A

Project Report

On

“Design and ASIC Implementation of Floating Point FIR FILTER”

Submitted to

RAJIV GANDHI UNIVERSITYOFKNOWLEDGE TECHNOLOGIES RK VALLEY

in partial fulfillment of the requirements for the awardof the Degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

Y.RAJESH R200012

P.KARTHIK R200036

Under the Guidance of

K.HARINATHA REDDY sir, Assistant Professor RGUKT RK VALLEY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

RAJIV GANDHI UNIVERSITY OFKNOWLEDGE TECHNOLOGIES

RKVALLEY,KADAPA 2024-2025

RAJIV GANDHI UNIVERSITY OFKNOWLEDGE TECHNOLOGIES
RK VALLEY,KADAPA 516330

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DECLARATION

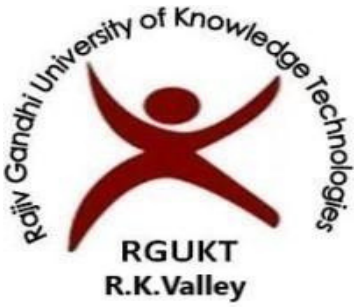
We hereby declare that the project report entitled **"Design and ASIC Implementation of Floating Point FIR FILTER for ECG Signal DeNoising "** submitted to the Department of **ELECTRONICS AND COMMUNICATION ENGINEERING** in partial fulfillment of requirements for the award of the degree of **BACHELOR OF TECHNOLOGY**.

This project is the result of our own effort and that it has not been submitted to any other University or Institution for the award of any degree or diploma other than specified above.

Y. RAJESH (R200012)
P. KARTHIK (R200036)

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES
RK VALLEY,KADAPA 516330

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING



CERTIFICATE

This is to certify that the project report entitled
Design and ASIC Implementation of Floating Point FIR FILTER a bonaflde record of project work done and submitted by

Y. RAJESH R200012
P. KARTHIK R200036

for the partial fulfment of the requirements for the award of B.Tech.
Degree in **ELECTRONICS AND COMMUNICATION ENGINEERING,** RGUKT,RK VALLEY.

Project Guide
K.HARINATHA REDDY
Assistant Professor
RGUKT,RK VALLEY
Kadapa-516330

Head Of the Department
Dr.B V.SUDHAKAR REDDY
Assistant Professor
RGUKT,RK VALLEY
Kadapa-516330

ACKNOWLEDGEMENT

We thankful to our guide **K.HARINATHA REDDY,**for his valuable guidance and encouragement. His helping attitude and suggestions have helped us in the successful completion of the project.

We would like to express our gratefulness and sincere thanks to **Dr**.**B V.SUDHAKAR REDDY**, Head of The Department of ELECTRONICS AND COMMUNICATION ENGINEERING, for his kind help and encouragement during the course of our study and in the successful completion of the project work. Successful completion of any project cannot be done without proper support and encouragement. We sincerely thanks to the management for providing all the necessary facilities during the course of study.

We would like to thank our parents and my friends, who have the greatest contributions in all our achievements, for the great care and blessings in making us successful in all our endeavors.

Y. RAJESH (R200012)
P.KARTHIK (R200036)

Abstract

The accurate processing of biomedical signals, particularly ECG (Electrocardiogram) data, requires high-precision numerical computation to preserve clinically significant features while effectively removing noise. This project focuses on the design of a Floating-Point FIR Filter for ECG Signal Denoising, integrating both digital signal processing techniques and Verilog- based hardware architecture.

A 10-tap FIR filter was designed using IEEE 754 single-precision floating-point arithmetic to ensure wide dynamic range, numerical stability, and minimized quantization error. Various windowing techniques—including Hamming, Hanning, Rectangular, and Blackman windows—were applied to the ideal impulse response to generate coefficients that offer different trade-offs in ripple suppression and transition sharpness. These coefficients were first validated in MATLAB and then exported in IEEE 754 hexadecimal format for hardware implementation.

Custom floating-point adder and multiplier modules were developed with a 6-stage pipelined architecture, supporting normalization, rounding, and exception handling. These modules were integrated into the FIR filter design to compute real-time convolution efficiently. The complete system was implemented and simulated in Verilog, with MATLAB used as a reference for evaluating filtered ECG signals in both time and frequency domains.

Results demonstrate that the floating-point FIR filter effectively suppresses high-frequency noise, power-line interference, and white noise while preserving critical ECG waveform characteristics. The close match between MATLAB and Verilog outputs verifies the correctness of the hardware design. Overall, this project establishes a robust and efficient hardware-friendly framework for ECG signal denoising, suitable for deployment in ASIC/FPGA-based biomedical devices.

Contents

List of Figures

List of Tables

Chapter1

Introduction

In digital signal processing (DSP), especially in biomedical applications like ECG analysis, precise numerical computation is crucial.ECG signals often exhibit low amplitude and are vulnerable to various noise sources such as baseline drift and high-frequency interference. Implementing filters to clean these signals requires arithmetic that can handle small variations without introducing significant quantization errors.

Floating-point representation offers a robust solution by allowing dynamic scaling of values, similar to scientific notation. It represents real numbers using three fields: sign, exponent, and mantissa. This flexibility enables accurate processing of both very small and large values without complex scaling logic.

In our project, we designed a 10-tap FIR filter using Verilog to reduce noise in ECG signals. By using IEEE 754 single-precision floating-point arithmetic, the filter achieves high precision and numerical stability, which is critical for preserving the integrity of medical data.Floating-point operations simplify coefficient handling,avoid overflow issues, and improve overall filter performance.

The IEEE 754 standard ensures consistent behavior across ASIC /FPGA implementations and toolchains, making it ideal for portable, high-reliability designs. In this context, floating-point arithmetic is not only practical but essential for building effective and accurate ECG signal enhancement systems in hardware.

Let's dive into floating point representation using IEEE 754 single-precision format!

1.1 Floating Point Representation

Floating-point numbers are one possible way of representing real numbers in binary format. The IEEE 754 standard defines two primary floating-point formats: the *Binary Interchange Format* and the *Decimal Interchange Format*. This paper focuses exclusively on the single-precision normalized binary interchange format.

32-bit Single-Precision Floating-Point Format (IEEE 754):

S EEEEEEEE MMMMMMMMMMMMMMMMMMMMMMM

Where:

- S – Sign bit (1 bit)
- E – Exponent (8 bits)
- M – Mantissa/Fraction (23 bits)

Figure 1.1: IEEE 754 floating-point single precision format

M The IEEE 754 standard represents floating-point numbers using a combination of sign, exponent, and mantissa (fraction). The general formula used to compute the actual decimal value from the stored binary fields is:

$$\text{Value} = (-1)^{\text{sign}} \times (1.\text{mantissa}) \times 2^{\text{exponent}-\text{bias}}$$

- $(-1)^{\text{sign}}$: Determines the sign of the number.
  - If sign = 0 , the number is positive.
  - If sign = 1, the number is negative.
- (1.mantissa): This is the normalized significand (also called mantissa or fraction).
  - IEEE 754 assumes a leading 1 in normalized numbers, which is not stored but implicitly used.
    The mantissa represents the fractional part after the binary point.

- $2^{\text{exponent}-\text{bias}}$: The exponent field is stored with a bias to allow both positive and negative exponents.
  - For single precision, the bias is 127.
  - The actual exponent is calculated as: exponent – bias

- An extra bit is added to the mantissa to form what is called the significand, which used to represent 1 in (1.mantissa) and that extra bit also called as Hidden bit.
  - If the exponent is greater than 0 and less than 255, and there is a 1 in the MSB of the significand, then the number is said to be a normalized number.

1.1.1 Special Cases in IEEE 754 Floating-Point Representation

The IEEE 754 standard defines several special cases to handle unique or extreme values in binary floating-point formats. These are essential for ensuring correctness and stability in numerical computations.

- Zero (± 0):
  - Exponent: All bits = 0
  - Mantissa: All bits = 0
  - The sign bit distinguishes between $+0$ and 0.
  - Used to represent a true zero value with a specific direction in limits or underflows.
- Infinity ($\pm$):
  - Exponent: All bits = 1 (255 in single precision)
  - Mantissa: All bits = 0
  - Represents results of overflow (e.g., dividing by zero or exceeding max range).
  - The sign bit determines $+$or .

- NaN (Not a Number):
  - Exponent: All bits $= 1$
  - Mantissa: Any non-zero value
  - Indicates undefined or unrepresentable results (e.g., $0/0$, sqrt(1)).
  - Can be quiet NaN (qNaN) or signaling NaN (sNaN), based on the MSB of the mantissa.
- Denormalized Numbers (Subnormals):
  - Exponent: All bits $= 0$
  - Mantissa: Non-zero
  - These represent values very close to zero that cannot be normalized.
  - Useful for maintaining *gradual underflow*, ensuring smoother transitions near zero.

1.1.2 Fixed point vs Floating point

Table 1.1: Comparison between Floating-Point and Fixed-Point Representation

Feature	Floating-Point	Fixed-Point
Dynamic Range	Very wide, due to scalable exponent	Limited, fixed by bit width and scaling
Precision Handling	High precision across a range of magnitudes	High precision only in a limited range
Scaling	Handled automatically via exponent	Must be manually designed and managed
Complexity of Arithmetic	More complex hardware and logic	Simpler and faster arithmetic units
Use in DSP and Scientific Computation	Preferred for algorithms like FIR, FFT, LMS due to accuracy	used in simple, low-power systems with limited resources
Standardization	Governed by IEEE 754 standard	No universal standard; application-dependent
Resource Usage (Hard-ware)	Requires more area, power, and latency	Efficient in area and power
Portability	High portability across platforms (CPU, GPU, FPGA)	Portability depends on scaling conventions

1.2 Hardware Realization

1.2.1 Floating Point Adder

The algorithm for floating-point addition is explained through the flow chart in Figure 1.2. While adding two floating-point numbers, two cases may arise:

- Case I: When both numbers have the same sign (i.e., both are either positive or negative). In this case, the MSB (sign bit) of both numbers is either 0 or 1.
- Case II: When the numbers have different signs (i.e., one is positive and the other is negative). In this case, the MSB of one number is 0 and the other is 1.

1.2.1.1 When both numbers are of same sign

1. Enter two numbers N_1 and N_2. Let E_1, S_1 and E_2, S_2 represent the exponent and significand of N_1 and N_2, respectively.
2. Check if $E_1 = 0$ or $E_2 = 0$.
   - If yes, set the hidden bit of N_1 or N_2 to zero.
   - Otherwise, compare exponents: if $E_2 > E_1$, swap N_1 and N_2. If $E_1 > E_2$, no swap is needed.
3. Compute the exponent difference $d = E_1 - E_2$.
   - If $d = 0$, no shift is required.
   - If $d > 0$, shift S_2 right by d bits and fill the left-most bits with zeros (including hidden bit).
4. Add the shift amount d to the exponent of N_2:

   $$E_2 = E_2 + d$$

   Now, both exponents are equal, and the operands are normalized.
5. Check the signs of N_1 and N_2.
   - If the signs are the same, proceed to addition.
6. Add the significands including the hidden bit (24-bit addition):

   $$S = S_1 + S_2$$

6. Check for carry-out from the significand addition.

   - If carry exists, increment the exponent by 1.
   - Then shift the result right by 1, set MSB of S to 1, and drop the LSB.

7. If no carry-out occurs, retain the current exponent as the final exponent.

8. Determine the sign bit of the result. It will be the same as the sign of N_1 or N_2, depending on which one had the greater magnitude.

9. Assemble the result into 32-bit IEEE 754 format by excluding the 24 th hidden bit from the significand.

1.2.1.2 When both numbers are of different sign

Step 1, 2, 3 4 are the same as done in case I.

5. Check if N_1 and N_2 have different signs. If "Yes";

6. Take 2's complement of S_2 and then add it to S_1, i.e.,

$$S = S_1 + (\text{2's complement of } S_2)$$

7. Check if there is carry-out in the significand addition. If yes, discard the carry and shift the result to the left until there is a '1' in the MSB. Also, count the number of shifts, say "z".

8. Subtract "z" from the exponent value of either E_1 or E_2. Now, the original exponent becomes $E_1 - z$. Append z zeros at the LSB of the significand.

9. If there is no carry-out in Step 6, then MSB must be '1'.

   In this case, simply replace S with its 2's complement.

10. The sign of the result (MSB) is the same as the sign of the larger number —i.e., MSB of either N_1 or N_2.

11. Assemble the result into 32-bit IEEE 754 format, excluding the 24 th bit of the significand (i.e., the hidden bit).

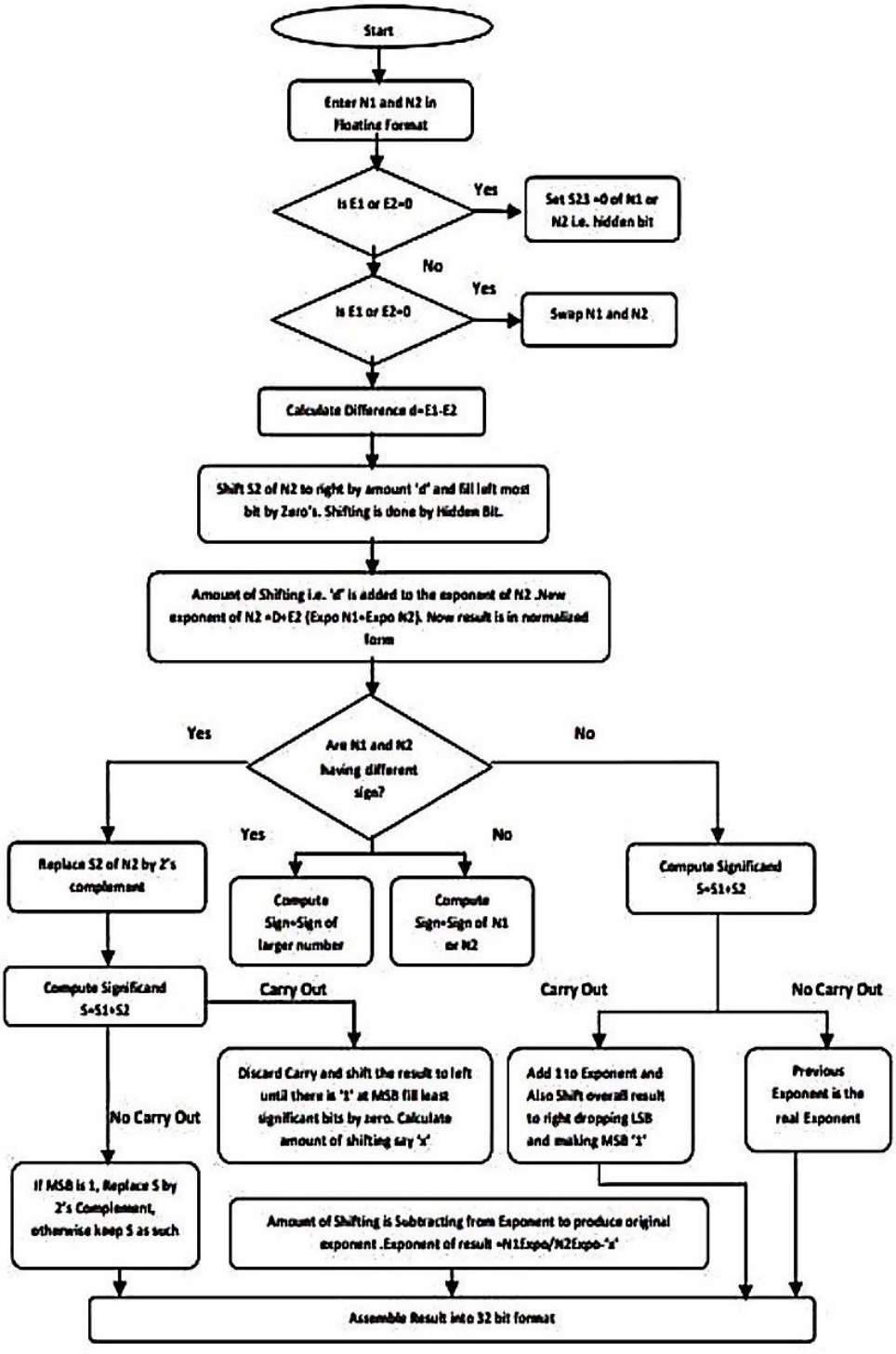


Figure 1.2: Flowchart of Floating Point Addition

1.2.2 Floating Point Multiplier

The algorithm for floating-point multiplication is illustrated through the flow chart in Figure ??. Let N_1 and N_2 be normalized operands represented by their respective sign bits, significands (mantissas), and exponents: $Sign_1$, M_1, E_1 and $Sign_2$, M_2, E_2.

Floating-point multiplication is generally performed in the following steps:

1. Multiply significands, add exponents, and determine the sign:
   - $M = M_1 \times M_2$
   - $E = E_1 + E_2 - \text{Bias}$
   - $\text{Sign} = \text{Sign}_1 \oplus \text{Sign}_2$
2. Normalize the Mantissa M: Shift the result left or right as needed so that it fits the normalized form. Adjust the exponent E accordingly.
3. Round the result: Apply rounding to ensure the result fits within the available bits for the mantissa (23 bits in single precision).
4. Check for exceptions (overflow/underflow): Determine and handle any exceptional cases, such as overflow or underflow. Set the appropriate flags or values.

Sign Bit Calculation: The result sign is negative if only one of the operands is negative, and this is obtained using the XOR of the two sign bits.

Exponent Addition: Exponents are added using an unsigned adder and then the bias (127 for single precision) is subtracted from the result:

$$E = E_1 + E_2 - 127$$

Significand Multiplication: The unsigned significands (including the implicit leading 1) are multiplied. The product is referred to as the intermediate product (IP). This multiplication is performed on 24-bit operands (23-bit mantissa $+1$ implicit bit).

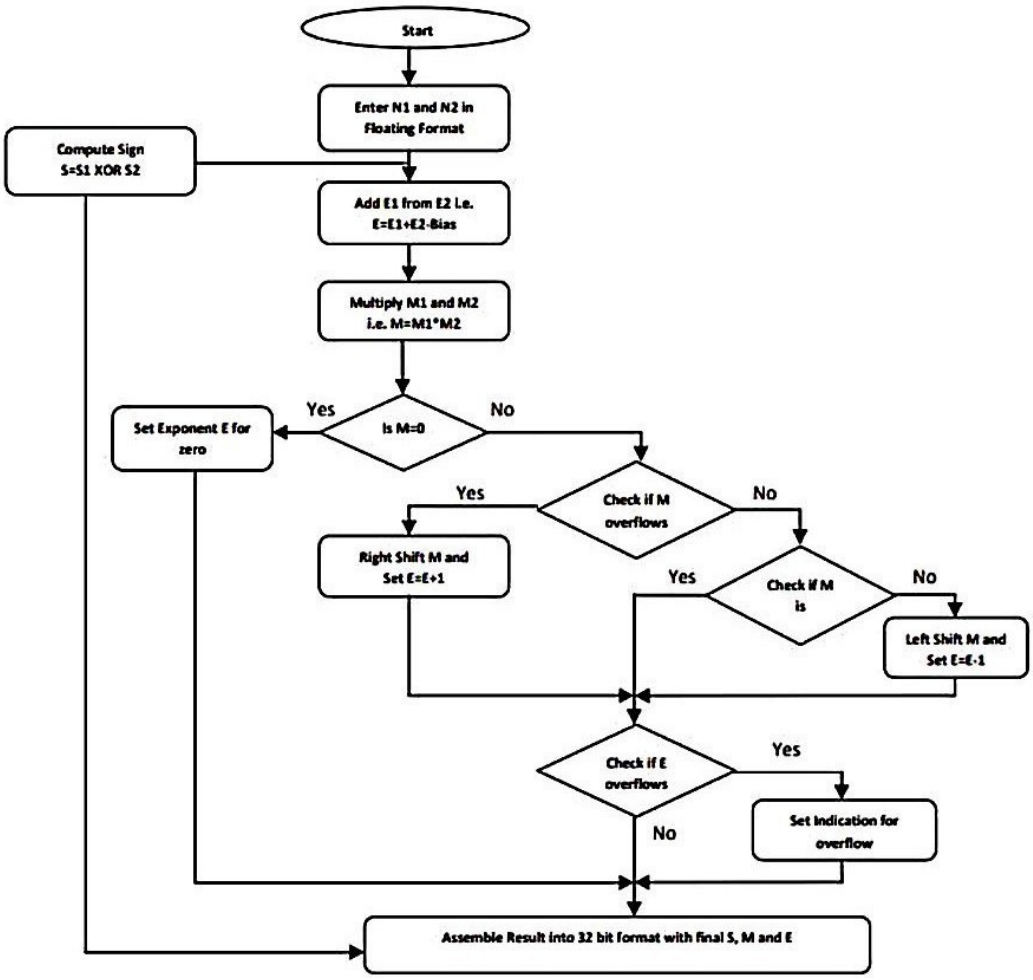


Figure 1.3: Flowchart of Floating Point Multiplicaton

1.3 Design Flow

The design of both the floating-point adder and floating-point multiplier has been implemented using a 6-stage pipelining approach for the adder and a 6-stage pipelining approach for the multiplier. Inputs are provided directly in IEEE 754 single-precision format, and the respective operations are carried out as per the algorithms described above. The primary difference in the number of pipeline stages arises from the way the output is assigned. In the case of the adder,the final output is generated using a combinational assign statement. However, in the multiplier, the output is produced using a sequential alwaysblock,which introduces an additional clockcycle.This results in one extra pipeline stage for the multiplier compared to the adder.

The fp adder dsmodule employs a 6-stage pipelined architecture efficiently perform floating-point addition in compliance with the IEEE 754 single-precision standard. Each pipeline stage is designed to break the addition process into well- defined operations,enabling high-throughput arithmetic in digital signal processing (DSP) applications and ASIC-based systems.The stages are as follows:

1.3.1 Pipelining Strategy in Floating Point Addition

1. Stage 1 – Input Preparation and Swapping: The input operands (n1 , n2)are compared based on their exponent values. Ife2 > e1 , the operands are swapped to ensure proper alignment. The sign, exponent, and mantissa fields are extracted for further processing.

2. Stage 2 – Normalization and Alignment: The hidden bit is added to both mantissas depending on whether the numbers are normalized or subnormal. The difference between the exponents is computed, and the smaller mantissa is right shifted accordingly to align both operands for addition or subtraction.

3. Stage 3 – Operation Determination and Execution: Based on the signs of the operands, either addition or subtraction is performed on the aligned mantissas.Ripple-carry adder modules are used to compute both signed and unsigned results. The appropriate result is selected depending on the input signs.

4. Stage 4 – Normalization of the Result: If the result of the significand operation requires normalization, the leading 1 is identified, and the result is left-shifted.

The exponent is adjusted accordingly to preserve numerical accuracy.

5. Stage 5 – Rounding: The result is rounded to the nearest even value using guard,round, and sticky bits to meet IEEE 754 requirements.

6. Stage 6 – Final Assembly: The final result is assembled in IEEE 754 format using the computed sign, adjusted exponent, and the normalized significand. A special case is handled explicitly when the two operands are equal in magnitude but have opposite signs (resulting in zero).

1.3.2 Pipeline Structure of Floating Point Multiplication

The fp mutiplication module im[plements single precision floating point multiplication using a 6 stage pipelining architecture. This structure enables high throughput by overlapping multiple computation stages over several clockcycles. Below is the breakdown of each stage:

1. Operand Extraction: The 32-bit floating-point inputs n1 and n2 are decomposed as
    - Sign bits: sign1,sign2
    - Exponents: e1,e2,m2
    - Mantissas: m1
2. Significand Preparation: Hidden bits are appended:
    - Normalized numbers get a leading 1.
    - Denormalized numbers get a leading 0.

    This yields the 24-bit significands s1 and s2.
3. Multiplication Core: This stage performs:
    - Sign calculation via XOR: sign1 ^ sign2
    - Exponent addition and bias removal: e1 + e2 - 127
    - Significand multiplication: s = s1 * s2 (48 bits)
4. Normalization: Based on the MSBs of the product s[47:46], the result is normalized:
    - If 10 or 11 : Already normalized.
    - If 01 : Shift left by 1 and adjust exponent.
    - Else: Use a loop to find leading 1 and shift appropriately.
5. Rounding Preparation:

The upper 23 bits of s norm(leading bits)are taken as the base significand.The bit at position 23 (s norm[23]) is used as 11the rounding indicator (s norm inter), and the lower 23 bits (s norm[22:0]) are OR-ed to determine if any remaining bits are set, producing the round signal.Rounding is performed by adding 1 to the leading bits if both s norm inter and round are high, implementing a round-to-nearest approach. This is achieved by:

```
leading_bits <= s_norm[46:24];
round<= |s_norm[22:0];
s_norm_inter <= s_norm[23];
rounded_result <= leading_bits + (s_norm_inter & round);
```

6. Final Assembly and Exception Handling: The final 32-bit result is assembled using:

   • Calculated sign

   • Adjusted exponent

   • Rounded significand

Special cases are handled per IEEE 754:

   • NaN: Exponent = 255, Mantissa 0

   • Infinity: Exponent = 255, Mantissa = 0

   • Zero: Exponent = 0, Mantissa = 0

   • Overflow: Result set to Infinity

   • Underflow: Result set to Zero (gradual underflow not handled)

Thus, the overall pipeline latency of this module is 6 clock cycles,making it well-suited for high-speed DSP applications like FIR filtering and ECG signal processing,where repeated floating-point multiplications are needed.

1.4 Simulation Results

1.4.1 Floating-Point Adder Simulation

Figure 1.4 illustrates the simulation waveform for the fp adder ds module. This module performs floating-point addition on 32-bit IEEE 754 numbers using a 6-stage 12 pipelined design. The data shown in Table 1.2 includes both the hexadecimal input values and their corresponding decimal values. Each input pair is added, and the result is reflected in the y out signal. The table validates the accuracy of the adder even under edge cases such as:

- Addition of a positive and negative number of equal magnitude, producing zero.
- Addition of negative and positive fractional values.
- Positive overflow and full-precision rounding (e.g., 1 +1 =2.0).

These results indicate that both the floating-point multiplier and adder modules are functionally correct and conform to IEEE 754 standards,making them suitable for integration into larger DSP systems such as FIR filters.

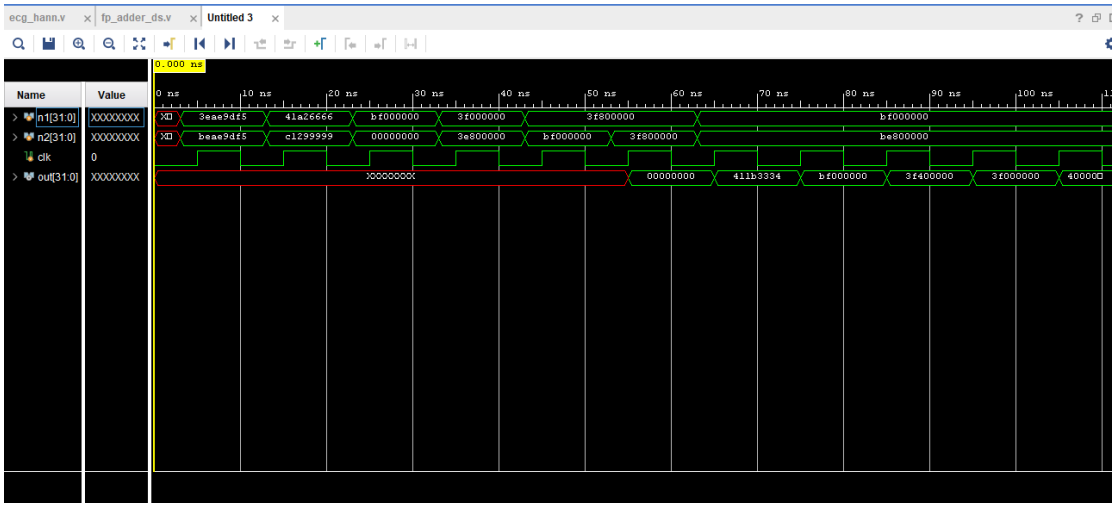


Figure 1.4: Floating point Adder Verilog Output Waveform

Table 1.2: Floating-Point Adder Testbench Results

Time (ns)	n1 (Hex)	n2 (Hex)	y out (Hex)	n1 (Decimal)	n2 (Decimal)	y out (Decimal)
65	41A26666	C1299999	411BB394	+20.3	−10.6	+9.7
75	BF000000	00000000	BF000000	-0.5	0	−0.5
85	3F000000	3E800000	3F400000	+0.5	+0.25	+0.75
95	3F8000000	BF000000	3F000000	+1	+0.5	+0.5
105	3F800000	3F800000	40000000	+1	+1	+2.0
115	BF000000	BE800000	BF400000	-0.5	-0.25	-0.75

1.4.2 Floating-Point Multiplier Simulation

Figure 1.5 shows the waveform output from the simulation of the fp multiplication module. The module performs multiplication on IEEE 754 singles precision floating 13point inputs (n1andn2) and produces a 32-bit floating-point output. Table 1.3 presents the values observed during the simulation.Each row corresponds to a specific clock cycle where valid input-output transactions occur. The hexadecimal representations of the inputs and output are shown,along with their decimal equivalents. These results confirm the correct operation of the multiplier,as seen by the expected product values.

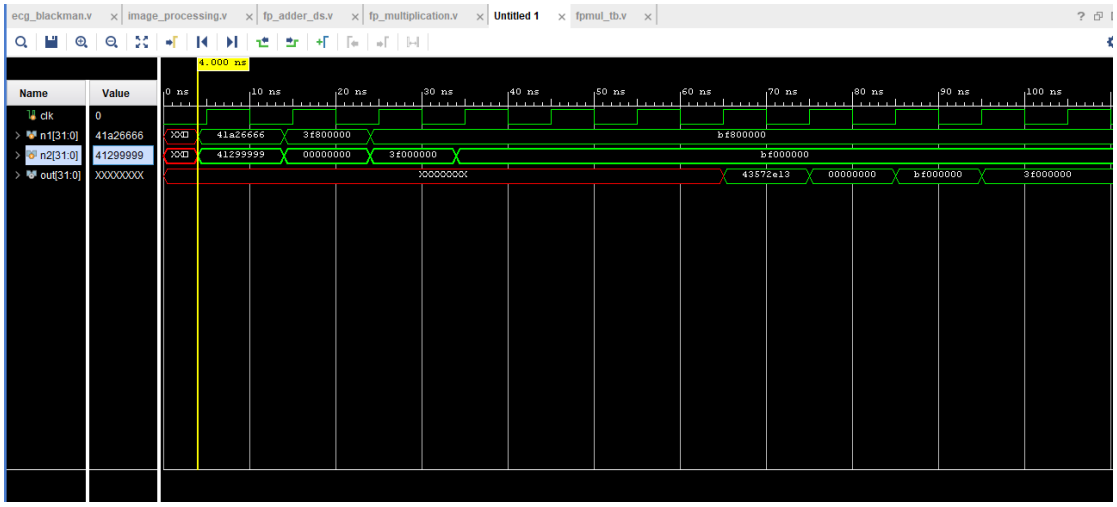


Figure 1.5: Floating point Multiplier Verilog Output Waveform

Time (ns)	n1 (Hex)	n2 (Hex)	out (Hex)	n1 (Decimal)	n2 (Decimal)	out (Decimal)
60	41A26666	41299999	43572E13	20.300001	10.599999	215.37001
70	3F800000	00000000	00000000	1	0	0
80	BF800000	3F000000	BF000000	-1	0.5	-0.5
90	BF800000	BF000000	3F000000	-1	-0.5	0.5

Table 1.3: Floating Point Multiplier Testbench Results

Chapter 2

Filters

Filters are fundamental components in Digital Signal Processing (DSP), used to enhance or suppress certain parts of a signal. In simple terms, a filter selectively allows certain frequency components of a signal to pass through while blocking others.

There are two main categories of filters:

- Analog filters (operate on continuous signals)
- Digital filters (operate on discrete-time signals)

In DSP, we mainly deal with digital filters, which are implemented using algorithms and mathematical operations on sampled data. These filters are widely used in applications such as audio processing, communication systems, and biomedical signal analysis.

For biomedical applications like ECG signal processing, filters play a crucial role in removing noise and artifacts (such as baseline drift, muscle noise, or powerline interference), thus improving the clarity and reliability of the signal for diagnosis.

2.1 Types of Digital Filters

In DSP, filters are categorized based on their frequency response characteristics. Common types include:

1. Low-Pass Filter (LPF): Allows low-frequency components to pass while attenuating higher frequencies. It is commonly used to remove high-frequency noise from signals.

2. High-Pass Filter (HPF): Passes high-frequency components and attenuates low-frequency components. It is used to eliminate slow-varying trends or DC offsets from signals.

3. Band-Pass Filter (BPF): Allows a specific range of frequencies to pass while rejecting both lower and higher frequencies outside this band.

4. Useful in applications where a particular frequency range carries the useful information.

5. Notch Filter (Band-Stop): Removes a narrow frequency band and passes all other frequencies. Typically used to eliminate specific interference frequencies,such as power line noise at 50 Hz or 60 Hz.

6. FIR (Finite Impulse Response) Filters:

These filters have a finite-duration response to an impulse input. They are inherently stable and can be designed to have a linear phase, which preserves the waveform shape.

7. IIR (Infinite Impulse Response) Filters: These filters have feedback elements,resulting in an impulse response that theoretically continues indefinitely. They are computationally efficient but can introduce phase distortion and may require stability checks.

2.2 FIR (Finite Impulse Response) Filter

A Finite Impulse Response (FIR) filter is a type of digital filter whose impulse response settles to zero in a finite number of sample intervals. FIR filters are widely used in Digital Signal Processing (DSP) due to their inherent stability, linear phase characteristics, and predictable performance.

The output y[n] of an FIR filter is given by the convolution sum:

$$y[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k]$$

Where:

- $x[n]$: Input signal
- $y[n]$: Output signal
- $h[k]$: Filter coefficients (impulse response)

- N: Number of taps (filter order +1)

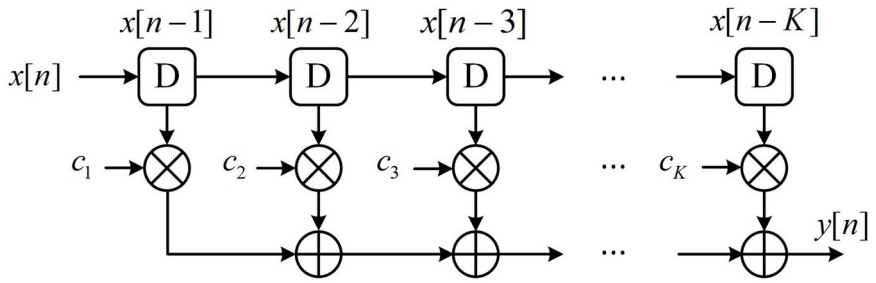


Figure 2.1: K-tap FIR Filter

2.2.1 Why FIR over IIR?

FIR (Finite Impulse Response) filters are often preferred over IIR (Infinite Impulse Response) filters in digital signal processing due to the following reasons:

• Linear Phase Response:

FIR filters can be designed to have a linear phase, which means they do not distort the phase of different frequency components in the signal.This is important in applications where the shape of the signal must be preserved.

• Guaranteed Stability:

FIR filters are always stable because they do not use feedback elements in their structure.

• Design Flexibility:

They offer more precise control over the filter's frequency response, especially when using windowing techniques.

• Simpler Implementation in Hardware:

FIR filters require only multipliers and adders, making them easier to implement on digital hardware like FPGAs and DSP processors. While IIR filters are more computationally e fficient (requiring fewer coefficients),FIR filters are generally chosen when signal integrity, phase accuracy, and stability are more critical than computational speed.

2.3 Windowing Methods in Filters

When designing a FIR filter, we often start from the ideal impulse response (e.g., sinc for low-pass), which is infinite in length.We need to truncate this to make it implementable in hardware or software.

However, truncation causes discontinuities, which lead to:

- Gibbs phenomenon (ripples in passband/stopband)
- Poor frequency selectivity (leakage of unwanted frequencies)

When you change the window function, you directly affect the frequency response of the FIR filter, which in turn impacts the output signal.

1.Transition Band Width

The main lobe width of the window determines the transition sharpness from passband to stopband.

Effect on output:

- Narrow main lobe → Sharper transition → Less distortion near cutoff.
- Wide main lobe → Smoother transition → Some leakage between bands.

2. Ripple and Stopband Attenuation

Side lobes in the window's frequency response introduce ripples in the filter's passband and stopband.

Effect on output:

- High side lobes (e.g., Rectangular) → Ripple and leakage.
- Low side lobes (e.g., Blackman) → Clean filtering.

3. Noise Suppression and Leakage

Better windows suppress more noise and unwanted frequencies due to improved stopband attenuation.

Comparison of Common Window Functions

1. Rectangular Window (No Tapering)

$$w[n] = 1, \quad 0 \leq n \leq N$$

Main lobe width: Narrowest

Side lobe level: Very high (approx. -13 dB)

Transition band: Sharp

- Very sharp cutoff
- High ripple in stopband (Gibbs phenomenon)
- Not suitable for clean filtering

2. Hanning (Hann) Window

$$w[n] = 0.5 \left(1 - \cos\left(\frac{2\pi n}{N}\right)\right)$$

Main lobe width: Moderate

Side lobe level: Approximately -31 dB

- Smooth roll-off
- Good general-purpose filtering
- Reduces Gibbs phenomenon better than Rectangular

3. Hamming Window

$$w[n] = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N}\right)$$

Main lobe width: Similar to Hanning

Side lobe level: Approximately -41 dB

- Better stopband attenuation than Hanning
- Slightly more passband ripple
- Good for audio or ECG signals

4. Blackman Window

$$w[n] = 0.42 - 0.5\cos\frac{2\pi n!}{N} + 0.08\cos\frac{4\pi n!}{N}$$

Main lobe width: Wide

Side lobe level: Very low (approx. -58 dB)

- Excellent ripple suppression
- Wide transition band
- Best when output must be clean (e.g., biomedical)

Window	Main Lobe Width	Side Lobe Attenuation	Ripple	Transition Sharpness	Output Quality
Rectangular	Narrowest	-13 dB	High	Sharpest	Noisy (leaky)
Hanning (Hann)	Moderate	-31 dB	Low	Moderate	Clean
Hamming	Moderate	-41 dB	Very Low	Moderate	Very Clean
Blackman	Widest	-58 dB	Minimal	Smoothest	Cleanest

Table 2.1: Comparison of Window Functions in FIR Filter Design

2.4 Designing of FIR Filter

2.4.1 Design specifications for FIR Filter

Design Specification 1: FIR Filter Design Task

Design a 5-tap low-pass FIR filter whose sampling frequency is 500 Hz with a cut- off frequency of 150 Hz. Plot its magnitude and phase responses. Generate a random composite signal of 100 Hz and 150 Hz with a sampling frequency of 500 Hz.Use this designed filter to process the generated signal."

Design Specification 2: ECG Signal Filtering Task

"ECG signals capture the electrical activity of the heart and can be recorded fairly easily with surface electrodes on the limbs or chest. The file "bob ecg.dat" contains an ECG signal sampled at 200 Hz.However, the signal is corrupted by multiple types of noise, including baseline wander, 60 Hz power-line interference, and additive white noise, making it difficult to interpret. Your task is to apply digital signal processing techniques to clean up the ECG signal and restore it to its original form as much as possible using MATLAB."

1. Load the ECG signal from the file “bob ecg.dat” into a MATLAB array. Plot the raw, noisy ECG signal in the time domain. Ensure the plot includes appropriate labels.

2. High-frequency noise (white noise) is present in the ECG signal. Design a high-pass filter manually without using any built-in filter functions, implementing it
from scratch. Choose an appropriate cutoff frequency to eliminate high-frequency noise while preserving the ECG signal. Plot the magnitude and phase response of the designed high-pass filter. Apply the filter to the noisy ECG signal. Plot the noisy ECG signal and the signal after high-pass filtering in the same graph both in the time and frequency domain to analyze the reduction in white noise.”

2.4.2 Design and Simulation in MATLAB

1. **Composite Signal Filtering**:

To filter a noisy composite signal composed of 100 Hz and 150 Hz components, a 10-tap low-pass FIR filter was designed using both manual and built-in methods.

The sampling frequency was set to 500 Hz, and the cutoff frequency was 150 Hz.

The built-in fir1 function with a Hamming window was used for standard filter design.

For manual implementation, an ideal impulse response was computed and multiplied with the Hamming window to generate the FIR coefficients.The designed filter was applied to the composite signal using both the filter() function and a manual convolution loop.Both the waveform are plotted in Figure 2.3 and Figure 2.2.

2. **ECG Signal Filtering**:

To clean a real-world ECG signal (loaded from bob ecg.dat) sampled at 200 Hz,a low-pass FIR filter was manually designed using di fferent windowing techniques including Rectangular,Hanning,Hamming and Blackman windows.

2.4.2.1 Simulation Results

FIR Filter on Composite Signal Filtering

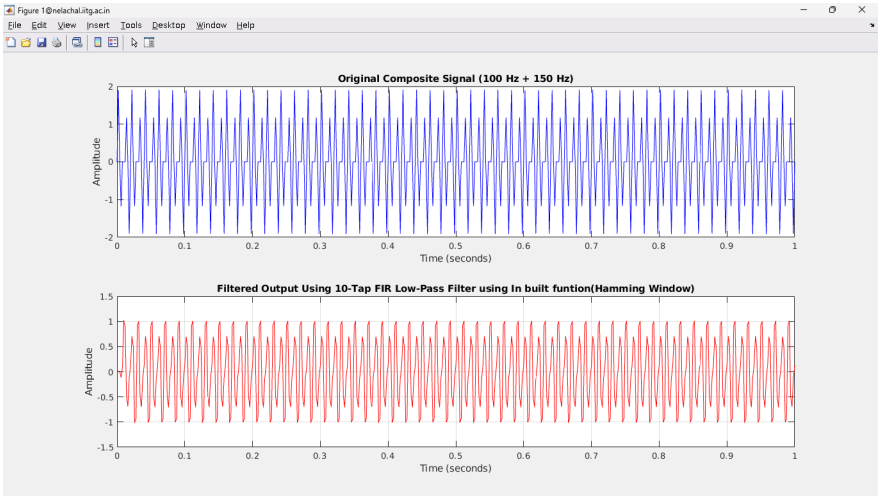


Figure 2.2: FIR Filter using Built-In FIR Function (Hamming Window)

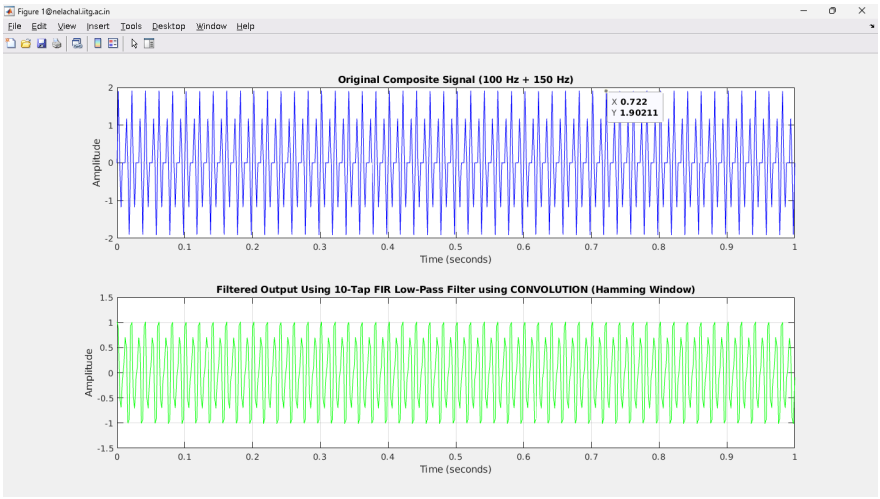


Figure 2.3: FIR Filter using Convolution (Hamming Window)

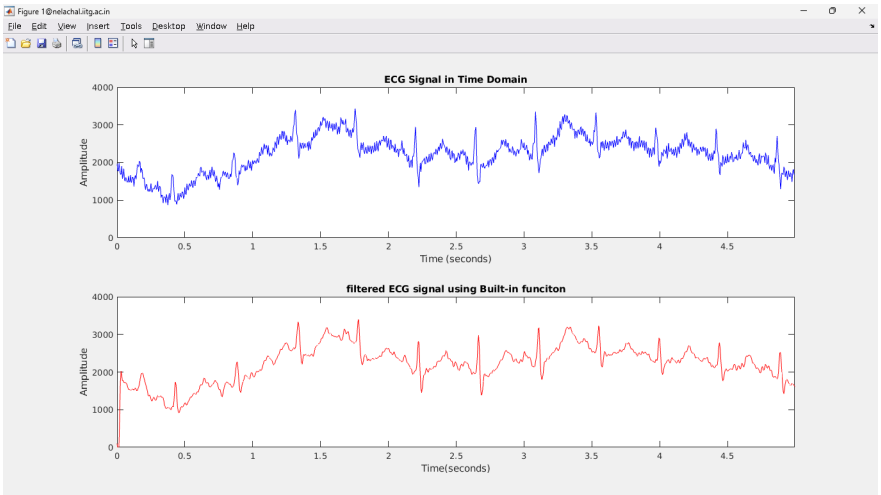


Figure 2.4: ECG Filtering using Buit-In FIR Function (Rectangular Window)

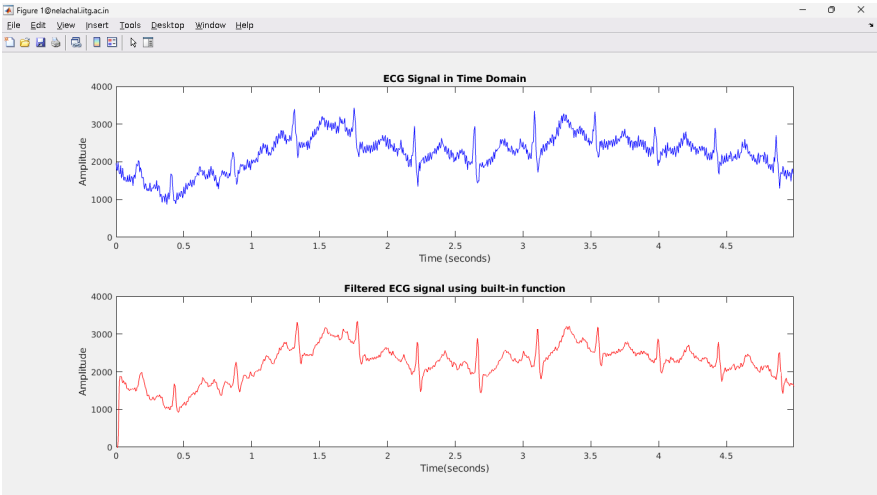


Figure 2.5: ECG Filtering using Buit-In FIR Function (Hanning Window)

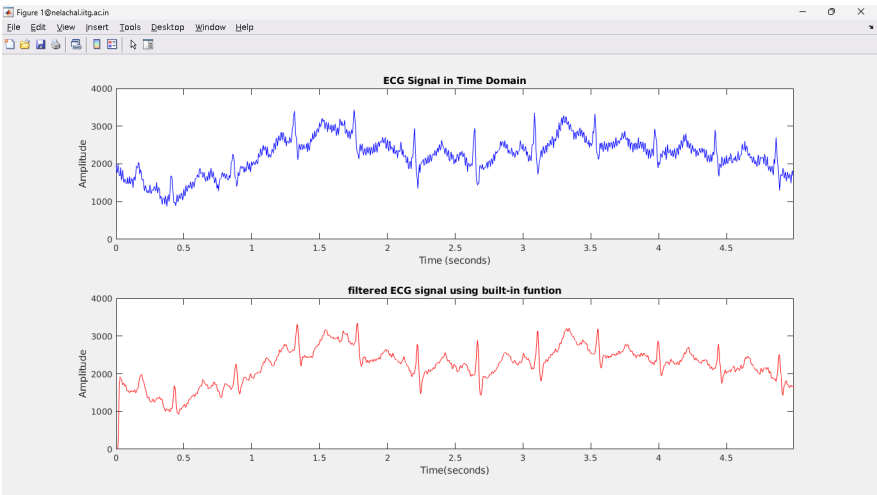


Figure 2.6: ECG Filtering using Buit-In FIR Function (Hamming Window)

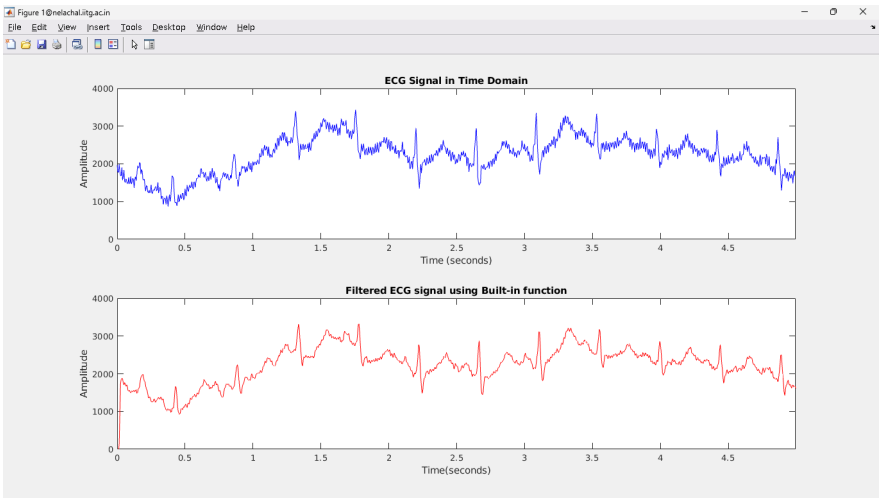


Figure 2.7: ECG Filtering using Buit-In FIR Function (Blackman Window)

Chapter 3

Hardware Implementation

3.1 Hardware Implementation of FIR Filter

3.1.1 Custom Floating-Point Arithmetic Modules

Implemented IEEE 754 compliant:

- Floating-point Adder(fp adder_ds)
- Floating-point Multiplier(fp multiplication)

Used 6-stage and 6-stage pipelined architectures respectively.

Key Functionalities

- Normalization
- Rounding (with guard, round, sticky bits)
- Sign, exponent, and significand handling

Custom Rounding Logic (Verilog)

```
leading_bits <= s_norm[46:24];

round <= |s_norm[22:0];

s_norm_inter <= s_norm[23];

rounded_result <= (leading_bits + (s_norm_inter & round))
```

3.1.2 10-Tap FIR Filter Architecture

Filter Equation

$$y[n] = h[0]x[n] + h[1]x[n-1] + \cdots h+[9]x[n-9]$$

Design Highlights

• Implemented using data broadcast structure (inputs are broadcasted to all taps).

• • Each tap uses:

• – One instance of fp multiplication

• – One pipeline stage of fp adder dsusing data broadcast structure (inputs are broadcasted to all taps).

Register Structure

10 registers ($x[n]$ to $x[n-9]$) were used to store delayed inputs using a shift-register mechanism.

Coefficient Handling

• Filter coefficients were generated in MATLAB using windowed sinc functions(e.g., Hamming, Rectangular).

• files using the following MATLAB command:

Exported as .mem

• h_hex = dec2hex(typecast(single(h_data), 'uint32'));

• These were read into Verilog using $readmemh.h

Pipelining

Each multiplier and adder was pipelined with 6 and 5 cycles delay respectively.

• Total pipeline depth:

Pipeline Depth =Multiplier Delay+ (N× Adder Delay) =6+9× (6) =60clock cycles

• Registers were added at intermediate outputs to balance stages and maintain timing closure.

The output was further processed in Verilog using the same coe fficients and input signal represented in IEEE 754 hexadecimal format.

The filtered outputs from both MATLAB and Verilog were visualized and verified in the time domain, confirming the effectiveness and consistency of the filter design..

3.2 Hardware Implementation of ECG Signal Filtering using 10-Tap FIR Filter

1. Design Goals

- Clean the ECG signal from "bob ecg.dat"
- Build a 10-tap FIR filter in Verilog with support for various window functions.
- Implement floating-point multiplication and addition using IEEE 754 format.
- Interface with Vivado using .memfiles containing hexadecimal representations of signal and data.

2. Verilog Architecture Breakdown

1)Input/Output ports

2) Shift Registers

3) Floating point multipliers

4) Pipelining

5) Floating point Adders

3. Floating-Point Support

• fp multiplication : IEEE 754-compliant floating-point multiplier with a 6-cycle latency.

• fp adder ds: Custom 6-cycle pipelined floating-point adder.

Each input sample and coe fficient is represented in 32-bit IEEE 754 format and provided to the Verilog design using.mem files generated from MATLAB.

3. Coefficient Generation & Windowing

MATLAB was used to compute the ideal impulse response and apply the following windowing functions:

• hamming()

• hanning()

• blackman()

• rectwin()

The final filter coefficients were computed using: $h = h_{\text{ideal}} \cdot w$

where w is the window vector. The coefficients were normalized and written into a .mem file using the following MATLAB script:

```
h_hex = dec2hex(typecast(single(h), 'uint32'));
fid = fopen('coeff_hamming.mem', 'w');
fprintf(fid, '%s\n', h_hex');
fclose(fid);
```

Window Function Effects in Hardware

• Rectangular: Offers a sharp transition but introduces large ripples in the stopband, which degrades signal clarity.

• Hamming/Hanning: Provide balanced side-lobe suppression with acceptable transition width, suitable for biomedical filtering.

• Blackman: Yields excellent ripple suppression in the stopband, ideal for high- precision filtering, though it spreads the main lobe.

Each set of coefficients was tested in hardware by changing the .mem file input to the Verilog design and observing the output in both time and frequency domains.

3.2.1 Simulation Results

The output from the Verilog is obtained by feeding the coe fficients and input signal represented in IEEE 754 hexadecimal format using .mem files. By feeding this output again to MATLAB using .txt files to plot reconstructed waveforms as well as magnitude response, so that can visualize and verify in the time domain, confirming the effectiveness and consistency of the filter design.

For comparison of output of Verilog y_out , I have plotted the hex values of the filters with di fferent windowing methods(Hamming, Hanning,..).

Figure 3.1: ECG Filtering using Verilog

MATLAB Output

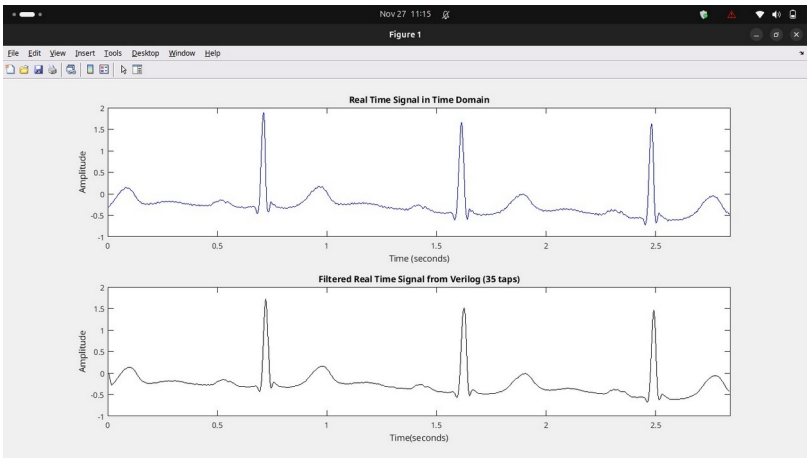


Figure 3.2: ECG Filtering using Verilog (Rectangular Window)

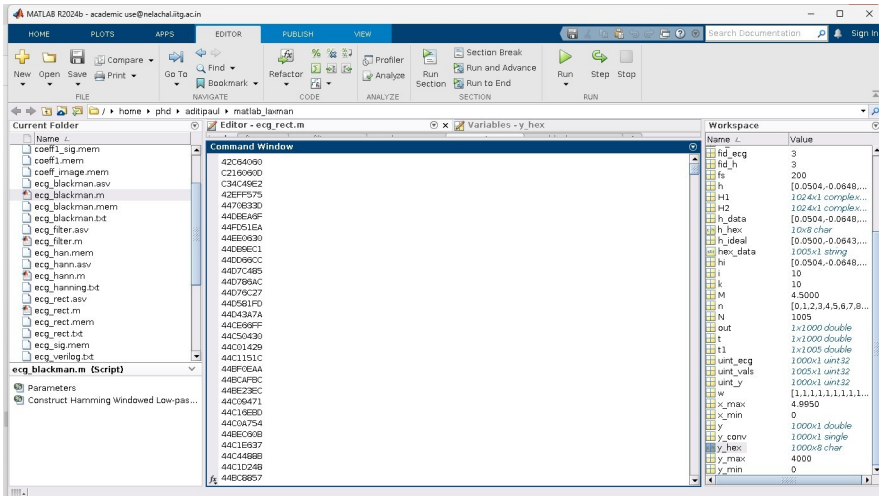


Figure 3.3: ECG Filtering Rectangular Window HEX Values

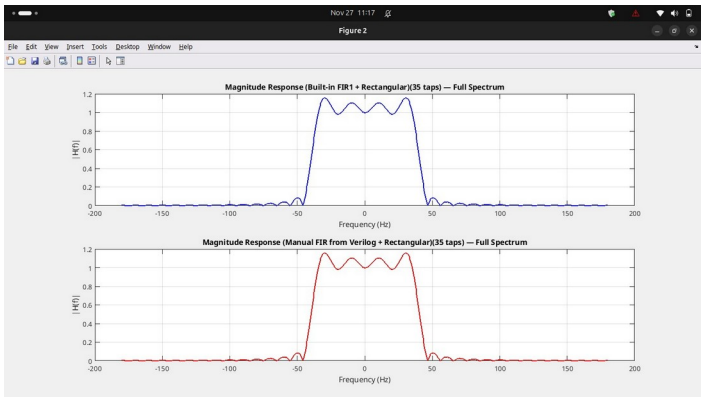


Figure 3.4: ECG Filtering Magnitude Response (Rectangular Window)

2.**Hanning window**

Verilog Output

Figure 3.5: ECG Filtering using Verilog

MATLAB Output

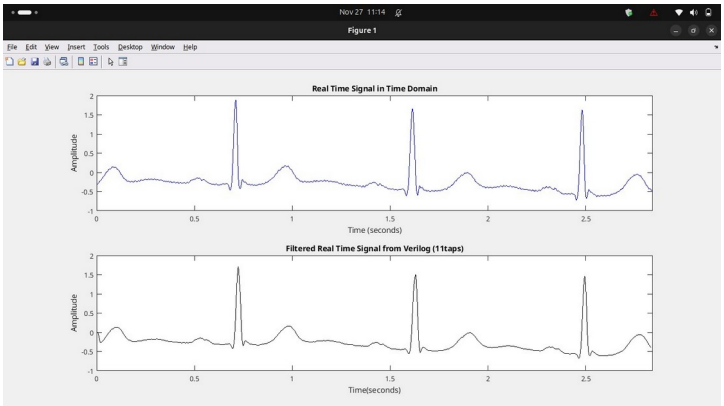


Figure 3.6: ECG Filtering using Verilog (Hanning Window)

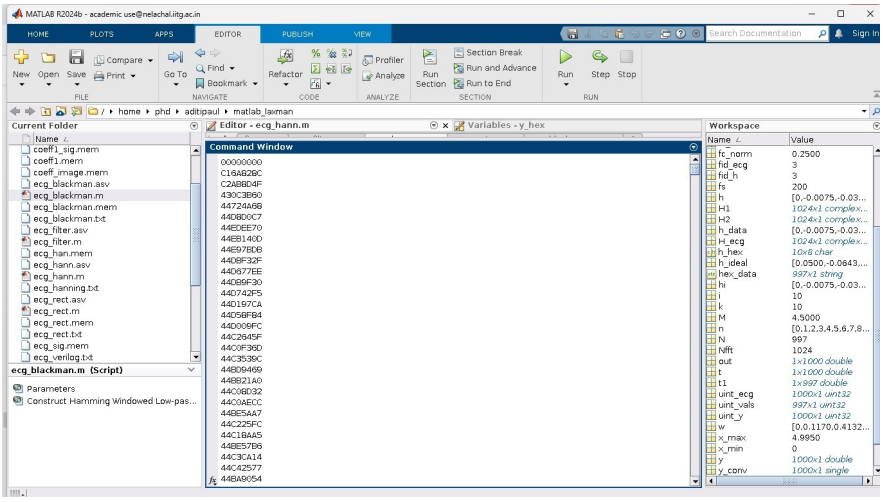


Figure 3.7: ECG Filtering Hanning Window HEX Values

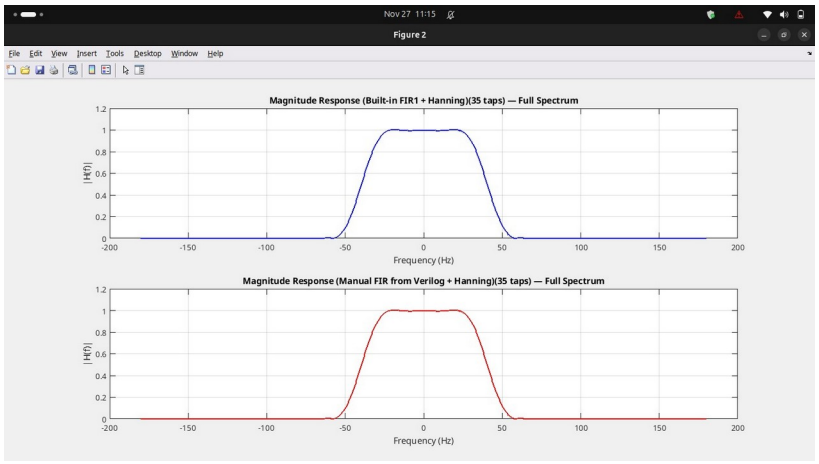


Figure 3.8: ECG Filtering Magnitude Response (Hanning Window)

3.**HammingWindow**

Verilog Output

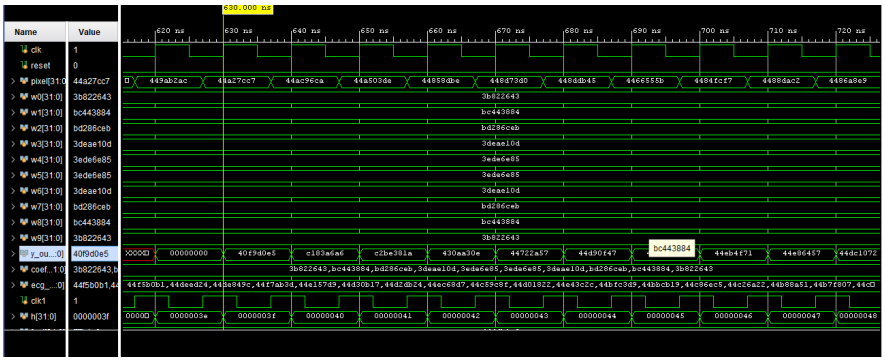


Figure 3.9: ECG Filtering using Verilog

MATLAB Output

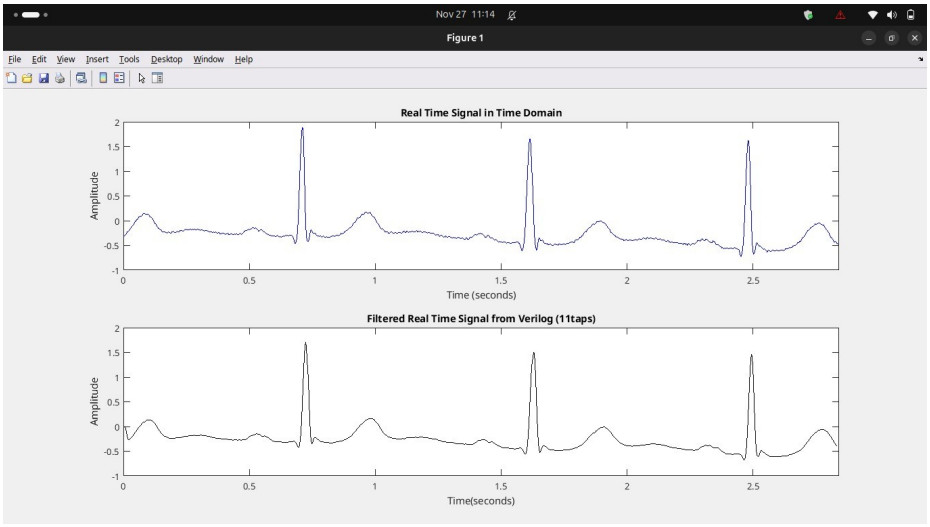


Figure 3.10: ECG Filtering using Verilog (Hamming Window)

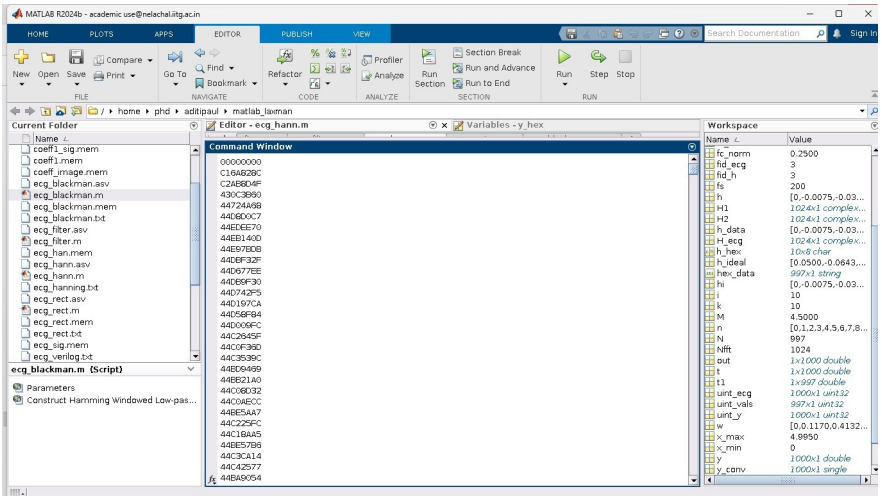


Figure 3.11: ECG Filtering Hamming Window HEX Values

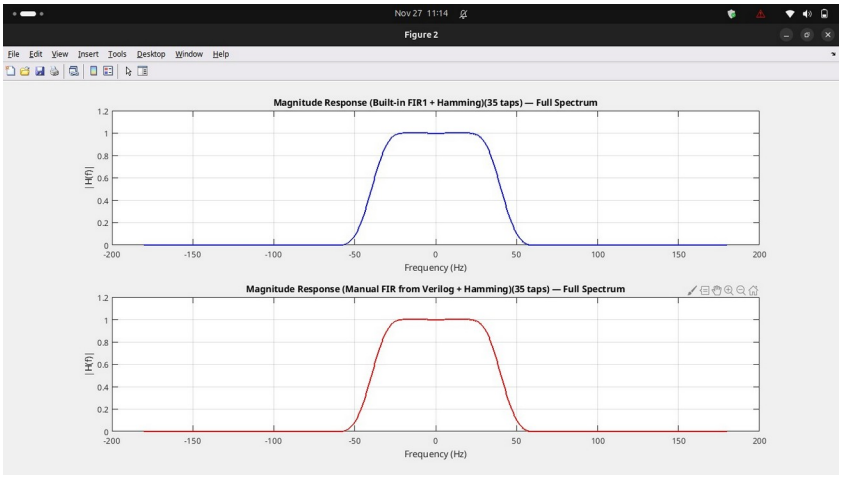


Figure 3.12: ECG Filtering Magnitude Response (Hamming Window)

4.Blackman

Window Verilog

Output

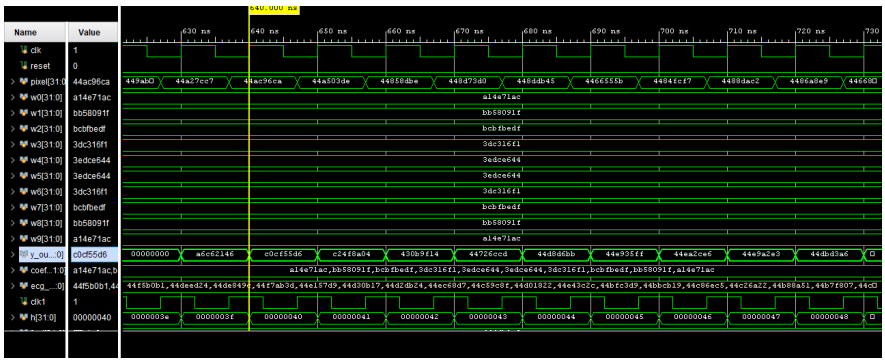


Figure 3.13: ECG Filtering using Verilog

MATLAB Output

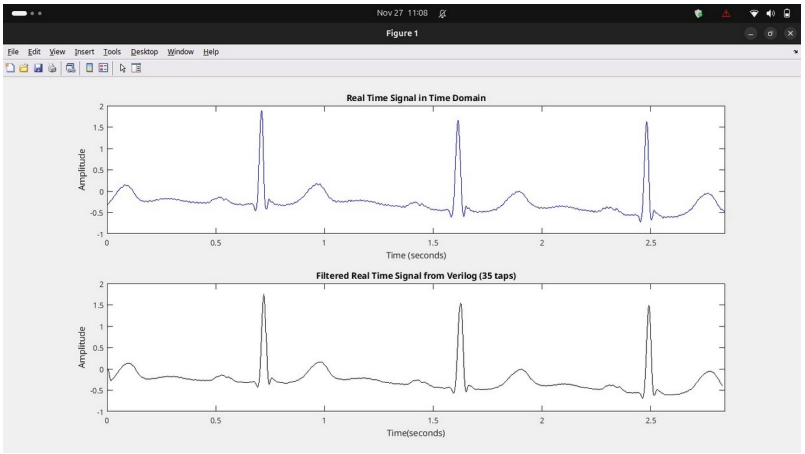


Figure 3.14: ECG Filtering using Verilog (Blackman Window)

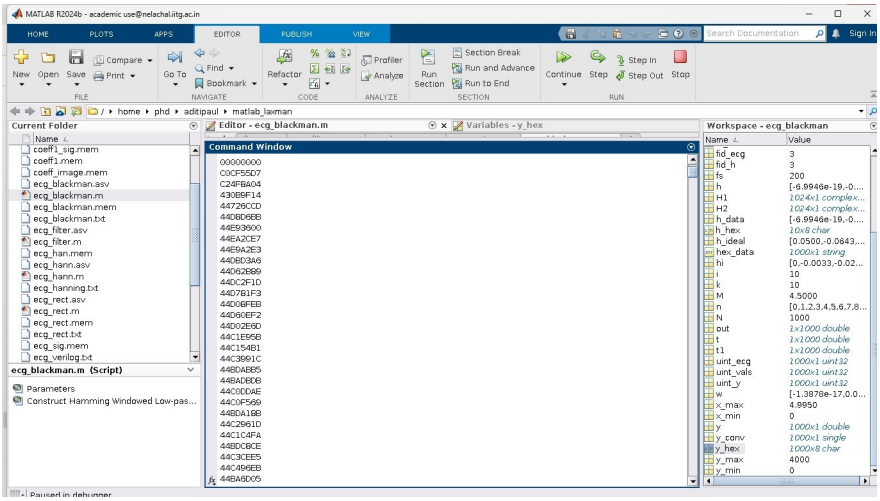


Figure 3.15: ECG Filtering Blackman Window

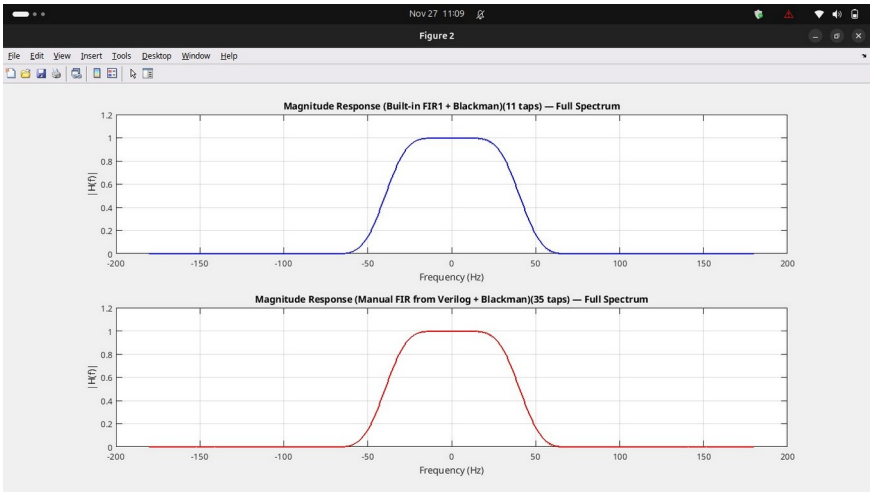


Figure 3.16: ECG Filtering Magnitude Response (Blackman Window)

References

[1] AbrarAbrar Ahmed Chhipa, et al., Adaptive Neuro-fuzzy Inference System Based Maximum Power Tracking Controller for Variable Speed WECS, 2021Energies,Vol. 14, No. 19, pp.6275. https://doi.org/10.3390/en14196275

[2] Abrar Ahmed Chhipa,et al.,MPPT optimisation techniques and power electronics for renewable energy systems: energy systems, 2022 Int. J. Swarm Intelligence wind and solar (IJSI), Vol. 7, No. 2. https://doi.org/10.1504/IJSI.2021.10041290

[3] Abrar Ahmed Chhipa and Vinod Kumar,using Dual DC-Microgrid Voltage Regulation Active Bridge based SVR,2021 IEEE 7th InternationalConference on Electrical Energy Systems (ICEES), 2021, pp. 490-495, doi:10.1109/ICEES51510.2021.9383696.

[4] Abrar Ahmed Chhipa and Vinod Kumar,Grid-Connected PV System Power Forecasting Using Nonlinear Autoregressive Exogenous Model, The 2nd Electric Power and Renewable Energy Conference (EPREC-2021), 28-30 May, 2021. (In Process)

[5] @online Raspberry pi, https://www.raspberrypi.org/ Online; accessed 10-June-2019

[6] HU, Yun Chao, et al., Mobile edge computing?A key technology towards 5G,ETSI white paper, 2015, vol. 11, no 11, p. 1-16.

Conclusion

This project successfully demonstrated the complete design, development, and hardware implementation of a Floating-Point FIR Filter for ECG Signal Denoising, combining digital signal processing principles with ASIC-oriented hardware design. By utilizing IEEE 754 single-precision floating-point arithmetic, the system achieved high numerical accuracy, a wide dynamic range, and effective handling of small biomedical signal variations, which are essential for reliable ECG analysis.

10-tap FIR filter was implemented with multiple windowing techniques—Rectangular, Hanning, Hamming, and Blackman—to analyze their effects on filtering performance. MATLAB simulations confirmed the expected behavior of each window function, while the hardware implementation in Verilog replicated the same results, validating the functional correctness and consistency of the design across platforms.

Custom developed floating-point adder and multiplier modules, each employing a pipelined architecture, enabled high-speed execution and efficient resource utilization. The integration of these modules within the FIR filter allowed real-time convolution processing suitable for ASIC/FPGA- based biomedical applications. The close correlation between MATLAB and Verilog outputs further verifies the accuracy of the rounding, normalization, and exception-handling mechanisms implemented in hardware.

Overall, the project demonstrates that floating-point FIR filtering is highly effective for ECG signal enhancement, providing clean, noise-suppressed waveforms while preserving essential cardiac features. The architecture developed here lays a strong foundation for future advancements such as longer tap filters, adaptive filtering, low-power ASIC optimization, or real-time embedded deployment in wearable healthcare devices.