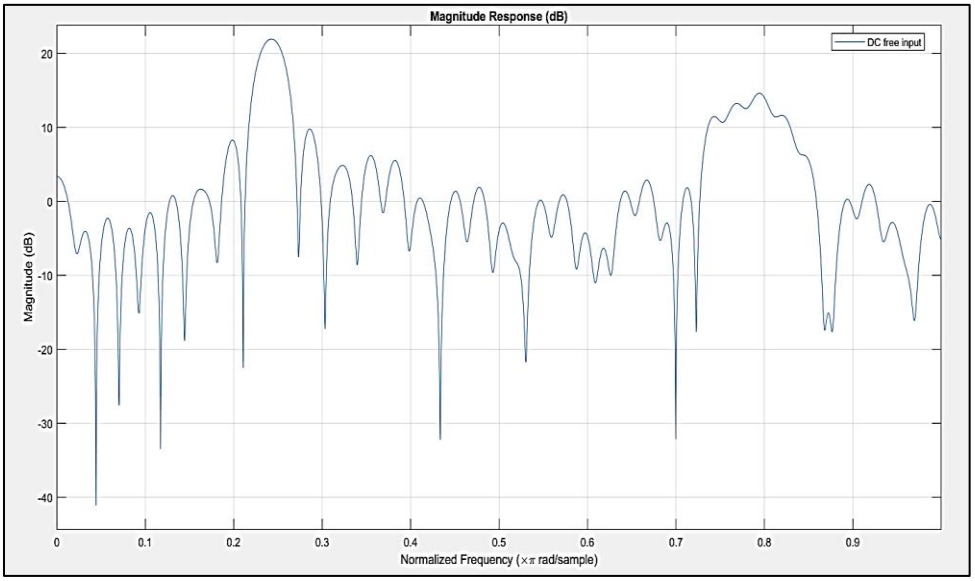



*Figure 7.10: DC free mixed input signal (2.5kHz and 7.5kHz)*

The input signal has two peaks, one at 0.25(2.5*kHz*) and other at 0.75(7.5*kHz*). The filter should block the 7.5*kHz* frequency and should allow 2.5*kHz* frequency to pass through it.

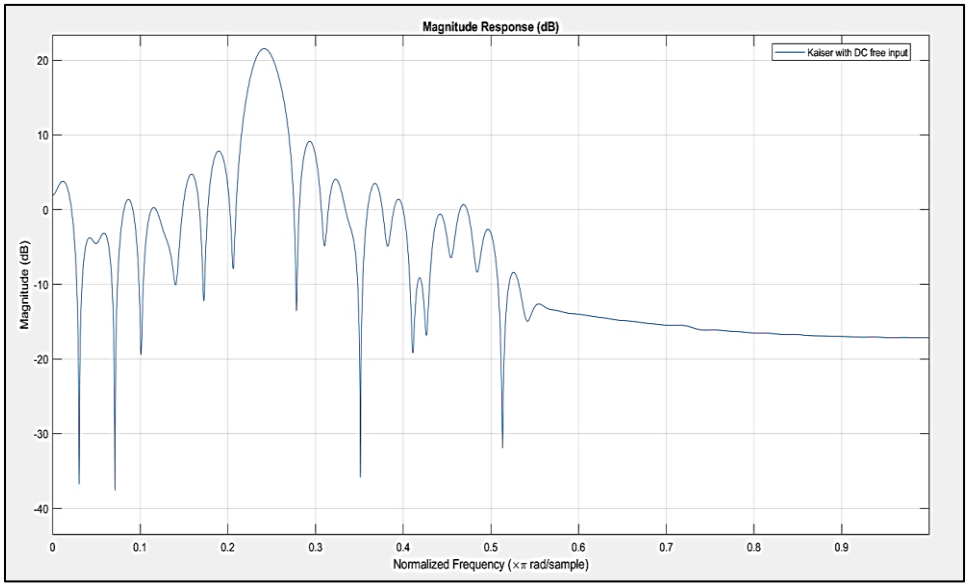



*Figure 7.11: Output of figure 7.9 after passing though low pass filter.*

As expected, the filter allowed 2.5*kHz* signal to pass and blocks the 7.5*kHz* signal. The impulse response of input and output signals are shown in figure 7.12 and figure 7.13.

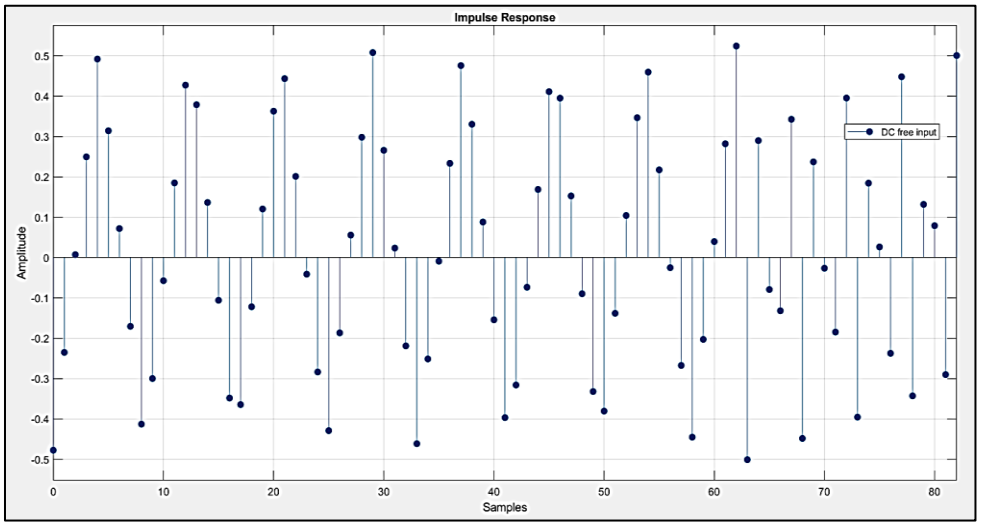



*Figure 7.12: Impulse response of DC free input signal.*

The impulse response of the input signal is ranging from -0.45 to +0.45 or +0.5 for the 2.5*kHz* signal and from -0.5 to +0.52 for 7.5*kHz* (higher frequency) signal. Figure 7.13 confirms the output signal is attenuated as the range of the higher frequency is attenuated to -0.05 to +0.05(amplitude), whereas the impulse response of the input signal is almost like that of figure 7.10.

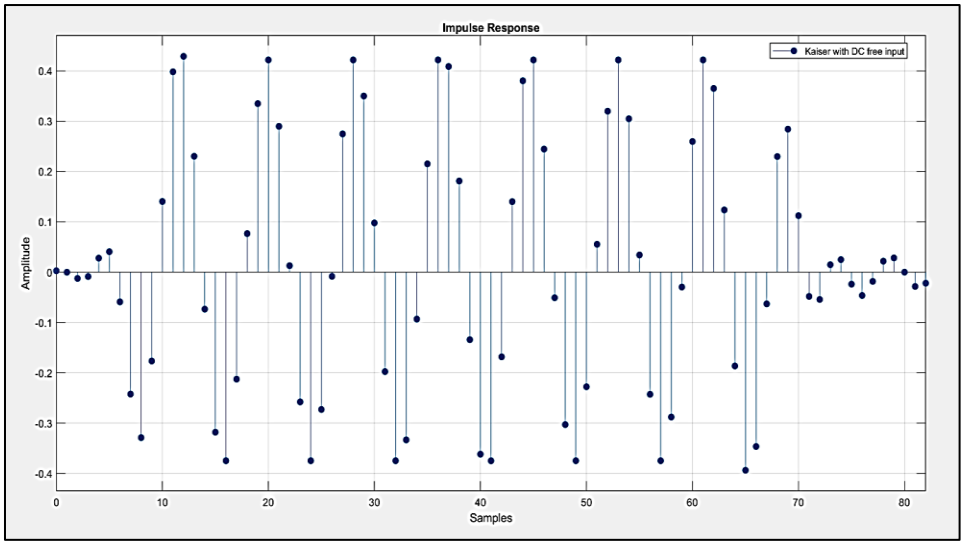



*Figure 7.13: Impulse response of filter.*

## 7.4 Results

The asynchronous processor is successfully designed in FPGA.

The objective of the project was to reduce the power dissipation due to the synchronous processors (clock-based processors). Because an asynchronous processor was designed on FPGA, there is no clock in the system which usually takes a lot of power [2].

The power dissipation (synthesis report) of the synchronous processor designed by Junfei Mao in [4] is **22.45*W***. This is shown in figure 7.14.

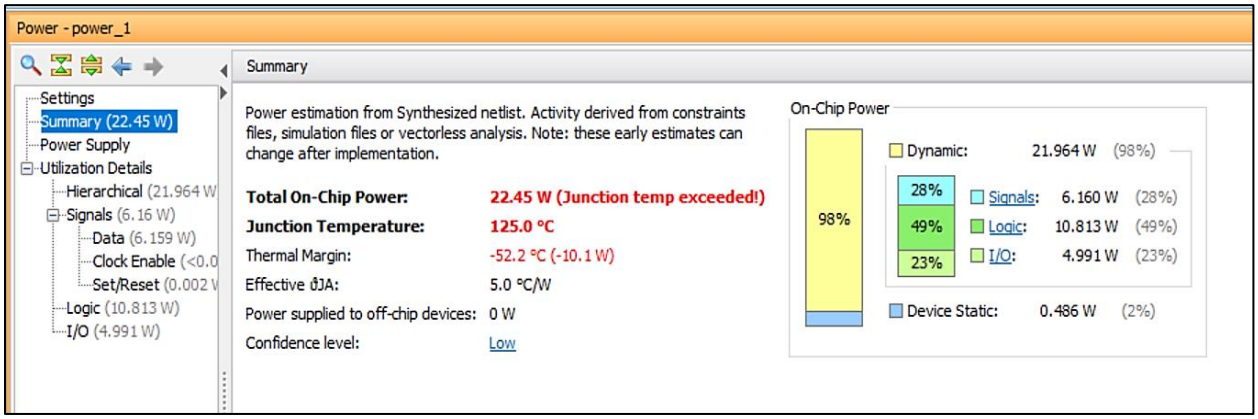



*Figure 7.14: Power dissipation (synthesis report) of Junfei Mao's synchronous processor.*

The power dissipation (synthesis) report of the asynchronous processor designed in this project is **0.084*W*** which is presented in figure 7.15.

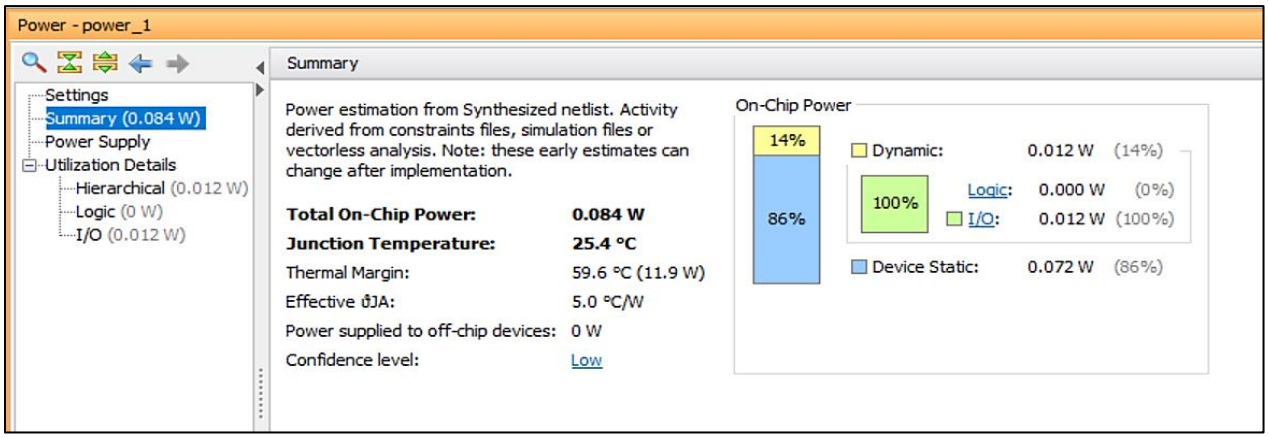



*Figure 7.15: Power dissipation (synthesis report) of asynchronous processor designed in this project.*

This difference in the power between the two approaches is due to the dynamic power. Because the system has no clock, therefore there is very less dynamic power which can be compared from the two figures 7.14 and 7.15.

Dynamic power dissipation is due to sequential logic design which uses flip-flops and clock. Power consumed in sequential circuits are due to latches which are NAND gates. Since, there is no sequential circuit (logic) used in the design, there is negligible power consumption due to this factor.

The only power which is dissipating through the chip (FPGA) is static power. Static power is due to wire losses, wire resistances and capacitances (refer section 1.2). It includes the power consumed across the wire when the current is transferred from part a to part b ( $P = I^2R$ where $P$ is power in $W$, $I$ is current and $R$ is resistance). Second, static power is also consumed is due to the changing and discharging of the capacitors.

The power dissipation across the analogue delay line is calculated by the formula $P = V_{DD}(I_{SS}) – V_{SS}(I_{SS})$. $V_{DD}$ and $V_{SS}$ are supplied through the power supply(voltage generator) which are $+5V$ and $-5V$. $I_{SS}$ is $\sim 42mA$ when measured through a multimeter. Therefore, the power dissipation across the analogue delay block calculated is **0.42*W***.

The **total power dissipation** of the whole system is $0.084W + 0.42W$ = **0.504*W***.

## 7.5 Summary

In this chapter, an overview of the system block diagram is presented. The analogue-to-event conversion is done asynchronously. The processed data is then converted to digital domain for a processor to process an FIR filter. The output of the FIR filter is then verified by simulating the output results on MATLAB. Lastly, event-to-analogue converter is implemented which generates an analogue wave at the output.

In a nutshell, the whole process is able to reduce the power to **0.504W** thus fulfilling the project goal.

# Chapter 8

# Project conclusion and further work

## 8.1 Conclusions

This project was an ambitious undertaking, requiring huge implementation of circuits both on breadboards and cadence. It also required many novel digital design ideas to process asynchronous processors. In addition, there were many challenges faced, for instance, the implantation of analogue delay lines and connecting them asynchronously to the FPGA, designing a less tap filter on MATLAB for the first time, verifying the outputs from FPGA on MATLAB, coding in Verilog-a for analogue-to-digital converter (Flash ADC) thus, exploring the digital side of cadence tool. Ultimately all the challenges were done successfully.

This system can be used to power low power devices which requires less amount of energy to function, for example, fit bits, smart watches and even mobile phones. Asynchronous processers can save a lot of energy and can be used for radio communications, military purposes and aviation (where clear channel of frequency is required for the system to run accurately), thus improving the efficiency to power loss ratio.

## 8.2 Achievements

At the end of the project, the following tasks were achieved-

- Successful simulation of analogue-to-digital conversion using level-sampling scheme and implementing it using a flash ADC on cadence.
- Implementation of analogue delay lines on breadboard.
- Successful in decreasing the filter coefficients to 16 from 179 as implemented by Junfei Mao in[3].
- Designing a successful asynchronous processor which functions as a successful low pass filter and can pass a signal up to $\approx 6kHz$ frequency and block the signals above $6kHz$. This is tested successfully on MATLAB in section 3.7 and section 7.2.
- The main objective of the project was to reduce the power dissipation which is achieved in the project (refer section 7.3). The overall power dissipation was $\mathbf{0.5}\boldsymbol{W}$.
- Designed digital to analogue converter using op-amp as an integrator on PCB.

The overall project structure is shown in figure 7.1 and a breadboard design is shown in appendix A4.

## 8.3 Project further work

Although the project is implemented successfully but a lot can be done to improve the working of the whole processing system. Since there was a deadline for the project, there was not enough time to go through about every techniques, schemes and circuit implementations, therefore, one of the methods was opted for the design in order to complete the project on time. Hence, this section will highlight possible further works and suggestions.

- The delay system is designed and implemented on three different breadboards. Breadboards are not always accurate and have various losses and effects of its own which were covered in section 5.4.1. These losses can be avoided, and better results can be achieved by designing it on a PCB.

- The power of the analogue delay lines can be further reduced by reducing $V_{DD}$ and $V_{SS}$. Amplifier can be designed which uses less power to function (for example with $V_{DD}$ and $V_{SS}$ at $\pm 3.3V$) .

- The FPGA used in the project (Xilinx Digilent Basys-3 board) has only 32 input-output ports, so, a maximum of 9 delay blocks were successfully tested on the FPGA board. A better FPGA board with at-least 60 input-output pins is required to test the complete system successfully. Signal processing can also be implemented using field programmable analogue array as mentioned in [34].

- Implementation of the work done by [3] and [6] can be done on PCB to make event-to-analogue converters to work more efficiently.

- The speed of the analogue-to-digital converters depends on comparators, the speed can be increased by choosing high-speed comparators.

# Acknowledgement

This report would not have been possible without the contributions of a few key individuals. First and foremost, I would like to express special gratitude to my project supervisor, Dr. Alister Hamilton, who in spite of his extraordinarily busy schedule, invested his full effort in guiding me to achieve my project goal. I wish to thank him for his patience and constant guidance and encouragement in the past few months. I have been fortunate to learn so much from his technical insight and mentorship.

I would also like to thank my friends who supported and encouraged me to strive towards my goals. Last but not least, I would like to express my appreciation to my beloved family. Their words of encouragement and support have striven me towards the completion of my MSc thesis project.

# References


[1] Maharshi, “What is static power dissipation and dynamic power dissipation?,” edaboard.com, [Online]. Available: https://www.edaboard.com/showthread.php?67491-What-is-static-power-dissipation-and-dynamic-power-dissipation. [Accessed 8 August 2018].

[2] Y. Tsividis, “Event-Driven Data Acquisition and Continuous-Time Digital Signal Processing,” *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS,* vol. 57, pp. 577-581, 2010.

[3] A. G. Mitropoulos, “An Event Driven Continuous Time Analogue to Digital,” University of Edinburgh, Edinburgh, 2011.

[4] J. Mao, “An Event-based Signal Processing Platform,” The University Of Edinburgh, Edinburgh, 2016.

[5] B. S. a. Y. Tsividis, “A continuous-time ADC/DSP/DAC system with no clock and,” *IEEE J. Solid-State Circuit,* vol. 43, pp. 2472-2481, 2008.

[6] A. T. W. K. Inose H, “Asynchronous delta-modulation system,” *Electron. Lett.,* vol. 2, no. 3, pp. 95-96, 19661.

[7] S. B. a. F. M. Mohammadmehdi Kafashan*, “Asynchronous analog-to-digital converter based on level-crossing sampling scheme,” *EURASIP Journal on Advances in Signal Processing,* p. 12, 2011.

[8] T. Zhu, *Analog to Digital Converters,* 2008.

[9] R. S. N. Makarov, “Practical Electrcal Engineering,” Springer Internatonal Publishing, Switzerland, 2013.

[10] R. S. Thakur, “An Event-based Signal Processing System,” The University of Edinburgh, Edinburgh, 2018.

[11] P. Sideris, “Verilog-A Modelling of Organic Electrochemical Transistors and Read-out Instrumentation,” ARISTOTLE UNIVERSITY OF THESSALONIKI, THESSALONIKI, 2016.

[12] Renesas, “An Introduction to Digital Filters,” Renesas Electronivs Corporation, 1999.

[13] MATLAB, “FIR Filter Design,” MathWorks, [Online]. Available: https://uk.mathworks.com/help/signal/ug/fir-filter-design.html#brbq5xn. [Accessed 1 August 2018].

[14] J. G. P. D. G. Manolakis, “Design of Digital Filters,” in *Digital Signal Processing*, Harlow, Essex : Pearson, 2014, pp. 670-694.

[15] E. C. I. &. B. W. Jervis, Digital Signal Processing : A practical Approach, Pearson Education Asia, 2003.

[16] S. K. Mitra, Digital Signal Processing : A computer Based Approach, McGraw-Hill, 2006.

[17] A. L. Choodarathnakara, “Digital FIR–LP Filter using Window Functions,” *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET),* vol. 4, no. 4, pp. 1349 - 1356, 2015.

[18] A. P. Rainy Chaplot, “Low Power Reconfigurable FIR Filter Based on Window Techniques for On Chip Network,” *IEEE,* pp. 291-296, 2013.

[19] MathWorks, “kaiser,” [Online]. Available: https://uk.mathworks.com/help/signal/ref/kaiser.html. [Accessed 5 August 2018].

[20] Z. Hassan, “ASYNCHRONOUS AND SELF TIMED,” Tampere University of Technology.

[21] K. A. Fawaz, “A Dynamically Reconfigurable Asynchronous Processor,” The University of Edinburgh, Edinburgh, 2012.

[22] M. N. a. H. S. Jukiya Furushima, “Design of an Asynchronous Processor with Bundled-data Implementation on a Commercial Field Programmable Gate Array,” *Informatica,* vol. 40, pp. 399-408, 2016.

[23] P. L.-E. Magnus Sjalander, “High-speed and low-power multipliers using the Baugh-Wooley algorithm and HPM reduction tree,” *IEEE Electronics, Circuits and Systems,* pp. 33-36, 2008.

[24] C. Today, “Ripple carry adder,” Circuits Today, 2012. [Online]. Available: http://www.circuitstoday.com/ripple-carry-adder. [Accessed 9 August 2018].

[25] “Bucket Brigade Devices: MN3007,” Electrosmash, [Online]. Available: https://www.electrosmash.com/mn3007-bucket-brigade-devices. [Accessed 11 August 2018].

[26] R. MARSTON, “Analogue Delay Lines,” *Circuits,* pp. 67-72, 1986.

[27] Wikipedia, “All pass filter,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/All-pass_filter. [Accessed 11 August 2018].

[28] H. Zumbahlen, “All Pass Filters,” Analog Devices, Norwood, MA, USA.

[29] Renesas, “Op-Amps, Comparator Circuit,” Renesas (big ideas for every space), [Online]. Available: https://www.renesas.com/eu/en/support/technical-resources/engineer-school/electronic-circuits-03-op-amps-comparator-circuit.html. [Accessed 11 August 2018].

[30] “What is a breadboard?,” Wiring, [Online]. Available: http://wiring.org.co/learning/tutorials/breadboard/. [Accessed 11 August 2018].

[31] T. Instruments, “LM741 Operational Amplifier,” October 2015. [Online]. Available: http://www.ti.com/lit/ds/symlink/lm741.pdf. [Accessed 12 August 2018].

[32] N. Semiconductor, “LM148/LM248/LM348,” November 2003. [Online]. Available: http://datasheetz.com/data/Integrated%20Circuits%20(ICs)/Amplifiers%20-%20Instrumentation,%20OP%20Amps,%20Buffer%20Amps/LM348M-datasheetz.html. [Accessed 12 August 2018].

[33] T. Instruments, “CMOS NAND GATE,” 2003. [Online]. Available: http://www.ti.com/lit/ds/symlink/cd4011b.pdf. [Accessed 12 August 2018].

[34] Wikipedia, “Digital-to-analogue-converter,” [Online]. Available: https://en.wikipedia.org/wiki/Digital-to-analog_converter. [Accessed 12 August 2018].

[35] R. Nave, “Digital-to-Analog Conversion,” [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/dac.html. [Accessed 12 August 2018].

[36] “The Integrator Amplifier,” Electronics Tutorial, [Online]. Available: https://www.electronics-tutorials.ws/opamp/opamp_6.html. [Accessed 12 August 2018].

[37] W. Jiang, “Asynchronous Processor Design,” 2007.

[38] Wikipedia, “Electronic design automation,” [Online]. Available: https://en.m.wikipedia.org/wiki/Electronic_design_automation. [Accessed 8 August 2018].

[39] Xilinx, “Digilent Basys 3 Artix-7 FPGA Board,” Xilinx, [Online]. Available: https://www.xilinx.com/products/boards-and-kits/1-54wqge.html. [Accessed 9 August 2018].

[40] Wikipedia, “Analogue delay line,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Analog_delay_line. [Accessed 11 August 2018].

[41] J. B. Calvert, “Analogue delay line,” January 2002. [Online]. Available: https://mysite.du.edu/~etuttle/electron/elect39.htm. [Accessed 11 August 2018].

[42] F. Athar, “How to Use Moving Average Filter to Counter Noisy Data Signal?,” BLUEEAST, 3 December 2017. [Online]. Available: https://medium.com/blueeast/how-to-use-moving-average-filter-to-counter-noisy-data-signal-5b530294a12e. [Accessed 13 August 2018].

[43] T. J. K. a. A. H. Luiz Carlos Gouveia, “An Asynchronous Spike Event Coding Scheme for Programmable Analog Arrays,” *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS,* vol. 58, no. 4, pp. 791- 799, 2011.

# A1

## Verilog-a code, schematic and circuit diagrams

### Comparator

#Comparator module

```
// VerilogA for PROJECT2018, Comparator, veriloga
`include "constants.vams"
`include "disciplines.vams"

module Comparator(vin, vout, vdd, vss, ref);
input vin, ref;                                  // Input signals
inout vss, vdd;                                  // inout signals
output vout;                                     // output signal

parameter delay=0, ttime =1p;
electrical vin,ref,vdd,vout, vss;
real result;                                     // main block
analog begin
@(cross((V(vin)-V(ref)),0) or initial_step ) begin
if(V(vin) >= V(ref))                             // condition for comparator
result = V(vdd);
else
result = V(vss);
end
V(vout) <+ transition (result, delay, ttime);    // assigning output to vout
end
endmodule
```

### 8Bit ADC code with up down signal

```
// VerilogA for PROJECT2018, Thesis_1, veriloga
// This code is an Encoder + Thermometer + Up and down signal code for the Flash ADC part
`include "constants.vams"
`include "disciplines.vams"

module Thesis_1(in,in2,t0,t1,t2,t3,t4,t5,t6,t7,d0,d1,d2,up,down);
                                                                    //IO ports
input in,in2,t1,t2,t3,t4,t5,t6,t7,t0;
```

```
output d0,d1,d2,up,down;                                        //Input Ports


electrical in,in2,t0,t1,t2,t3,t4,t5,t6,t7,d0,d1,d2,up,down;     //Output Ports

                                                                //Signals
real dzero,done,dtwo;
real up1,down1;
real x;
real DDLAY;
real Delayed_in;

                                                        //Main block starts here (Encoder)
analog begin

                                                        //Condition for Up and down signals
if (V(in) > V(in2))
        begin
        up1 = 1;
        down1 = 0;
        end

else if (V(in) < V(in2))
        begin
        up1 = 0;
        down1 = 2;
        end

else if (V(in2) == V(in))
        begin
        up1 = 0;
        down1 = 0;
        end
                                                //Conditions of Thermometer code or Encoder
if (V(t7) == 10)
        begin
        dzero   = 1;
        done    = 1;
        dtwo    = 1;
        end

else if (V(t6) ==10)
        begin
        dzero   = 0;
        done    = 1;
        dtwo    = 1;
        end
```

```
else if (V(t5) ==10)
        begin
        dzero   = 1;
        done    = 0;
        dtwo    = 1;
        end


else if (V(t4) ==10)
        begin
        dzero   = 0;
        done    = 0;
        dtwo    = 1;
        end

else if (V(t3) ==10)
        begin
        dzero   = 1;
        done    = 1;
        dtwo    = 0;
        end

else if (V(t2) ==10)
        begin
        dzero   = 0;
        done    = 1;
        dtwo    = 0;
        end

else if (V(t1) ==10)
        begin
        dzero   = 1;
        done    = 0;
        dtwo    = 0;
        end

else if (V(t0) ==10)
        begin
        dzero   = 0;
        done    = 0;
        dtwo    = 0;
        end

V(d0) <+ transition(dzero, 0, 1p);
V(d1) <+ transition(done, 0, 1p);
V(d2) <+ transition(dtwo, 0, 1p);
V(up) <+ transition(up1, 0, 1p);
V(down) <+ transition(down1, 0, 1p);
```

```
end
endmodule
```

## Schematic Diagram

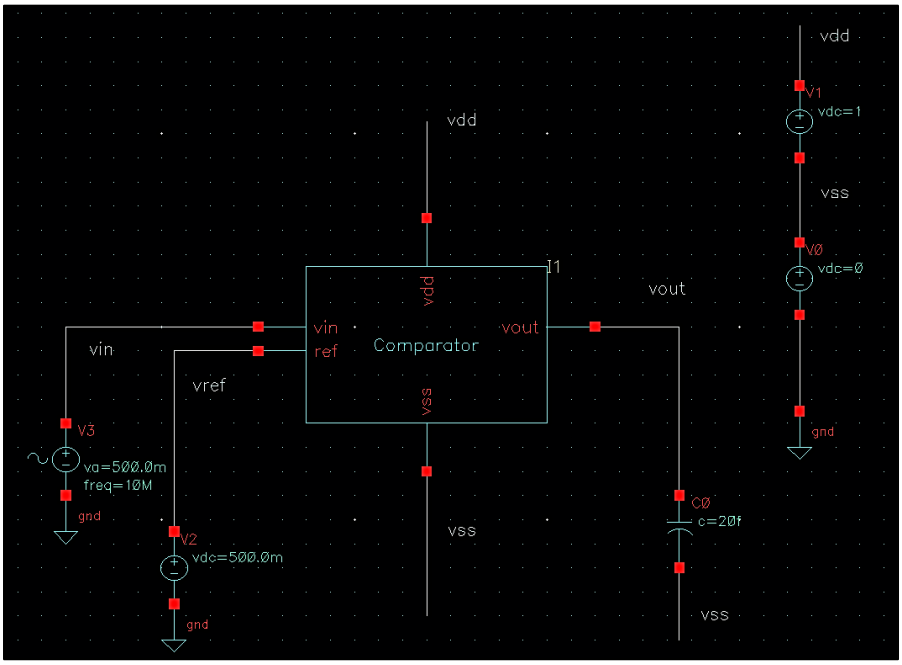



*Figure A1.1: Schematic of comparator designed in Verilog-a*

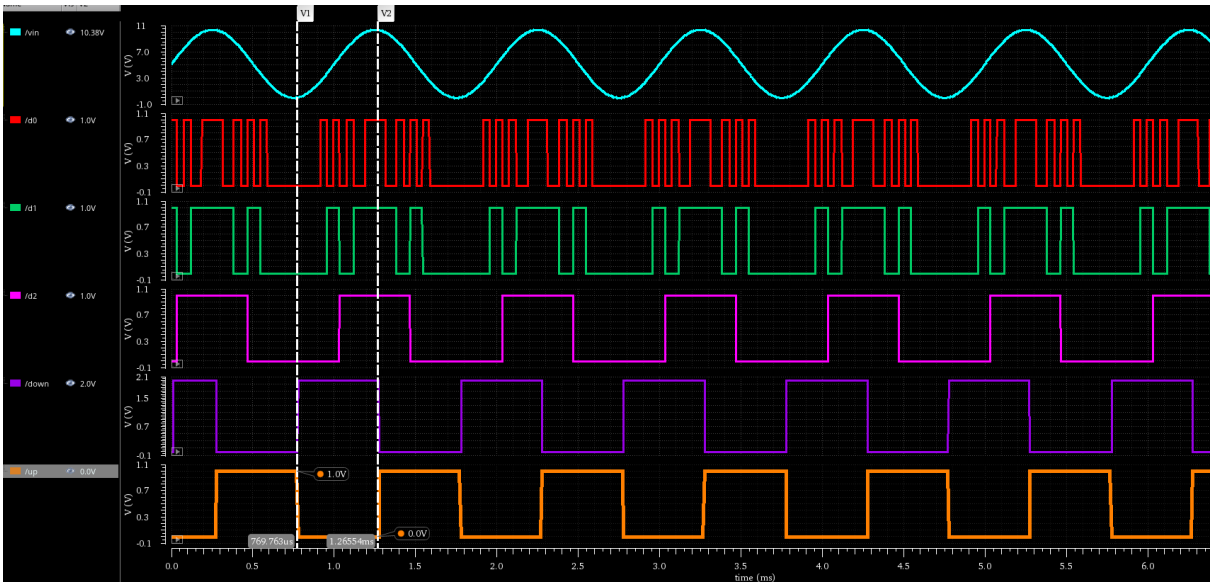



*Figure A1.2: Simulation of 8-bit flash ADC*

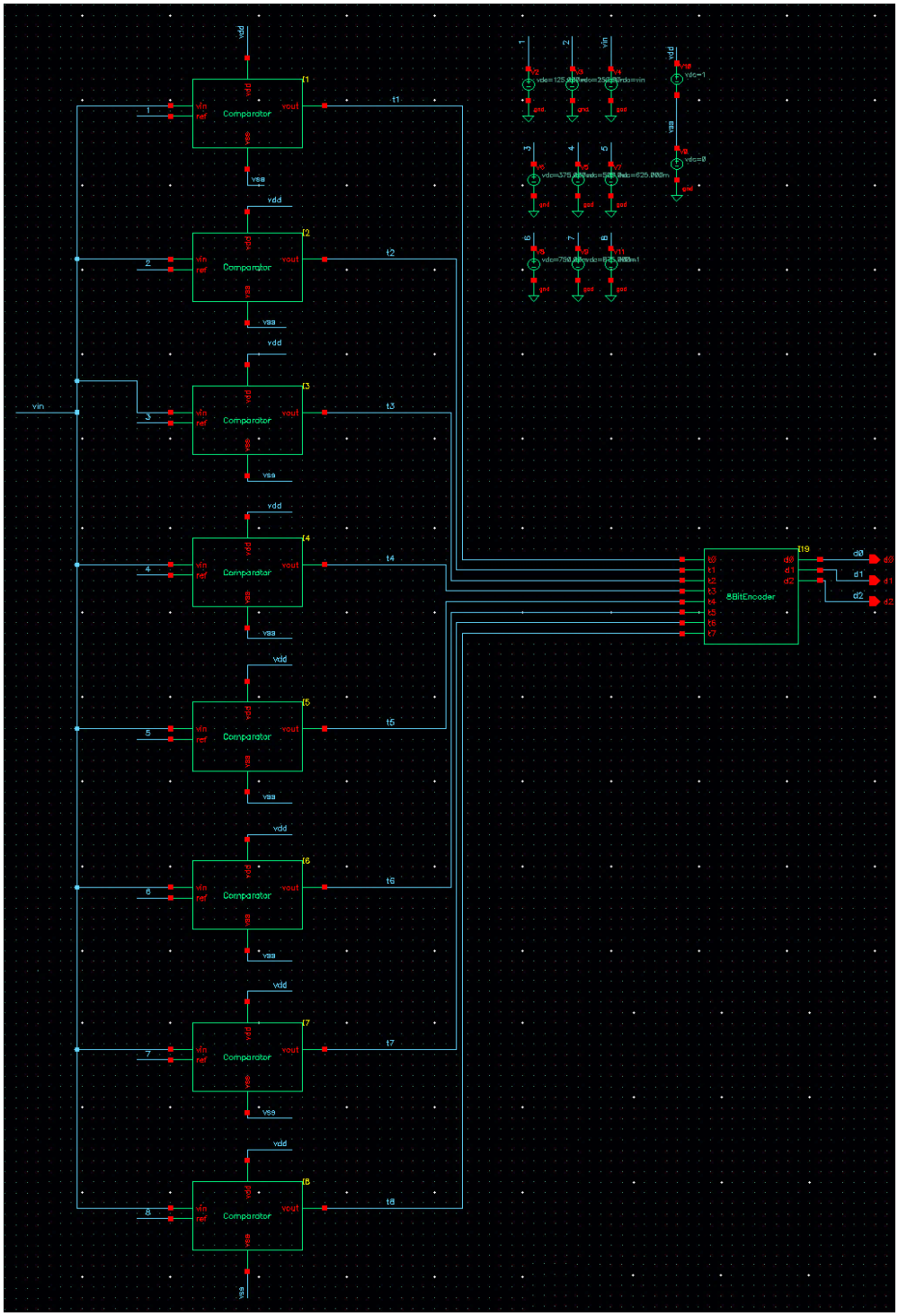



*Figure A1.3: Circuit diagram of Flash ADC converter.*

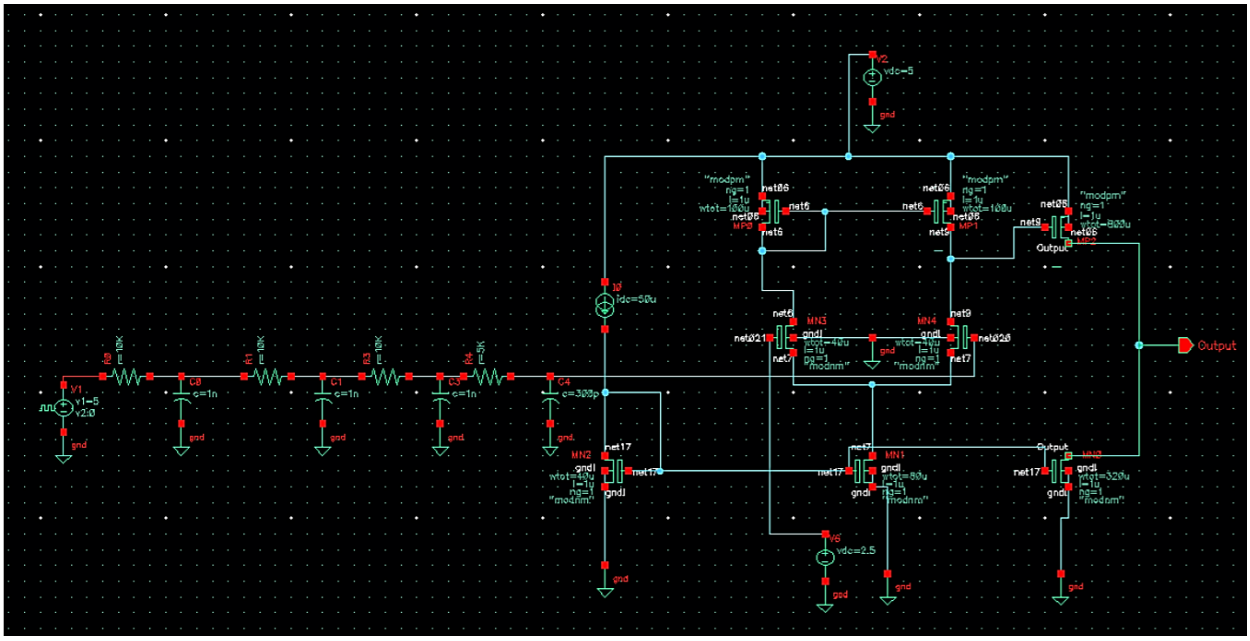



*Figure A1.4: Schematic of delay generation using amplifier as a comparator*

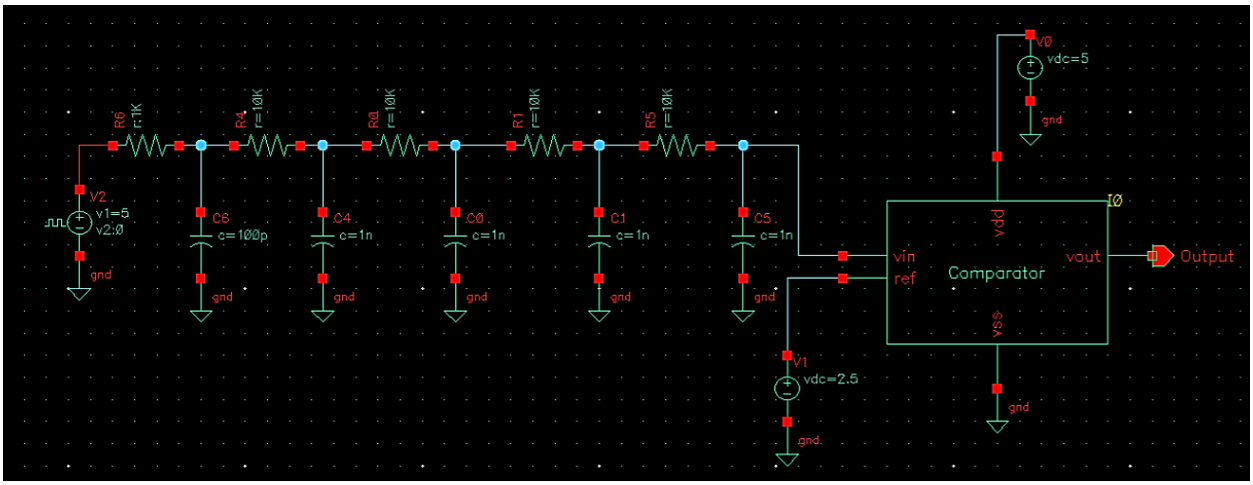



*Figure A1.5: Schematic of delay using comparator(designed in Verilog-a)*

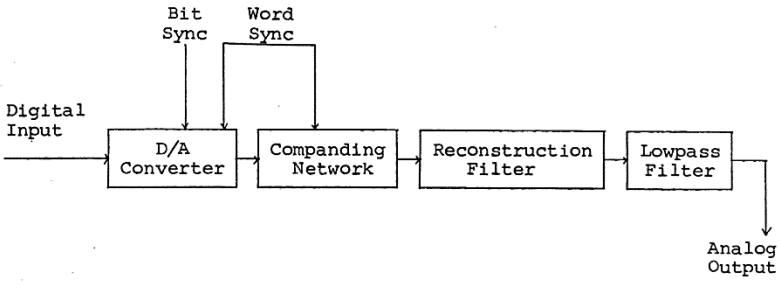



*Figure A1.6: Demodulator of asynchronous delta modulator [6]*

# A2

## MATLAB Simulations

```
n=0:32
x = sin(2*pi*0.1*n)
H =[-0.0008916,0.0008265,0.010128,0.0017292,-0.0463616,-
0.0349115,0.1615095,0.4084107,0.4084107,0.1615095,-0.0349115,-
0.0463616,0.0017292,0.010128,0.0008265,-0.0008916]
%hfvt = fvtool(H,1);

hfvt2 = fvtool(x,1);
YY = filter(H,1,x);
hfvt = fvtool(YY,1);
```

*Listing A2.1: Source code of 2Khz signal in MATLAB*

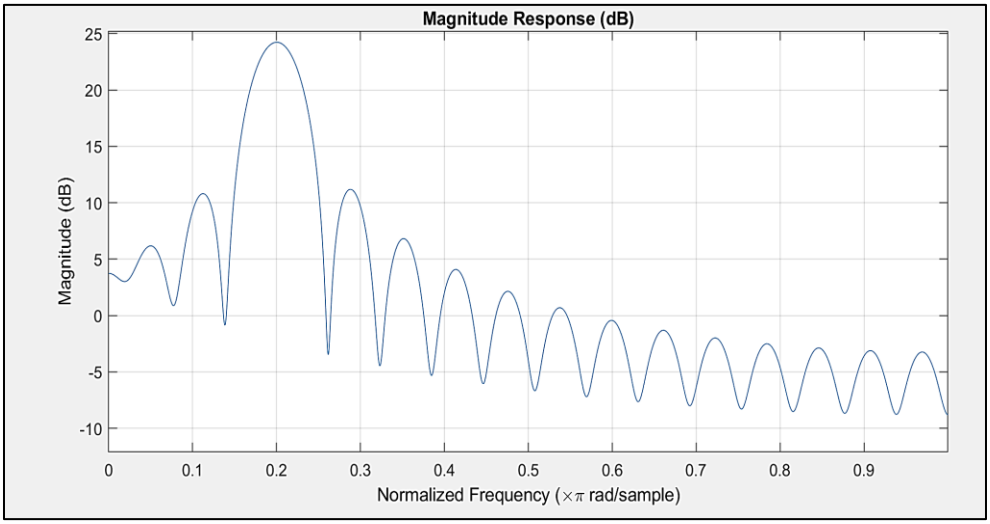



*Figure A2.1: Input signal of Listing A2.1*

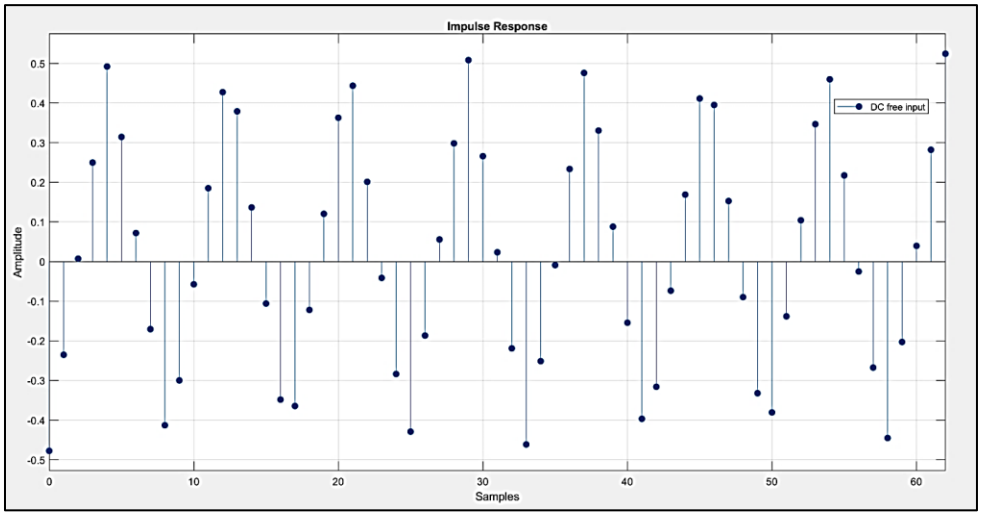



*Figure A2.2: Input impulse response of figure 7.5*

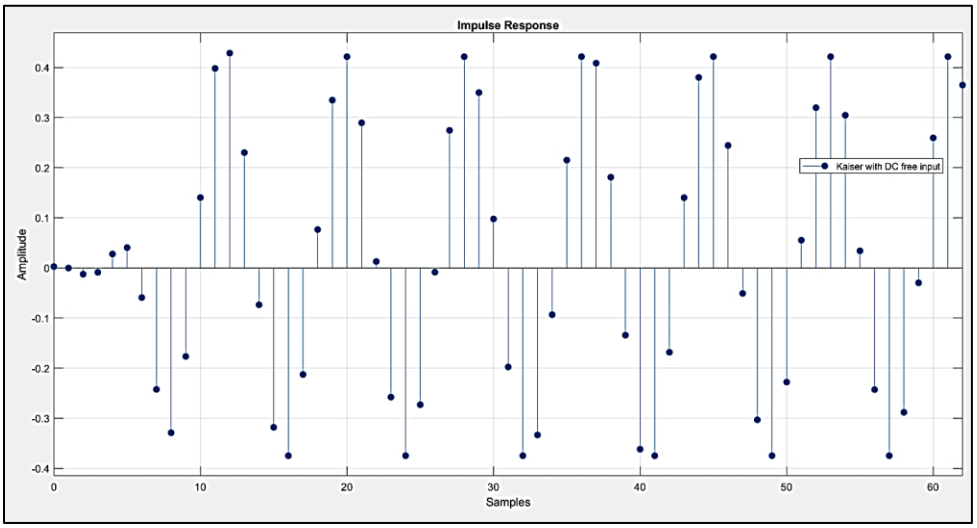



*Figure A2.3: Filter impulse response of figure 7.6.*

In figure A2.2, the input impulse has a range of -0.45 to +0.5. In figure B.3, the filter impulse response has a range from -0.4 to +0.42. The 2.5KHz signal is passed through the designed low pass filter.

# A3

## Verilog Simulations

### BAUGH WOOLEY MULTIPLIER

```
// Half Adder
module half_adder(a, b, s, cout);
input a, b;
output s, cout;
assign s = a^b;
assign cout = a&b;
endmodule
```

```
// Full Adder
module full_adder(a, b, cin, s, cout);
        input a, b, cin;                                //Input signal
        output s, cout;                                 //Output signal
                assign s = a^b^cin;                     //Sum
                assign cout = (a&b) | (b&cin) | (a&cin); //Carry
endmodule
```

```
//4Bit Multiplier (SIGNED MULTIPLIER)

module mult4bw(x, y, p);
        input [3:0] x, y;
        output [7:0] p;
        // constant logic-one value
        supply1 one;
        // Internal results which is given to next stage adders
        wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12,t13, t14, t15, t16, t17, t18, t19, t20, t21, t22, t23;
        // structural description of the multiplier circuit
        assign p[0] = x[0]&y[0];
                half_adder ha1(x[1]&y[0], x[0]&y[1], p[1], t1);
                half_adder ha2(x[2]&y[0], x[1]&y[1], t2, t3);
                full_adder fa1(t2, t1, x[0]&y[2], p[2], t4);
                half_adder ha3(x[3]&~y[0], x[2]&y[1], t5, t6);
                full_adder fa2(t5, t3, x[1]&y[2], t7, t8);
                full_adder fa3(t7, t4, ~x[0]&y[3], t9, t10);
                full_adder fa4(t9, x[3], y[3], p[3], t11);
                full_adder fa5(x[3]&~y[1], t6, x[2]&y[2], t12, t13);
                full_adder fa6(t12, t8, ~x[1]&y[3], t14, t15);
                full_adder fa7(t14, t10, t11, p[4], t16);
                full_adder fa8(x[3]&~y[2], t13, ~x[2]&y[3], t17, t18);
```

```
        full_adder fa9(t17, t15, t16, p[5], t19);
        full_adder fa10(~x[3], ~y[3], x[3]&y[3], t20, t21);
        full_adder fa11(t20, t18, t19, p[6], t22);
        full_adder fa12(one, t21, t22, p[7], t23);
endmodule
```

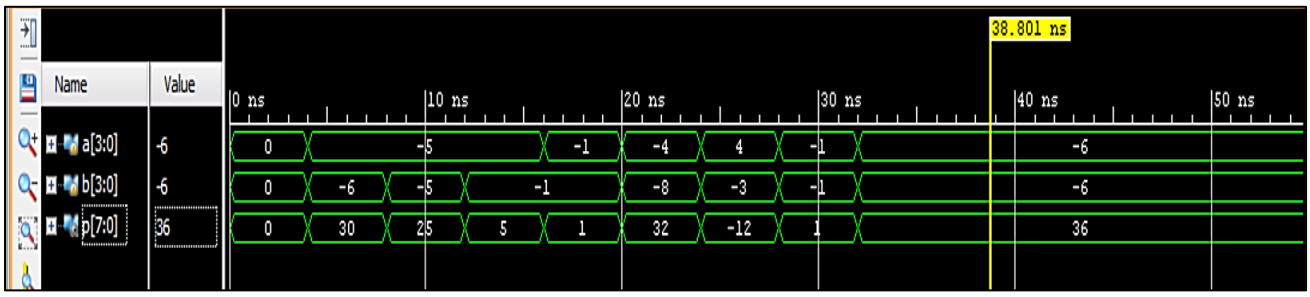



*Figure A3.1 : Baugh Wooley signed multiplier*

The output generated in figure A3.1 is correct.

# A4

## Breadboard Design and simulation


*Figure A4.1: Output of 16th delay block generated from signal generator*

The overall delay from the input (from signal generator) (to the 1st delay) to the output of 16th delay is 1.16ms.


*Figure A4.2: Output of 16th Delay block generated from FPGA.*

The overall delay from the input (from FPGA) (to the 1st delay) to the output of 16th delay is 1.18ms.

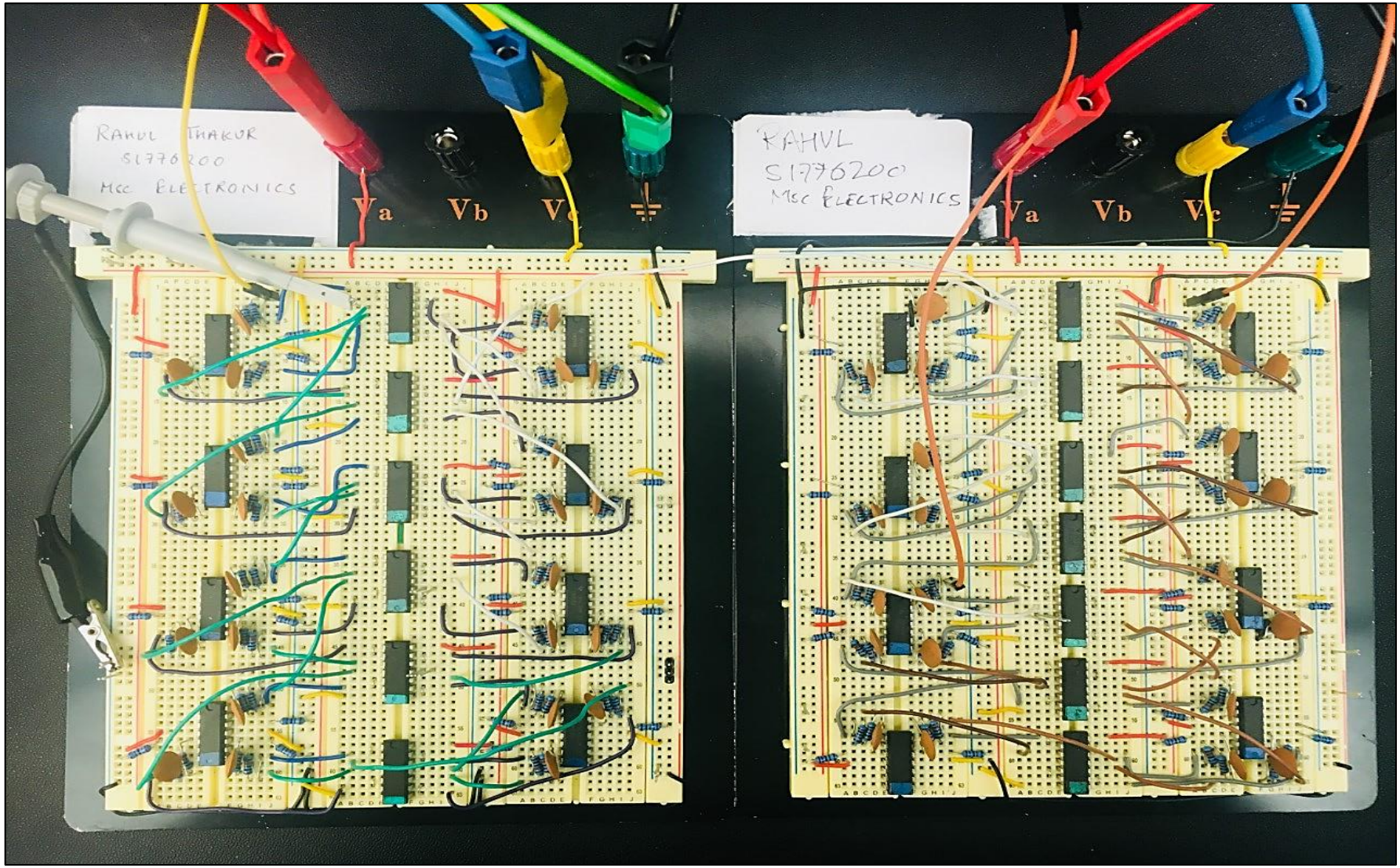



*Figure A4.3: 16 delay blocks on breadboard*

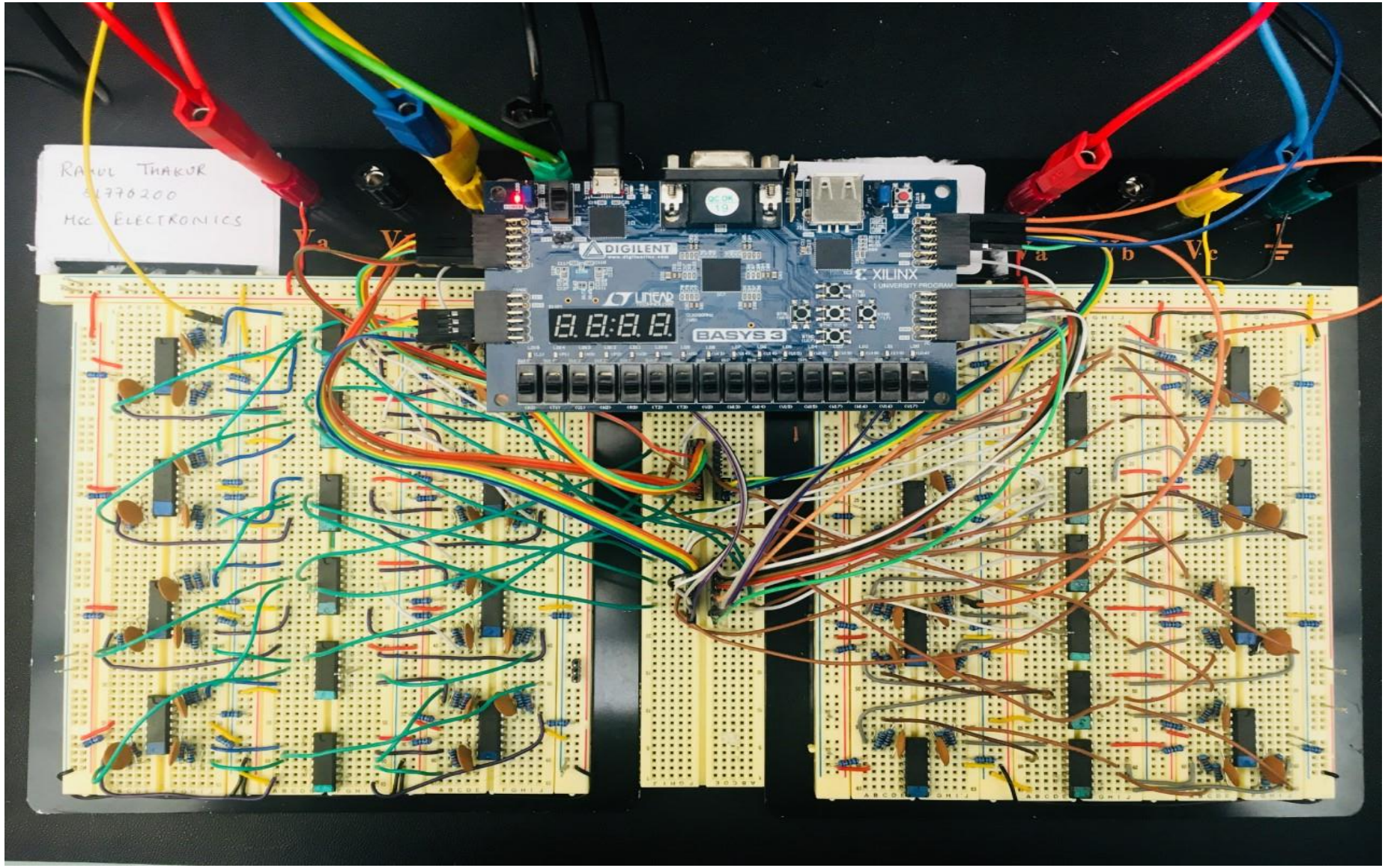



*Figure A4.4: Overall project structure.*

# A5

## Components and data sheet

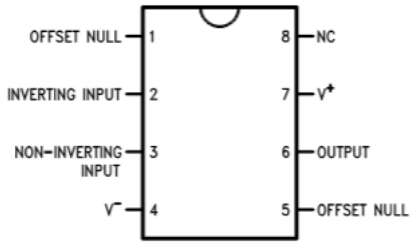



*Figure A5.1: Data sheet of IC LM741 op-amp [31]*

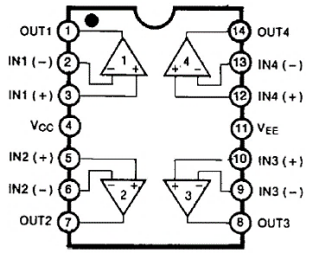



*Figure A5.2: Data sheet of IC LM348M op-amp [32]*

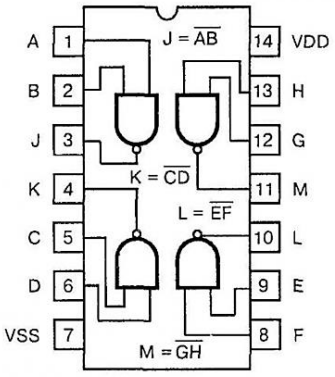



*Figure A5.3: Data sheet of IC CD4011BE NAND gate [33]*

*Table A5.1 : Components list*

| COMPONENTS | QUANTITY | VALUE | TOLERANCE |
|---|---|---|---|
| LM348M OP-AMP | 16 | | |
| LM741CN | 1 | | |
| RESISTOR R1 | 288 | 10kΩ | ±1% |
| RESISTOR R2 | 50 | 3kΩ | ±1% |
| RESISTOR R2/R3 | 4 | 5kΩ | ±1% |
| CAPACITOR | 50 | 10n | +80/−20% |