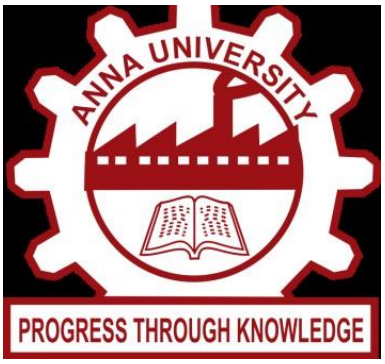


CENTRE FOR WIRELESS SYSTEM DESIGN (C-WiSD)

&

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

REPORT

SUBMITTED BY:

L.R.SOWMIDRA

FOR THE PROJECT WORK EXECUTED DURING THE FOUR WEEKS

(20.05.2024 TO 14.05.2024)

INTERNSHIP PROGRAM TITLED

SEMICONDUCTOR IC DESIGN AND TESTING

(Realization of Complex number multiplier for FFT computation)

IN

CENTRE FOR WIRELESS SYSTEM DESIGN (C-WiSD)

ANNA UNIVERSITY CHENNAI : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this Report titled **"Realization of Complex number multiplier for FFT computation"** is the bonafide work of **L.R.Sowmidra** who carried out the project work as part of their Summer Internship programme titled "Semiconductor IC Design and Testing" at Centre for Wireless System Design (C-WiSD) during the period 03.06.2024 to14.06.2024.

Dr. J. Dhurga Devi
Project Co-ordinator &
Associate Professor,
Department of Electronics and Communication Engineering
College of Engineering, Guindy
Anna University, Chennai -600 025

Dr. M. Meenakshi
Director,
Centre for Wireless System Design,
College of Engineering, Guindy
Anna University, Chennai - 600 025

I. OBJECTIVE

The Fast Fourier Transform (FFT) is a crucial algorithm in the realm of signal processing, mathematics, and various fields of science and engineering. Its primary objective is to efficiently compute the Discrete Fourier Transform (DFT) and its inverse. To understand the objective of FFT, we must delve into the broader context of Fourier analysis and computational efficiency.

Fourier Analysis and the Discrete Fourier Transform (DFT)

Fourier analysis is a powerful mathematical tool used to decompose functions or signals into sinusoidal components of different frequencies. It is based on the idea that any periodic function can be represented as a sum of sine and cosine functions with different frequencies and amplitudes. The Fourier Transform extends this concept to non-periodic signals or functions, transforming them from the time (or spatial) domain to the frequency domain. For discrete, finite-length signals or data sequences, the Discrete Fourier Transform (DFT) is used. The DFT converts a sequence of N complex numbers (representing samples of a signal) into another sequence of N complex numbers, which represent the signal in terms of its frequency components

Need for FFT:

The Fast Fourier Transform (FFT) is essential because it significantly improves computational efficiency for analyzing signals and data. While the Discrete Fourier Transform (DFT) requires \(O(N^2)\) operations, the FFT reduces this to \(O(N \log N)\), making it practical for large datasets and real-time applications. This efficiency is crucial in fields like signal processing, image processing, and telecommunications, where fast and efficient data analysis is needed.

Objective of FFT

The objective of the Fast Fourier Transform (FFT) is to efficiently compute the Discrete Fourier Transform (DFT) and its inverse, reducing computational complexity from \(O(N^2)\) to \(O(N \log N)\). This efficiency enables faster real-time data analysis and processing in applications such as signal processing, communications, and image analysis.

1. Digital Signal Processing (DSP): FFT is extensively used in DSP for applications like filtering, spectral analysis, convolution, correlation, and modulation/demodulation.

2. Telecommunications: FFT is used in modems, coding schemes (e.g., OFDM - Orthogonal Frequency Division Multiplexing), and spectrum analysis.

3. Audio and Image Processing: FFT is employed in audio spectrum analysis, equalization, noise reduction, and image compression.

4. Scientific Computing: FFT finds applications in solving partial differential equations, solving linear systems, and simulating physical systems.

5. Biomedical Engineering: FFT is used in analysing EEG (electroencephalogram) signals, ECG (electrocardiogram) signals, and other biomedical data.

How FFT Achieves Efficiency

FFT achieves its efficiency primarily through clever use of the divide-and-conquer strategy, exploiting the periodicity properties of sinusoidal functions and symmetry properties of the DFT. By decomposing the DFT recursively into smaller DFTs, FFT dramatically reduces the number of arithmetic operations required.

Implementations and Variants

There are several FFT algorithms, each optimized for different scenarios (e.g., real data FFTs, power-of-2 FFTs, mixed-radix FFTs). Popular FFT implementations include the Cooley-Tukey algorithm, Radix-2 FFT, Radix-4 FFT, and mixed-radix FFTs.

II. SYSTEM FUNCTION DESCRIPTION

BRIEF DESCRIPTION

The system functionality of the Fast Fourier Transform (FFT) is fundamental to understanding its role and importance in signal processing and various scientific and engineering applications. Here's a detailed description of its system functionality:

1. Signal Transformation from Time Domain to Frequency Domain

The primary function of FFT is to transform a signal from the time domain into the frequency domain. In the time domain, signals are represented as amplitude versus time (or space). FFT converts this representation into the frequency domain, where signals are expressed as amplitude versus frequency. This transformation allows analysis of the signal's frequency components, revealing the presence and strength of various frequencies within the signal.

2. Computational Efficiency The computational efficiency of the Fast Fourier Transform (FFT) is a key advantage, as it significantly reduces the number of operations required to compute the Discrete Fourier Transform (DFT).

Specifically:

- **DFT Complexity**: Direct computation of the DFT requires \(O(N^2)\) operations, where \(N\) is the number of data points.

- **FFT Complexity**: The FFT reduces this to \(O(N \log N)\) operations, making it much faster, especially for large datasets. This improvement in efficiency allows the FFT to handle large-scale computations quickly and is essential for real-time applications and processing complex signals.

3. Decomposition and Butterfly Operations

FFT operates based on the principle of decomposition and butterfly operations. The algorithm decomposes the DFT computation into smaller DFTs, recursively applying the same technique until reaching base cases where the DFT is directly computed. Butterfly operations are fundamental steps in FFT, involving combining results from smaller DFTs to compute larger DFTs efficiently.

4. Applications in Digital Signal Processing

In digital signal processing (DSP), FFT finds widespread use in various applications:

- Spectrum Analysis: Analysing the frequency content of signals, essential for tasks like identifying dominant frequencies, detecting noise, and filtering.

- Filter Design and Analysis: Designing and analysing digital filters, including low-pass, high-pass, band-pass, and notch filters, by examining their frequency responses.

- Convolution: Efficient computation of convolution, which is fundamental in applications such as image processing, radar processing, and communications.

5. Applications Beyond DSP Beyond DSP,

FFT is essential in:

- Telecommunications: Techniques like Orthogonal Frequency Division Multiplexing (OFDM) rely on FFT for dividing a high-speed data stream into multiple lower-speed data streams for transmission.

- Medical Imaging: Analysing signals from medical instruments like MRI and EEG machines to understand brain activity or diagnose medical conditions.

- Scientific Research: Used extensively in fields such as physics, astronomy, and geophysics for analysing signals from instruments and simulations.

6. Implementation Variants and Considerations

FFT algorithms vary based on factors such as:

- Data Types: Handling real and complex data types efficiently.

- Radix: FFT algorithms are categorized by their radix (base of the decomposition), such as Radix-2, Radix-4, and mixed-radix FFTs, each suited to different signal lengths and computational environments.

- Memory and Computational Efficiency: Optimizing memory usage and computational speed depending on the hardware platform and application requirements.

7. Practical Considerations When implementing FFT:

• Windowing: Applying window functions to mitigate spectral leakage and improve frequency resolution.

• Precision: Choosing the appropriate precision (e.g., floating-point or fixed-point arithmetic) based on the dynamic range and accuracy requirements of the application.

III. BLOCK DIAGRAM

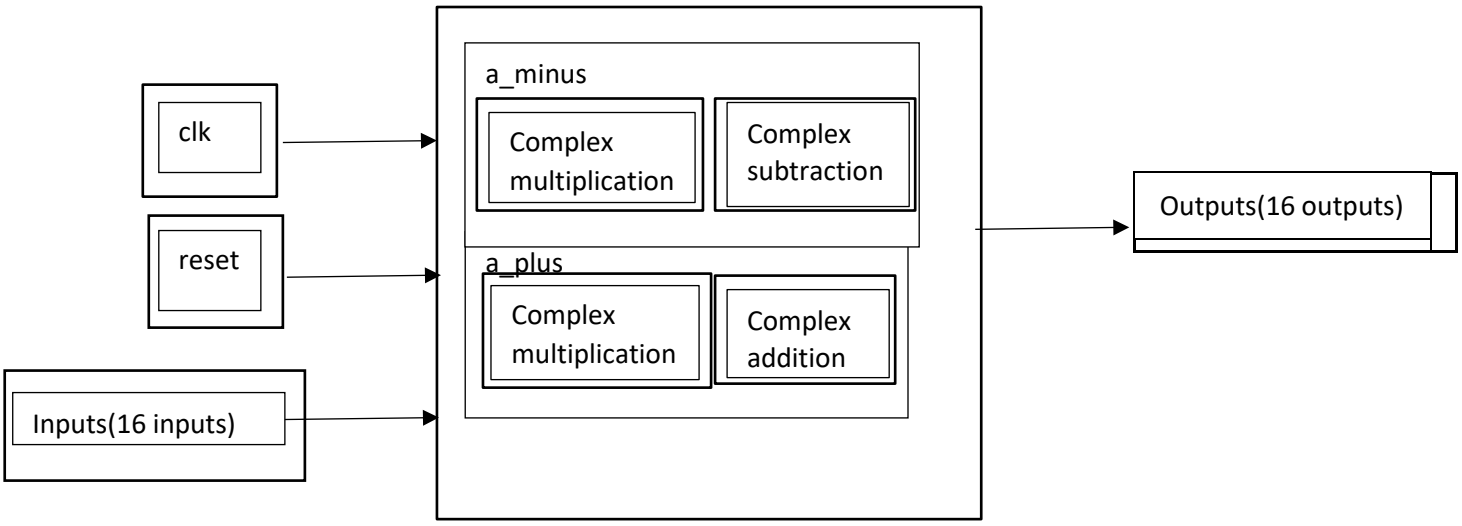


Fig: Block diagram of FFT

(1) **Input:** Begin with an input sequence (ai_real ai_imag to pi_real and pi_imag) inputs are in bit-reversal order.

(2) Using ai_plus and the complex multiplication and complex addition.

(3) Using ai_minus and the complex multiplication and complex subtraction

(4) In complex addition the values of first 2 inputs to be added.

(5) In complex subtraction the values of first two inputs are subtracted.

(6) In complex multiplication values obtained from the ai_plus and ai_minus are multiplied with the twiddle factor.

Perform complex multiplication (a + bi) * (c + di) = (ac - bd) + (ad + bc)

(7) **Twiddle factor:** Twiddle factor values are declared in the test bench code.

(8) **Output:** The output of first stage is a1_real and a1_imag to p1_real and p1_imag next stage is a2_real and a2_imag to p2_real and p2_imag and last stage is a3_real and a3_imag to p3_ real and p3_imag.

Butterfly diagram :

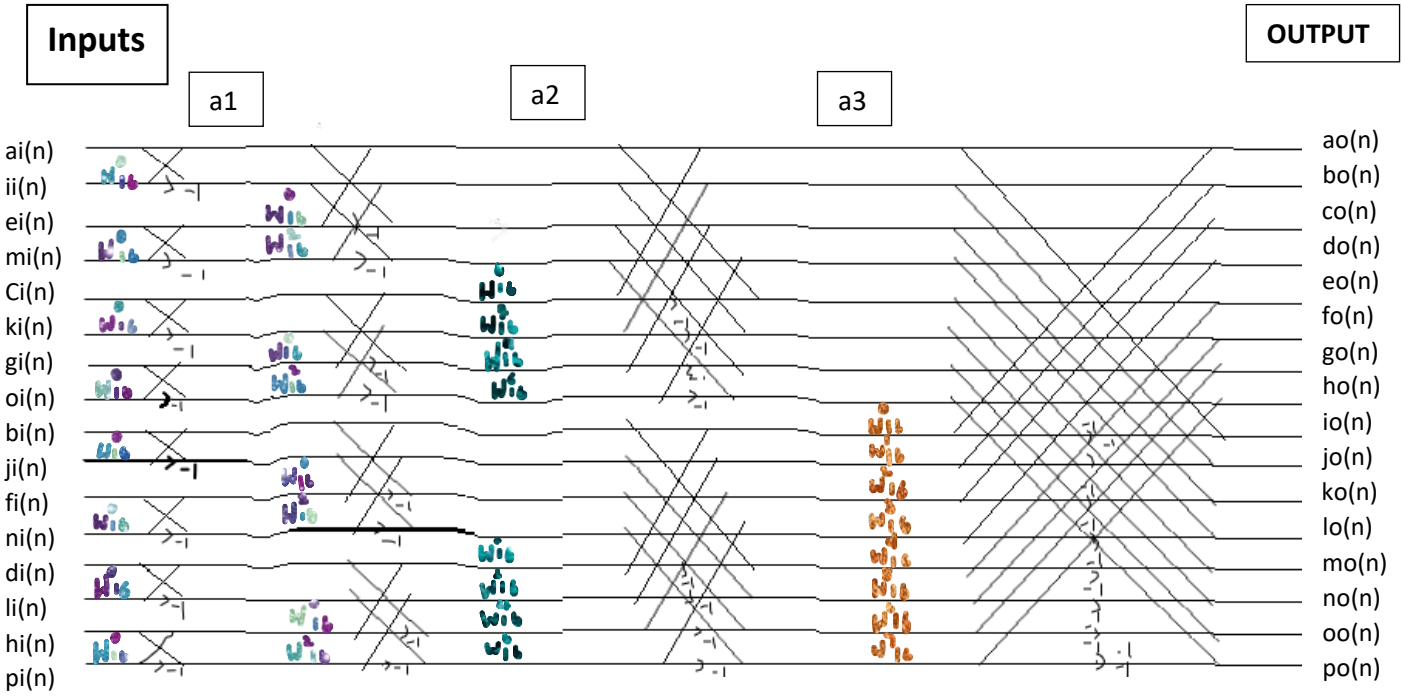


Fig 3: Flow diagram or butterfly diagram of FFT when N=16

Inputs: ai_real,ai _iag to pi_real,pi_imag .

Twiddle factors: W^{0}_{16}, W^{1}_{16}, W^{2}_{16}, W^{3}_{16}, W^{4}_{16}, W^{5}_{16}, W^{6}_{16}, W^{7}_{16}.

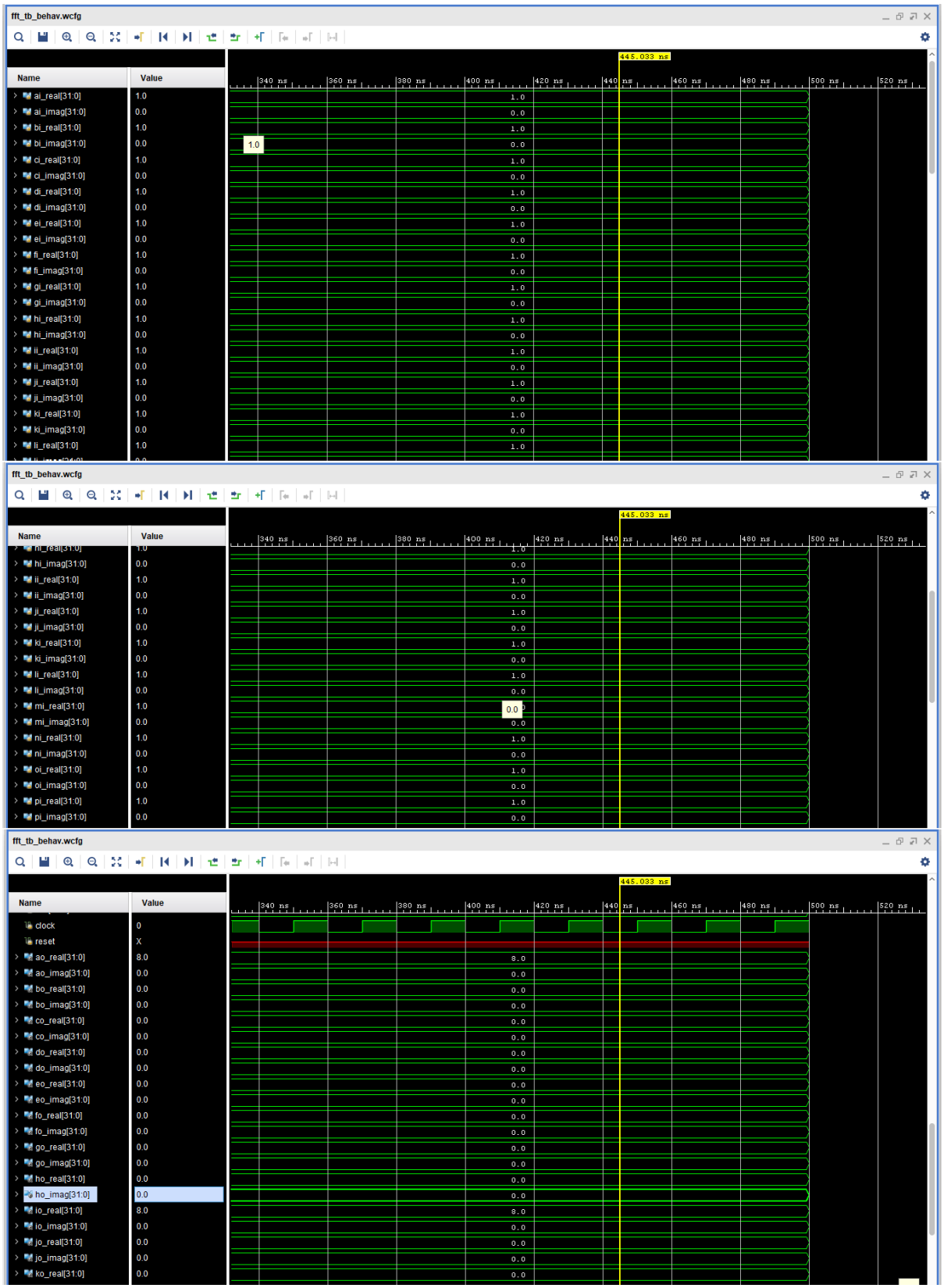
Stages: a1_rel,a1_imag , a2_rel,a2_imag, a3_rel,a3_imag

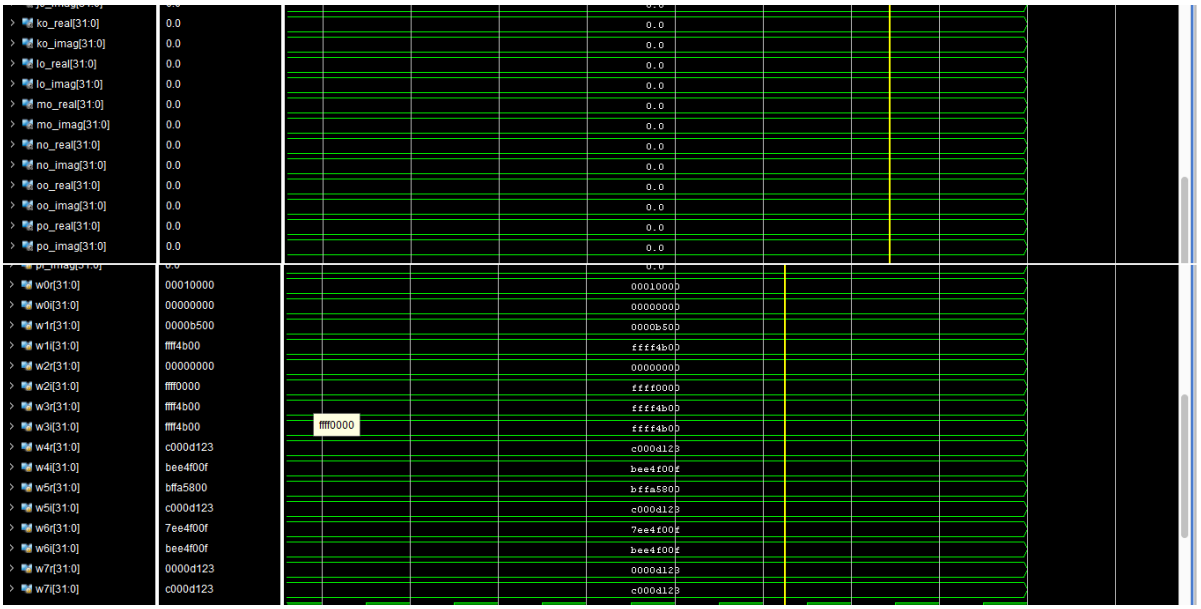
Outputs: ao_real,ao_imag to po-real,po-imag .

IV. RESULT:

Waveform Simulation(vivado)

The below figures show the digital waveform of the FFT inputs performing complex multiplication withtwiddle factor to obtain FFT output coefficient. As a sample input is given as { , , , , ….} with twiddle factors defined as {, , , --} , the complex number multiplication obtained is {, , ,…}. For these sample values of input signal after getting multiplied with twiddle factor , output coefficients obtained are {, , --}. These output coefficients can be observed in the simulation waveform after executing simulation in Vivado. The outputs are thus verified.


fft_tb_behav.wcfg
Name
Value
445.033 ns
340 ns
360 ns
380 ns
400 ns
420 ns
440 ns
460 ns
480 ns
500 ns
520 ns
ai_real[31:0] 1.0
ai_imag[31:0] 0.0
bi_real[31:0] 1.0
bi_imag[31:0] 0.0
ci_real[31:0] 1.0
ci_imag[31:0] 0.0
di_real[31:0] 1.0
di_imag[31:0] 0.0
ei_real[31:0] 1.0
ei_imag[31:0] 0.0
fi_real[31:0] 1.0
fi_imag[31:0] 0.0
gi_real[31:0] 1.0
gi_imag[31:0] 0.0
hi_real[31:0] 1.0
hi_imag[31:0] 0.0
ii_real[31:0] 1.0
ii_imag[31:0] 0.0
ji_real[31:0] 1.0
ji_imag[31:0] 0.0
ki_real[31:0] 1.0
ki_imag[31:0] 0.0
li_real[31:0] 1.0
li_imag[31:0] 0.0
mi_real[31:0] 1.0
mi_imag[31:0] 0.0
ni_real[31:0] 1.0
ni_imag[31:0] 0.0
oi_real[31:0] 1.0
oi_imag[31:0] 0.0
pi_real[31:0] 1.0
pi_imag[31:0] 0.0
clock 0
reset X
ao_real[31:0] 8.0
ao_imag[31:0] 0.0
bo_real[31:0] 0.0
bo_imag[31:0] 0.0
co_real[31:0] 0.0
co_imag[31:0] 0.0
do_real[31:0] 0.0
do_imag[31:0] 0.0
eo_real[31:0] 0.0
eo_imag[31:0] 0.0
fo_real[31:0] 0.0
fo_imag[31:0] 0.0
go_real[31:0] 0.0
go_imag[31:0] 0.0
ho_real[31:0] 0.0
ho_imag[31:0] 0.0
io_real[31:0] 8.0
io_imag[31:0] 0.0
jo_real[31:0] 0.0
jo_imag[31:0] 0.0
ko_real[31:0] 0.0



V. CONCLUSION AND FUTURE WORK:

This project successfully implemented complex number multiplier. This can be used to perform FFT 16 points using a Verilog code by computing the butterfly diagram using the code . With periodic waveform being given as input at the end of every period, this multiplier unit can be used to generate FFT coeffecients. This design is simulated using the xilinx vivado to analyse the waveform generation.

FFT Design:

This project is about to design FFT 16 points using Xilinx vivado .

Simulation:

This design was validated through the simulation I Xilinx vivado .

FUTURE WORKS:

- Implement the FFT on FPGA hardware.
- Create ASIC design flow for FFT 16 points.
- Generate the GDSII file.

VI. REFFERENCE:

1. **James W. Cooley and John W. Tukey**:
   - **"An Algorithm for the Machine Calculation of Complex Fourier Series"** (1965). This seminal paper introduced the Cooley-Tukey FFT algorithm.

2. **William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery**:
   - **"Numerical Recipes: The Art of Scientific Computing"** (various editions). This book includes comprehensive sections on FFT algorithms and their applications.

3. **Richard G. Lyons**:
   - **"Understanding Digital Signal Processing"** (3rd Edition, 2010). This book provides an accessible introduction to DSP concepts, including FFT.

4. **E. Oran Brigham**:
   - **"The Fast Fourier Transform and Its Applications"** (1988). A classic textbook offering detailed explanations of FFT theory and practical uses.

VII. APPENDIX

FFT Main block:

```
module
FFT_main(clock,reset,ai_real,ai_imag,bi_real,bi_imag,ci_real,ci_imag,di_real,di_imag,ei_real
,ei_imag,fi_real,fi_imag,gi_real,gi_imag,hi_real,hi_imag,ii_real,ii_imag,ji_real,ji_imag,ki_real
,ki_imag,li_real,li_imag,mi_real,mi_imag,ni_real,ni_imag,oi_real,oi_imag,pi_real,pi_imag,ao
_real,ao_imag,bo_real,bo_imag,co_real,co_imag,do_real,do_imag,eo_real,eo_imag,fo_real,
fo_imag,go_real,go_imag,ho_real,ho_imag,io_real,io_imag,jo_real,jo_imag,ko_real,ko_imag
,lo_real,lo_imag,mo_real,mo_imag,no_real,no_imag,oo_real,oo_imag,po_real,po_imag,w0r,
w0i,w1r,w1i,w2r,w2i,w3r,w3i,w4r,w4i,w5r,w5i,w6r,w6i,w7r,w7i);


input clock;

input reset;

input signed [31:0] ai_real;

input signed [31:0] ai_imag;

input signed [31:0] bi_real;

input signed [31:0] bi_imag;

input signed [31:0] ci_real;

input signed [31:0] ci_imag;

input signed [31:0] di_real;

input signed [31:0] di_imag;

input signed [31:0] ei_real;

input signed [31:0] ei_imag;

input signed [31:0] fi_real;

input signed [31:0] fi_imag;

input signed [31:0] gi_real;

input signed [31:0] gi_imag;

input signed [31:0] hi_real;

input signed [31:0] hi_imag;

input signed [31:0] ii_real;

input signed [31:0] ii_imag;
```

```
input signed [31:0] ji_real;
input signed [31:0] ji_imag;
input signed [31:0] ki_real;
input signed [31:0] ki_imag;
input signed [31:0] li_real;
input signed [31:0] li_imag;
input signed [31:0] mi_real;
input signed [31:0] mi_imag;
input signed [31:0] ni_real;
input signed [31:0] ni_imag;
input signed [31:0] oi_real;
input signed [31:0] oi_imag;
input signed [31:0] pi_real;
input signed [31:0] pi_imag;
output signed [31:0] ao_real;
output signed [31:0] ao_imag;
output signed [31:0] bo_real;
output signed [31:0] bo_imag;
output signed [31:0] co_real;
output signed [31:0] co_imag;
output signed [31:0] do_real;
output signed [31:0] do_imag;
output signed [31:0] eo_real;
output signed [31:0] eo_imag;
output signed [31:0] fo_real;
output signed [31:0] fo_imag;
output signed [31:0] go_real;
output signed [31:0] go_imag;
output signed [31:0] ho_real;
```

```
output signed [31:0] ho_imag;
output signed [31:0] io_real;
output signed [31:0] io_imag;
output signed [31:0] jo_real;
output signed [31:0] jo_imag;
output signed [31:0] ko_real;
output signed [31:0] ko_imag;
output signed [31:0] lo_real;
output signed [31:0] lo_imag;
output signed [31:0] mo_real;
output signed [31:0] mo_imag;
output signed [31:0] no_real;
output signed [31:0] no_imag;
output signed [31:0] oo_real;
output signed [31:0] oo_imag;
output signed [31:0] po_real;
output signed [31:0] po_imag;
wire signed [31:0] a1_real,b1_real,c1_real,d1_real,e1_real,f1_real,g1_real,h1_real,i1_real,j1_real,k1_real,l1_real,m1_real,n1_real,o1_real,p1_real;
wire signed [31:0] a1_imag,b1_imag,c1_imag,d1_imag,e1_imag,f1_imag,g1_imag,h1_imag,i1_imag,j1_imag,k1_imag,l1_imag,m1_imag,n1_imag,o1_imag,p1_imag;
wire signed [31:0] a2_real,b2_real,c2_real,d2_real,e2_real,f2_real,g2_real,h2_real,i2_real,j2_real,k2_real,l2_real,m2_real,n2_real,o2_real,p2_real;
wire signed [31:0] a2_imag,b2_imag,c2_imag,d2_imag,e2_imag,f2_imag,g2_imag,h2_imag,i2_imag,j2_imag,k2_imag,l2_imag,m2_imag,n2_imag,o2_imag,p2_imag;
wire signed [31:0] a3_real,b3_real,c3_real,d3_real,e3_real,f3_real,g3_real,h3_real,i3_real,j3_real,k3_real,l3_real,m3_real,n3_real,o3_real,p3_real;
```

```
wire signed [31:0] a3_imag,b3_imag,c3_imag,d3_imag,e3_imag,f3_imag,g3_imag,h3_imag,i3_imag,j3_imag,k3_imag,l3_imag,m3_imag,n3_imag,o3_imag,p3_imag;

input signed [31:0] w0r,w0i,w1r,w1i,w2r,w2i,w3r,w3i,w4r,w4i,w5r,w5i,w6r,w6i,w7r,w7i;

a_plus_bc dut01(clock,reset,ai_real,ai_imag,ii_real,ii_imag,w0r,w0i,a1_real,a1_imag);
a_minus_bc dut02(clock,reset,ai_real,ai_imag,ii_real,ii_imag,w0r,w0i,b1_real,b1_imag);

a_plus_bc dut03(clock,reset,ei_real,ei_imag,mi_real,mi_imag,w0r,w0i,c1_real,c1_imag);
a_minus_bc dut04(clock,reset,ei_real,ei_imag,mi_real,mi_imag,w0r,w0i,d1_real,d1_imag);
a_plus_bc dut05(clock,reset,ci_real,ci_imag,ki_real,ki_imag,w0r,w0i,e1_real,e1_imag);
a_minus_bc dut06(clock,reset,ci_real,ci_imag,ki_real,ki_imag,w0r,w0i,f1_real,f1_imag);

a_plus_bc dut07(clock,reset,gi_real,gi_imag,oi_real,oi_imag,w0r,w0i,g1_real,g1_imag);
a_minus_bc dut08(clock,reset,gi_real,gi_imag,oi_real,oi_imag,w0r,w0i,h1_real,h1_imag);
a_plus_bc dut09(clock,reset,bi_real,bi_imag,ji_real,ji_imag,w0r,w0i,i1_real,i1_imag);
a_minus_bc dut10(clock,reset,bi_real,bi_imag,ji_real,ji_imag,w0r,w0i,j1_real,j1_imag);
a_plus_bc dut11(clock,reset,fi_real,fi_imag,ni_real,ni_imag,w0r,w0i,k1_real,k1_imag);
a_minus_bc dut12(clock,reset,fi_real,fi_imag,ni_real,ni_imag,w0r,w0i,l1_real,l1_imag);
a_plus_bc dut13(clock,reset,di_real,di_imag,li_real,li_imag,w0r,w0i,m1_real,m1_imag);
a_minus_bc dut14(clock,reset,di_real,di_imag,li_real,li_imag,w0r,w0i,n1_real,n1_imag);
a_plus_bc dut15(clock,reset,ni_real,ni_imag,pi_real,pi_imag,w0r,w0i,o1_real,o1_imag);
a_minus_bc dut16(clock,reset,ni_real,ni_imag,pi_real,pi_imag,w0r,w0i,p1_real,p1_imag);
a_plus_bc dut17(clock,reset,a1_real,a1_imag,c1_real,c1_imag,w0r,w0i,a2_real,a2_imag);
a_plus_bc dut18(clock,reset,b1_real,b1_imag,d1_real,d1_imag,w4r,w2i,b2_real,b2_imag);
a_minus_bc dut19(clock,reset,a1_real,a1_imag,c1_real,c1_imag,w0r,w0i,c2_real,c2_imag);
a_minus_bc dut20(clock,reset,b1_real,b1_imag,d1_real,d1_imag,w4r,w2i,d2_real,d2_imag);
a_plus_bc dut21(clock,reset,e1_real,e1_imag,g1_real,g1_imag,w0r,w0i,e2_real,e2_imag);
a_plus_bc dut22(clock,reset,f1_real,f1_imag,h1_real,h1_imag,w4r,w2i,f2_real,f2_imag);
```

```
a_minus_bc dut23(clock,reset,e1_real,e1_imag,g1_real,g1_imag,w0r,w0i,g2_real,g2_imag);
a_minus_bc dut24(clock,reset,f1_real,f1_imag,h1_real,h1_imag,w4r,w2i,h2_real,h2_imag);
a_plus_bc dut25(clock,reset,i1_real,i1_imag,k1_real,k1_imag,w0r,w0i,i2_real,i2_imag);
a_plus_bc  dut26(clock,reset,j1_real,j1_imag,l1_real,l1_imag,w4r,w2i,j2_real,j2_imag);
a_minus_bc dut27(clock,reset,i1_real,i1_imag,k1_real,k1_imag,w0r,w0i,k2_real,k2_imag);
a_minus_bc dut28(clock,reset,j1_real,j1_imag,l1_real,l1_imag,w4r,w2i,l2_real,l2_imag);
a_plus_bc
dut29(clock,reset,m1_real,m1_imag,o1_real,o1_imag,w0r,w0i,m2_real,m2_imag);
a_plus_bc dut30(clock,reset,n1_real,n1_imag,p1_real,p1_imag,w4r,w2i,n2_real,n2_imag);
a_minus_bc
dut31(clock,reset,m1_real,m1_imag,o1_real,o1_imag,w0r,w0i,o2_real,o2_imag);
a_minus_bc dut32(clock,reset,n1_real,n1_imag,p1_real,p1_imag,w4r,w2i,p2_real,p2_imag);
a_plus_bc dut33(clock,reset,a2_real,a2_imag,e2_real,e2_imag,w0r,w0i,a3_real,a3_imag);
a_plus_bc dut34(clock,reset,b2_real,b2_imag,f2_real,f2_imag,w2r,w2i,b3_real,b3_imag);
a_plus_bc dut35(clock,reset,c2_real,c2_imag,g2_real,g2_imag,w4r,w4i,c3_real,c3_imag);
a_plus_bc dut36(clock,reset,d2_real,d2_imag,h2_real,h2_imag,w6r,w6i,d3_real,d3_imag);
a_minus_bc dut37(clock,reset,a2_real,a2_imag,e2_real,e2_imag,w0r,w0i,e3_real,e3_imag);
a_minus_bc dut38(clock,reset,b2_real,b2_imag,f2_real,f2_imag,w2r,w2i,f3_real,f3_imag);
a_minus_bc dut39(clock,reset,c2_real,c2_imag,g2_real,g2_imag,w4r,w4i,g3_real,g3_imag);
a_minus_bc dut40(clock,reset,d2_real,d2_imag,h2_real,h2_imag,w6r,w6i,h3_real,h3_imag);
a_plus_bc dut41(clock,reset,i2_real,i2_imag,m2_real,m2_imag,w0r,w0i,i3_real,i3_imag);
a_plus_bc dut42(clock,reset,j2_real,j2_imag,n2_real,n2_imag,w2r,w2i,j3_real,j3_imag);
a_plus_bc dut43(clock,reset,k2_real,k2_imag,o2_real,o2_imag,w4r,w4i,k3_real,k3_imag);
a_plus_bc  dut44(clock,reset,l2_real,l2_imag,p2_real,p2_imag,w6r,w6i,l3_real,l3_imag);
a_minus_bc
dut45(clock,reset,i2_real,i2_imag,m2_real,m2_imag,w0r,w0i,m3_real,m3_imag);
a_minus_bc dut46(clock,reset,j2_real,j2_imag,n2_real,n2_imag,w2r,w2i,n3_real,n3_imag);
a_minus_bc dut47(clock,reset,k2_real,k2_imag,o2_real,o2_imag,w4r,w4i,o3_real,o3_imag);
a_minus_bc dut48(clock,reset,l2_real,l2_imag,p2_real,p2_imag,w6r,w6i,p3_real,p3_imag);
a_plus_bc  dut49(clock,reset,a2_real,a2_imag,i2_real,i2_imag,w0r,w0i,ao_real,ao_imag);
```

```
a_plus_bc   dut50(clock,reset,b2_real,b2_imag,j2_real,j2_imag,w2r,w2i,bo_real,bo_imag);

a_plus_bc   dut51(clock,reset,c2_real,c2_imag,k2_real,k2_imag,w4r,w4i,co_real,co_imag);

a_plus_bc dut52(clock,reset,d2_real,d2_imag,l2_real,l2_imag,w6r,w6i,do_real,do_imag);

a_minus_bc
dut53(clock,reset,e2_real,e2_imag,m2_real,m2_imag,w0r,w0i,eo_real,eo_imag);

a_minus_bc dut54(clock,reset,f2_real,f2_imag,n2_real,n2_imag,w2r,w2i,fo_real,fo_imag);

a_minus_bc dut55(clock,reset,g2_real,g2_imag,o2_real,o2_imag,w4r,w4i,go_real,go_imag);

a_minus_bc dut56(clock,reset,h2_real,h2_imag,p2_real,p2_imag,w6r,w6i,ho_real,ho_imag);

a_plus_bc dut57(clock,reset,a2_real,a2_imag,i2_real,i2_imag,w0r,w0i,io_real,io_imag);

a_plus_bc dut58(clock,reset,b2_real,b2_imag,j2_real,j2_imag,w2r,w2i,jo_real,jo_imag);

a_plus_bc dut59(clock,reset,c2_real,c2_imag,k2_real,k2_imag,w4r,w4i,ko_real,ko_imag);

a_plus_bc  dut60(clock,reset,d2_real,d2_imag,l2_real,l2_imag,w6r,w6i,lo_real,lo_imag);

a_minus_bc
dut61(clock,reset,e2_real,e2_imag,m2_real,m2_imag,w0r,w0i,mo_real,mo_imag);

a_minus_bc dut62(clock,reset,f2_real,f2_imag,n2_real,n2_imag,w2r,w2i,no_real,no_imag);

a_minus_bc dut63(clock,reset,g2_real,g2_imag,o2_real,o2_imag,w4r,w4i,oo_real,oo_imag);

a_minus_bc dut64(clock,reset,h2_real,h2_imag,p2_real,p2_imag,w6r,w6i,po_real,po_imag);



endmodule
```

FFT test bench

```
module FFT_tb();

reg      signed  [31:0]
ai_real,ai_imag,bi_real,bi_imag,ci_real,ci_imag,di_real,di_imag,ei_real,ei_imag,fi_real,fi_im
ag,gi_real,gi_imag,hi_real,hi_imag,ii_real,ii_imag,ji_real,ji_imag,ki_real,ki_imag,li_real,li_im
ag,mi_real,mi_imag,ni_real,ni_imag,oi_real,oi_imag,pi_real,pi_imag,w0r,w0i,w1r,w1i,w2r,w
2i,w3r,w3i,w4r,w4i,w5r,w5i,w6r,w6i,w7r,w7i;
```

```
reg     clock,reset;

wire signed [31:0]

ao_real,ao_imag,bo_real,bo_imag,co_real,co_imag,do_real,do_imag,eo_real,eo_imag,fo_real,fo_imag,go_real,go_imag,ho_real,ho_imag,io_real,io_imag,jo_real,jo_imag,ko_real,ko_imag,lo_real,lo_imag,mo_real,mo_imag,no_real,no_imag,oo_real,oo_imag,po_real,po_imag;

FFT_main

dut(clock,reset,ai_real,ai_imag,bi_real,bi_imag,ci_real,ci_imag,di_real,di_imag,ei_real,ei_imag,fi_real,fi_imag,gi_real,gi_imag,hi_real,hi_imag,ii_real,ii_imag,ji_real,ji_imag,ki_real,ki_imag,li_real,li_imag,mi_real,mi_imag,ni_real,ni_imag,oi_real,oi_imag,pi_real,pi_imag,ao_real,ao_imag,bo_real,bo_imag,co_real,co_imag,do_real,do_imag,eo_real,eo_imag,fo_real,fo_imag,go_real,go_imag,ho_real,ho_imag,io_real,io_imag,jo_real,jo_imag,ko_real,ko_imag,lo_real,lo_imag,mo_real,mo_imag,no_real,no_imag,oo_real,oo_imag,po_real,po_imag,w0r,w0i,w1r,w1i,w2r,w2i,w3r,w3i,w4r,w4i,w5r,w5i,w6r,w6i,w7r,w7i);

initial begin

clock=1'b0;

forever

#10 clock=~clock;

end


task delay;

begin

#10;

end

endtask


task rst;

begin

@(negedge clock)

reset=1'b1;

@(negedge clock)

reset=1'b0;

end
```

```
endtask


task initialize;
begin
reset=1'b1;
ai_real=32'b0;
ai_imag=32'b0;
bi_real=32'b0;
bi_imag=32'b0;
ci_real=32'b0;
ci_imag=32'b0;
di_real=32'b0;
di_imag=32'b0;
ei_real=32'b0;
ei_imag=32'b0;
fi_real=32'b0;
fi_imag=32'b0;
gi_real=32'b0;
gi_imag=32'b0;
hi_real=32'b0;
hi_imag=32'b0;
ii_real=32'b0;
ii_imag=32'b0;
ji_real=32'b0;
ji_imag=32'b0;
ki_real=32'b0;
ki_imag=32'b0;
li_real=32'b0;
li_imag=32'b0;
```

```
mi_real=32'b0;
mi_imag=32'b0;
ni_real=32'b0;
ni_imag=32'b0;
oi_real=32'b0;
oi_imag=32'b0;
pi_real=32'b0;
pi_imag=32'b0;
end
endtask

task inputs;
begin

ai_real=32'b00000000000000010000000000000000;
ai_imag=32'b0;
bi_real=32'b00000000000000010000000000000000;
bi_imag=32'b0;
ci_real=32'b00000000000000010000000000000000;
ci_imag=32'b0;
di_real=32'b00000000000000010000000000000000;
di_imag=32'b0;
ei_real=32'b00000000000000010000000000000000;
ei_imag=32'b0;
fi_real=32'b00000000000000010000000000000000;
fi_imag=32'b0;
gi_real=32'b00000000000000010000000000000000;
gi_imag=32'b0;
hi_real=32'b00000000000000010001111110011010;
```

```
hi_imag=32'b0;
ii_real=32'b00000000000000001000101110010000;
ii_imag=32'b0;
ji_real=32'b00000000000000001000000000000000;
ji_imag=32'b0;
ki_real=32'b00000000000000001000000000000000;
ki_imag=32'b0;
li_real=32'b00000000000000001000000000000000;
li_imag=32'b0;
mi_real=32'b00000000000000001000000000000000;
mi_imag=32'b0;
ni_real=32'b00000000000000001000000000000000;
ni_imag=32'b0;
oi_real=32'b00000000000000001000000000000000;
oi_imag=32'b0;
pi_real=32'b00000000000000001000000000000000;
pi_imag=32'b0;
   w0r=32'b00000000000000001000000000000000;
   w0i=32'b00000000000000000000000000000000;
   w1r=32'b00000000000000001011010100000000;
   w1i=32'b11111111111111110100101100000000;
   w2r=32'b00000000000000000000000000000000;
   w2i=32'b11111111111111110000000000000000;
   w3r=32'b11111111111111110100101100000000;
   w3i=32'b11111111111111110100101100000000;
   w4r=32'b11000000000000001101000100100011;
   w4i=32'b10111110111001001111000000001111;
   w5r=32'b10111111111110100101100000000000;
   w5i=32'b11000000000000001101000100100011;
```

```
    w6r=32'b01111110111001001111000000001111;
    w6i=32'b10111110111001001111000000001111;
    w7r=32'b00000000000000001101000100100011;
    w7i=32'b11000000000000001101000100100011;
end
endtask

initial
begin
//rst;
//delay;
//initialize;
//delay;
inputs;
//#100;
end
initial #500 $finish;
initial
$monitor("ai_real=%b \nai_imag=%b \nbi_real=%b \nbi_imag=%b \nci_real=%b
\nci_imag=%b \ndi_real=%b \ndi_imag=%b \nei_real=%b \nei_imag=%b \nfi_real=%b
\nfi_imag=%b \ngi_real=%b \ngi_imag=%b \nhi_real=%b \nhi_imag=%b \nii_real=%b
\nii_imag=%b \nji_real;=%b \nji_imag=%b \nki_real=%b \nki_imag=%b \nli_real=%b
\nli_imag=%b \nmi_real=%b \nmi_imag=%b \nni_real=%b \nni_imag=%b \noi_real=%b
\noi_imag=%b \npi_real=%b \npi_imag=%b \nao_real=%b \nao_imag=%b \nbo_real=%b
\nbo_imag=%b \nco_real=%b \nco_imag=%b \ndo_real=%b \ndo_imag=%b \neo_real=%b
\neo_imag=%b \nfo_real=%b \nfo_imag=%b \ngo_real=%b \ngo_imag=%b \nho_real=%b
\nho_imag=%b \nio_real=%b \nio_imag=%b \njo_real;=%b \njo_imag=%b \nko_real=%b
\nko_imag=%b \nlo_real=%b \nlo_imag=%b \nmo_real=%b \nmo_imag=%b \nno_real=%b
\nno_imag=%b \noo_real=%b \noo_imag=%b \npo_real=%b \npo_imag=%b
\n",ai_real,ai_imag,bi_real,bi_imag,ci_real,ci_imag,di_real,di_imag,ei_real,ei_imag,fi_real,fi
_imag,gi_real,gi_imag,hi_real,hi_imag,ii_real,ii_imag,ji_real,ji_imag,ki_real,ki_imag,li_real,li
_imag,mi_real,mi_imag,ni_real,ni_imag,oi_real,oi_imag,pi_real,pi_imag,ao_real,ao_imag,bo
_real,bo_imag,co_real,co_imag,do_real,do_imag,eo_real,eo_imag,fo_real,fo_imag,go_real,g
o_imag,ho_real,ho_imag,io_real,io_imag,jo_real,jo_imag,ko_real,ko_imag,lo_real,lo_imag,
mo_real,mo_imag,no_real,no_imag,oo_real,oo_imag,po_real,pi_imag);
```

```
endmodule
```

Complex_multiplication program

```
module
complex_multiplication(clock,reset,real1,imag1,real2,imag2,real_result,imag_result);
input signed [31:0] real1; // Real part of the first complex number
input signed [31:0] imag1; // Imaginary part of the first complex number
input signed [31:0] real2; // Real part of the second complex number
input signed [31:0] imag2; // Imaginary part of the second complex number
output reg signed [63:0] real_result; // Real part of the result (16 bits for higher precision)
output reg signed [63:0] imag_result; // Imaginary part of the result (16 bits for higher precision)
input clock;
input reset;
always @(posedge clock)
begin
if(reset)
begin
real_result = 32'b0;
imag_result = 32'b0;
end
else
begin
// Perform complex multiplication (a + bi) * (c + di) = (ac - bd) + (ad + bc)i
real_result = (real1 * real2) - (imag1 * imag2); // (ac - bd)
imag_result = (real1 * imag2) + (imag1 * real2); // (ad + bc)
end endmodule
```

Complex_subtraction program

```
module complex_subtraction(clock,reset,real1,imag1,real2,imag2,real_result,imag_result);
input signed [31:0] real1; // Real part of the first complex number
input signed [31:0] imag1; // Imaginary part of the first complex number
```

```
input signed [31:0] real2; // Real part of the second complex number
input signed [31:0] imag2; // Imaginary part of the second complex number
output reg signed [31:0] real_result; // Real part of the result (16 bits for higher precision)
output reg signed [31:0] imag_result; // Imaginary part of the result (16 bits for higher precision)
input clock;
input reset;

always @(posedge clock)
begin
if(reset)
begin
real_result = 32'b0;
imag_result = 32'b0;
end
else
begin
real_result = real1 - real2; // Subtract the real parts
imag_result = imag1 - imag2; // Subtract the imaginary parts
end endmodule
```

Complex_addition program

```
module complex_addition(clock,reset,real1,imag1,real2,imag2,real_result,imag_result);
input signed [31:0] real1; // Real part of the first complex number
input signed [31:0] imag1; // Imaginary part of the first complex number
input signed [31:0] real2; // Real part of the second complex number
input signed [31:0] imag2; // Imaginary part of the second complex number
output reg signed [31:0] real_result; // Real part of the result
output reg signed [31:0] imag_result; // Imaginary part of the result
input clock;
input reset;
```

```
always @(posedge clock)
begin
if(reset)
begin
real_result = 32'b0;
imag_result = 32'b0;
end
else
begin
real_result = real1 + real2; // Add the real parts
imag_result = imag1 + imag2; // Add the imaginary parts
end
end
endmodule
```

a_minus program

```
module
a_minus_bc(clock,reset,a_real,a_imag,b_real,b_imag,c_real,c_imag,result_real,result_imag)
;
input signed [31:0] a_real; // Real part of complex number a
input signed [31:0] a_imag; // Imaginary part of complex number a
input signed [31:0] b_real; // Real part of complex number b
input signed [31:0] b_imag; // Imaginary part of complex number b
input signed [31:0] c_real; // Real part of complex number c
input signed [31:0] c_imag; // Imaginary part of complex number c
output signed [31:0] result_real; // Real part of the final result
output signed [31:0] result_imag; // Imaginary part of the final result
input clock;
input reset;
```

```
wire signed [63:0] bc_real; // Intermediate result for real part of b*c

wire signed [63:0] bc_imag; // Intermediate result for imaginary part of b*c


// Instantiate complex_multiplier module for multiplication of b and c

complex_multiplication
multiply_b_c(clock,reset,b_real,b_imag,c_real,c_imag,bc_real,bc_imag);


// Instantiate complex_subtractor module for subtraction of a and (b*c)

complex_subtraction
subtract_a_bc(clock,reset,a_real,a_imag,bc_real[47:16],bc_imag[47:16],result_real,result_i
mag);

endmodule
```

a_plus program

```
module
a_plus_bc(clock,reset,a_real,a_imag,b_real,b_imag,c_real,c_imag,result_real,result_imag);

input signed [31:0] a_real; // Real part of complex number a

input signed [31:0] a_imag; // Imaginary part of complex number a

input signed [31:0] b_real; // Real part of complex number b

input signed [31:0] b_imag; // Imaginary part of complex number b

input signed [31:0] c_real; // Real part of complex number c

input signed [31:0] c_imag; // Imaginary part of complex number c

output signed [31:0] result_real; // Real part of the final result

output signed [31:0] result_imag;// Imaginary part of the final result

wire signed [63:0] bc_real; // Intermediate result for real part of b*c

wire signed [63:0] bc_imag; // Intermediate result for imaginary part of b*c

input clock;

input reset;

// Instantiate complex_multiplier module for multiplication of b and c
```

```
complex_multiplication
multiply_b_c(clock,reset,b_real,b_imag,c_real,c_imag,bc_real,bc_imag);

// Instantiate complex_adder module for addition of a and (b*c)

complex_addition
add_a_bc(clock,reset,a_real,a_imag,bc_real[47:16],bc_imag[47:16],result_real,result_imag);

endmodule
```