SCALABLE DEVS MODELING OF QUANTIZED DSP BASED SYSTEMS

By

Harshavardhan Gopalakrishnan

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

2 0 0 5

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript is whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: _____

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

_____ _____

Bernard P Zeigler — Date

Professor of Electrical and Computer Engineering

ACKNOWLEDGMENTS

I express my sincere thanks to Dr. Bernard P. Zeigler for being my advisor and for his immense help and guidance. My sincere thanks to Dr. Doohwan Kim and Dr. James J. Nutaro for helping me in the course of my thesis.

I thank my thesis committee member Dr. Olgierd A. Palusinski for his time, effort and patience. I thank my family for their care, encouragement and support and also my friends who help me in all the ways they can.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

ABSTRACT

Discrete event modeling and simulation of complex Digital Signal Processing (DSP) systems provide a systematic approach for better engineering, be it system level or software level of design. Digital audio processing has become an indispensable part of everyday life and is found in most of the devices that we use daily. The design of such complex systems needs formal models and corresponding methods of design. Discrete event modeling and simulation provides a framework to analyze a digital audio system. In this thesis, a typical audio system is modeled and simulated using Discrete Events and effects of quantization, in the computation of the Fast Fourier Transform (FFT) are studied. The models are shown to be scalable, re-usable, performance-oriented and suited for real-time deployment. This thesis provides a powerful and configurable framework for complex DSP based systems.

Models for FFT (Butterfly, Stages, Controller and Quantizer) have been built for Radix–2 and Radix–4 transforms for Decimation in Frequency and Decimation Time Transforms. Audio Player, Spectrum Analyzer, Audio Generator (with Au format Decoder/Encoder) are the other models that have been built for the system. Experiments were performed to analyze the speed-up due to pipelining the stages of the FFT in Radix – 2 and Radix – 4 transforms and compared for efficiency. Quantization when applied during the computation of FFT reduces the time taken for the computation. When forward and inverse transform are applied to a music input, it is observed that time taken for the computation reduces to 82% of the actual time taken when noise starts getting audible.

1 INTRODUCTION

1.1 DIGITAL AUDIO SYSTEMS

Digital Signal Processing has rapidly advanced due to the significant advances in digital computer technology and integrated circuit fabrication. Digital audio processing has become an indispensable part of everyday life and is found in most of the devices that we use daily. With the advancement in digital media technology, the market has been experiencing an unparalleled inclination towards fully digital audio systems.

The rapid advancement in the digital infrastructure, along with new digital sound sources, has created an unprecedented opportunity to process digital audio data and the need to do so. Digital systems allow the use of digital signal processing, which has the power and flexibility to address, in digital, problems that were too expensive or too difficult to address well in analog. In addition to the increase in digital media and infrastructure, the market is seeing an unprecedented trend toward minimization and sleeker-than-ever audio equipment styling.

A block diagram of a typical digital audio processing system is shown below:

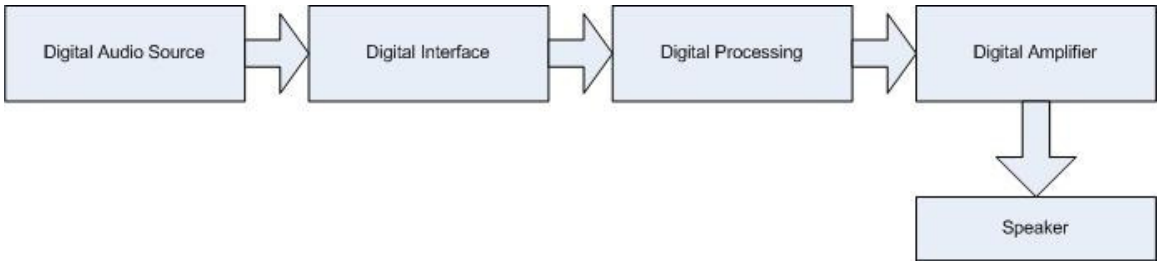


Figure 1.1 : Block diagram of a typical Digital Audio Processing system

Here, digital data from a digital audio source, which can be from CD/SPDIF/USB or any digital input, is routed to the Digital Processing Unit by a compatible interface. The interface can do the pre-processing jobs. The digital processing unit is where digital filtering, equalization and so forth are performed. Once this processing is completed, the signal is sent on to a digital amplifier, which converts the digital signal to a digital format that can supply adequate current and voltage to drive the speaker.

1.2 OBJECT-ORIENTED DISCRETE EVENT MODELING AND SIMULATION

The Discrete Event System Specification (DEVS) formalism introduced by Zeigler (1976) provides a means of specifying a system with its discrete inputs, states, and outputs, and functions for determining next states and outputs given current states and inputs (Zeigler, 1984b). Object-oriented programming is a paradigm in which a software system is decomposed into subsystems based on objects. The paradigm enhances software maintainability, extensibility and reusability. Computation is done by objects exchanging messages among themselves. Object oriented program encourages a much

more decentralized style of decision making by creating objects whose existence may continue throughout the life of the program. We can make such objects act as experts in their own task assignments by providing them with the appropriate knowledge [3]. Object-oriented modeling and simulation provides a structured, computer-aided, streamlined approach of modeling hierarchical systems. The OO methodology emphasizes code reuse or inheritance along with the other features like encapsulation and abstraction. The key object of simulation is the mapping between real world and digital world as objects that can be represented with their own attributes and which can act according to external inputs or internal transitions.

Looking at the problem's perspective, system design is a complicated process with the systems getting complex with the advances in technology and time. The process is simplified by use of modeling framework and simulation. Discrete event modeling and simulation of these complex systems provide a systematic approach for better engineering, be it system level or software level of design. Design of such complex systems needs formal models and corresponding methods of design. Discrete event modeling and simulation would provide a concrete framework to analyze a digital audio system using discrete events. When a digital audio system is broken down into its blocks (namely Forward Transform, Inverse Transform, Linear Filters, Audio capture, Audio Player etc…), modeling them on the basis of events would be advantageous looking at it as hardware or a software system. The models are aimed to be scalable, re-usable, performance-oriented and suited for real-time deployment. This thesis provides a

powerful and configurable framework for complex DSP systems. All these requirements fit into object-oriented discrete event modeling and simulation.

1.3 ADEVS

ADEVS (A Discrete EVent System simulator) is a C++ library for constructing discrete event simulations based on the Parallel DEVS and Dynamic Structure DEVS formalisms [4]. With its conformance to the DEVS formalisms, it is a highly configurable simulation environment with all the versatility provided by the C++ language in which it is implemented. Due to the fact that the simulations of DSP systems based on discrete events require high-performance simulation environment for real-time simulations, ADEVS provides a suitable framework for modeling and simulation of the DSP systems. The user manual and documentation for ADEVS available at [http://www.ece.arizona.edu/~nutaro/adevs-docs/index.html] details the package configuration, usage and simulation details.

1.4 AIM OF THESIS

The aim of this thesis is to model and simulate a digital audio system as shown in Figure 1.1 using DEVS and study the effects of quantization when introduced during the computation of FFT.

The advantages of using Discrete Event Simulation are:

1. When a process is running with multiple tasks or threads, the synchronization has to be done on the basis of events. This is because the tasks cannot be critically timed and polling the state of the tasks using a time driven controller is not efficient. For example, here we have pipelined stages for the FFT and the synchronization between the stages is done on the basis of events using a controller.
2. Quantization used in the computation of FFT makes the pipelined stages asynchronous. Synchronization can be done on the basis of events monitored by a controller. This is an efficient method of controlling the data flow between the stages, each of which might take a different time for processing based on the inputs.
3. The inputs to the system need not be discrete-time which gives the flexibility to use queued processing in pre-emptive processing environments.

This thesis includes formalization of discrete event models that form building blocks of a digital audio system. This approach gives a distinct advantage over other simulation methodologies in the way that the system can be hierarchically modeled and provides a solid implementation at the system level or software level. Also the other advantages have been mentioned above and in section 1.2. Most of the simulation environments available are arduously slow and are not suitable for real-time simulations. These models expose a way of overcoming the disadvantage and opens an avenue for discrete event based modeling and simulation of digital signal processing systems. The various blocks can be individually studied and optimized for performance based on time/space,

scalability and reusability. Here a typical audio system with Fast Fourier Transform (FFT) for forward and inverse transform (Decimation in Time as well as Decimation in Frequency: Radix – 2 and Radix - 4), spectrum analyzer, filter, player are modeled and simulated using Discrete Events. Effects of computational quantization, when introduced during computation of the FFT are studied. The system with pipelined FFT stages provides a way to utilize the most out of multi-processor systems, implemented in either software or hardware. The models can also be looked at as formalization in the hardware design with independent blocks that can be designed as hardware blocks. The models are aimed to be scalable, re-usable, performance-oriented and suited for real-time deployment. This thesis provides a powerful and configurable framework for complex DSP systems.

The following shows the block diagram of the system that has been simulated for the implementation and experiments:

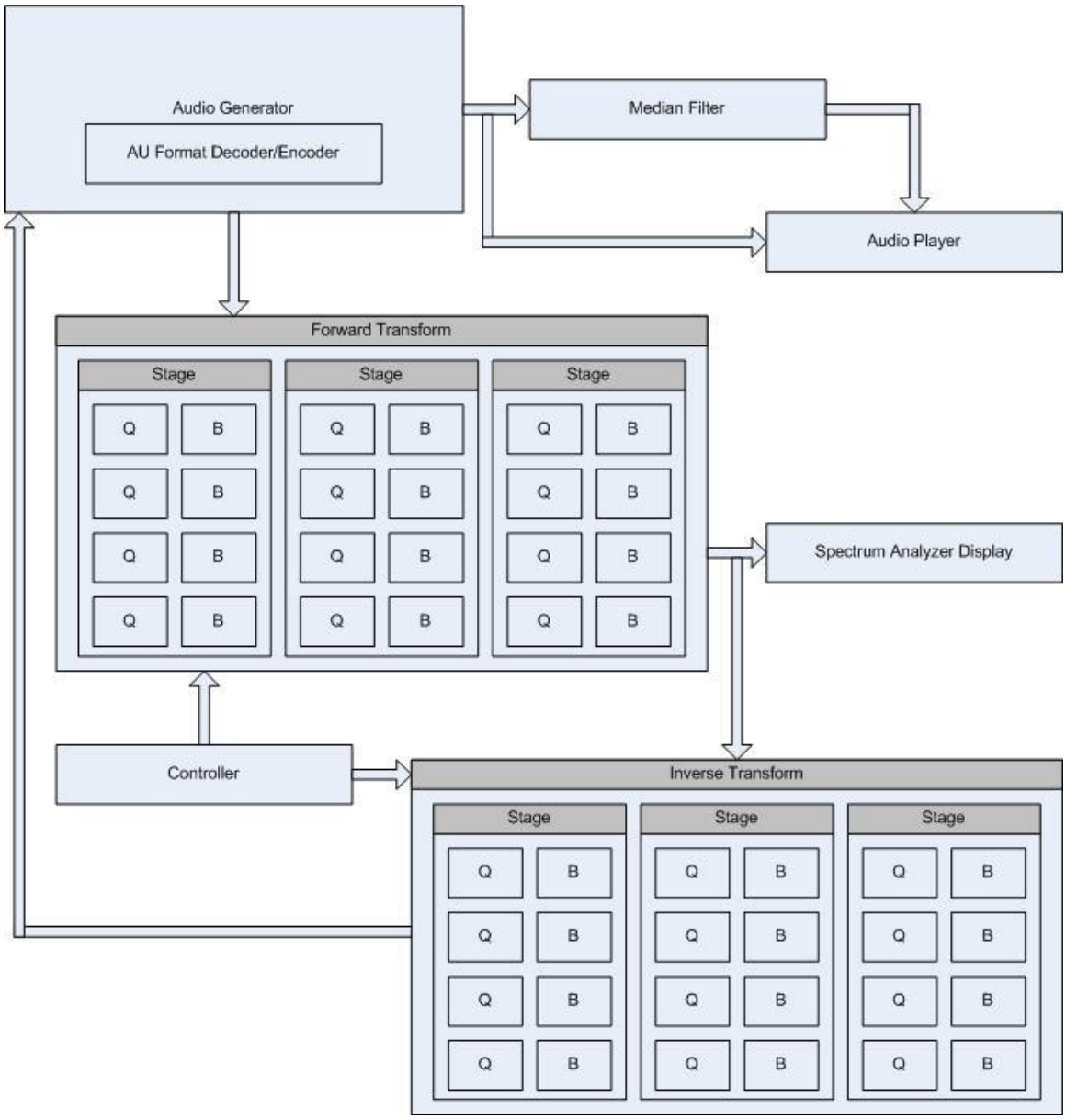


Figure 1.2 : Block diagram of a Simulated System

1.5 CONTRIBUTIONS

1. Discrete Event models for signal processing systems like pipelined Radix - 2 & Radix – 4 FFT, Audio Player, Spectrum Analyzer etc. have been formally modeled and implemented that can provide fast and effective simulations of signal processing systems. These models can also be used for building other complex DSP systems.

2. Quantization has been introduced into the computation of FFT and the system has been analyzed for the effects of quantization. This reduces the time taken for the computations.

3. Stages of the FFT are pipelined and the speed-up due to pipelining has been compared and analyzed for Radix – 2 and Radix – 4 transforms.

4. Discrete Event Model with callback events and Discrete Event Model with self – timed events have been used which helps in bridging the outputs of the discrete-event systems with discrete-timed systems. For example, Audio Player is a Discrete Event Model with callback events and plays the audio at the correct rate and uses callback events to retrieve buffers from the driver. Spectrum Analyzer is a Discrete Event Model with self-timed events that self-times itself to display the spectrum in synchronization with the audio player model.

2 ALGORITHMS AND MATH

This chapter illustrates the important algorithms that have been used in the thesis. Also, the associated math and equations are shown at the required places.

2.1 DISCRETE FOURIER TRANSFORM

Discrete Fourier Transform (DFT) is a very powerful computational tool for frequency analysis of discrete time signals. The DFT and Inverse DFT of a signal are given by the following formulae:

DFT

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \qquad k = 0, 1, 2, \mathrm{K}\ , N-1$$

Equation – 2.1 : DFT

IDFT

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j2\pi kn/N} \qquad k = 0, 1, 2, \mathrm{K}\ , N-1$$

Equation – 2.2 : IDFT

2.2 FAST FOURIER TRANSFORM

The Fast Fourier transform (FFT) [1] is a Discrete Fourier transform algorithm, which reduces the number of computations, needed with the divide and conquer approach. The DFT requires N^2 complex multiplications where as the FFT requires only (N/2) log2N complex multiplications in the case of Radix-2.

2.2.1 RADIX – 2 FFT

When the number of data points is a power of 2, the sequence can be split up into even and odd data points or two N/2 point sequences and derived in such a way till individual two-point sequences. The algorithm is explained more in detail below.

2.2.1.1 DECIMATION IN FREQUENCY

The process of dividing the frequency components into even and odd parts, gives this algorithm its name. The block diagram of a Radix-2 Decimation-in-Frequency algorithm is shown below:

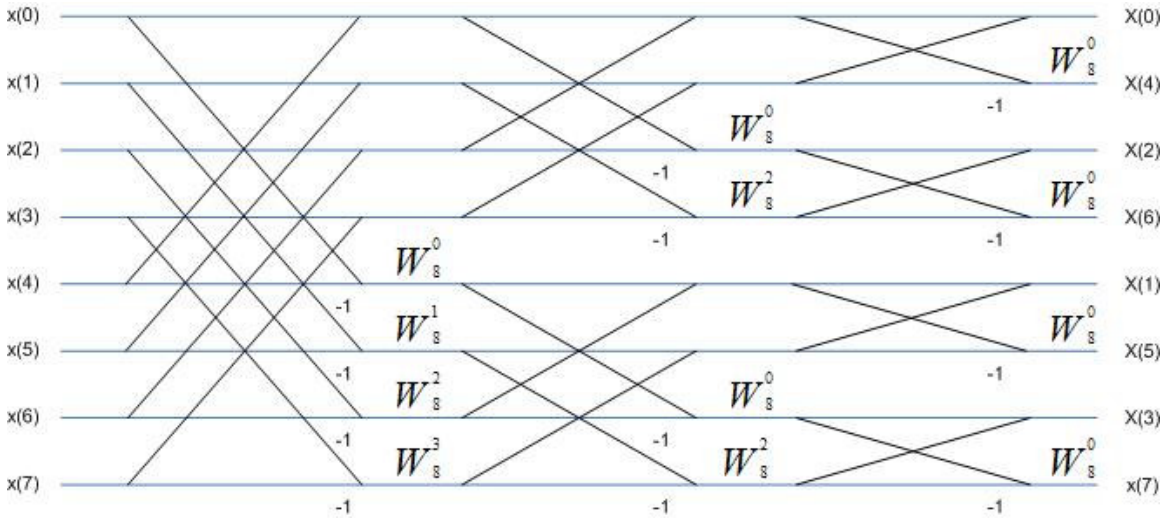


Figure 2.1 : Radix – 2 Decimation in Frequency FFT

The Basic Butterfly for the algorithm is shown below:

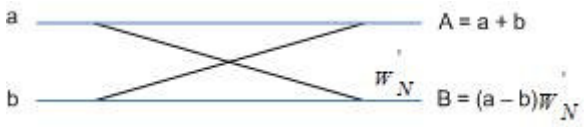


Figure 2.2 : Basic Butterfly of Radix – 2 Decimation in Frequency FFT

[*Digital Signal Processing: Principles, Algorithms and Applications : John G. Proakis, Dimitris G. Manolakis, Third Edition, Page 460*]

2.2.1.2 DECIMATION IN TIME

The process of splitting the time domain sequence into even an odd samples gives the algorithm its name. The block diagram of a Radix-2 Decimation-in-Time algorithm is shown below:

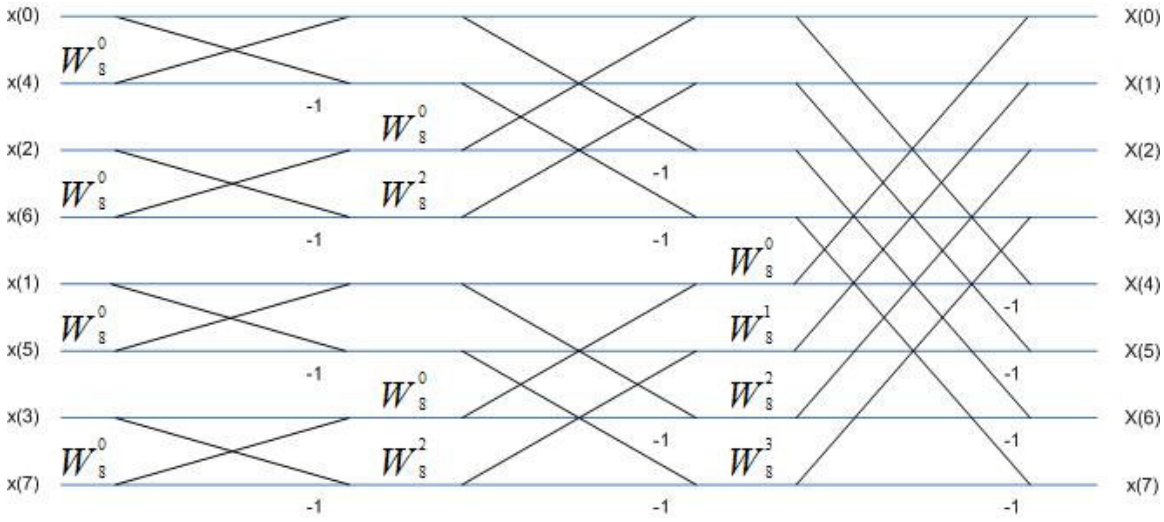


Figure 2.3: Radix – 2 Decimation in Time FFT

The Basic Butterfly for the algorithm is shown below:

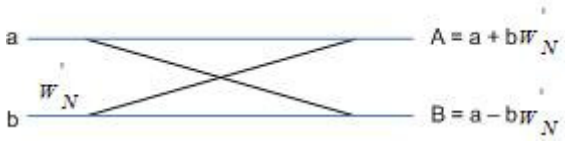


Figure 2.4 : Basic Butterfly of Radix – 2 Decimation in Time FFT

[*Digital Signal Processing: Principles, Algorithms and Applications : John G. Proakis, Dimitris G. Manolakis, Third Edition, Page 464*]

2.2.2 RADIX – 4 FFT

When the number of data points is a power of 4, the sequence can be split up into sequences of (n mod 4) data points or four N/4 point sequences and derived in such a way till individual four-point sequences. The algorithm is explained more in detail below.

2.2.2.1 DECIMATION IN FREQUENCY

The process of splitting the frequency domain sequence into (n mod 4) samples gives the algorithm its name. The block diagram of a Radix-4 Decimation-in-Frequency algorithm is shown below:

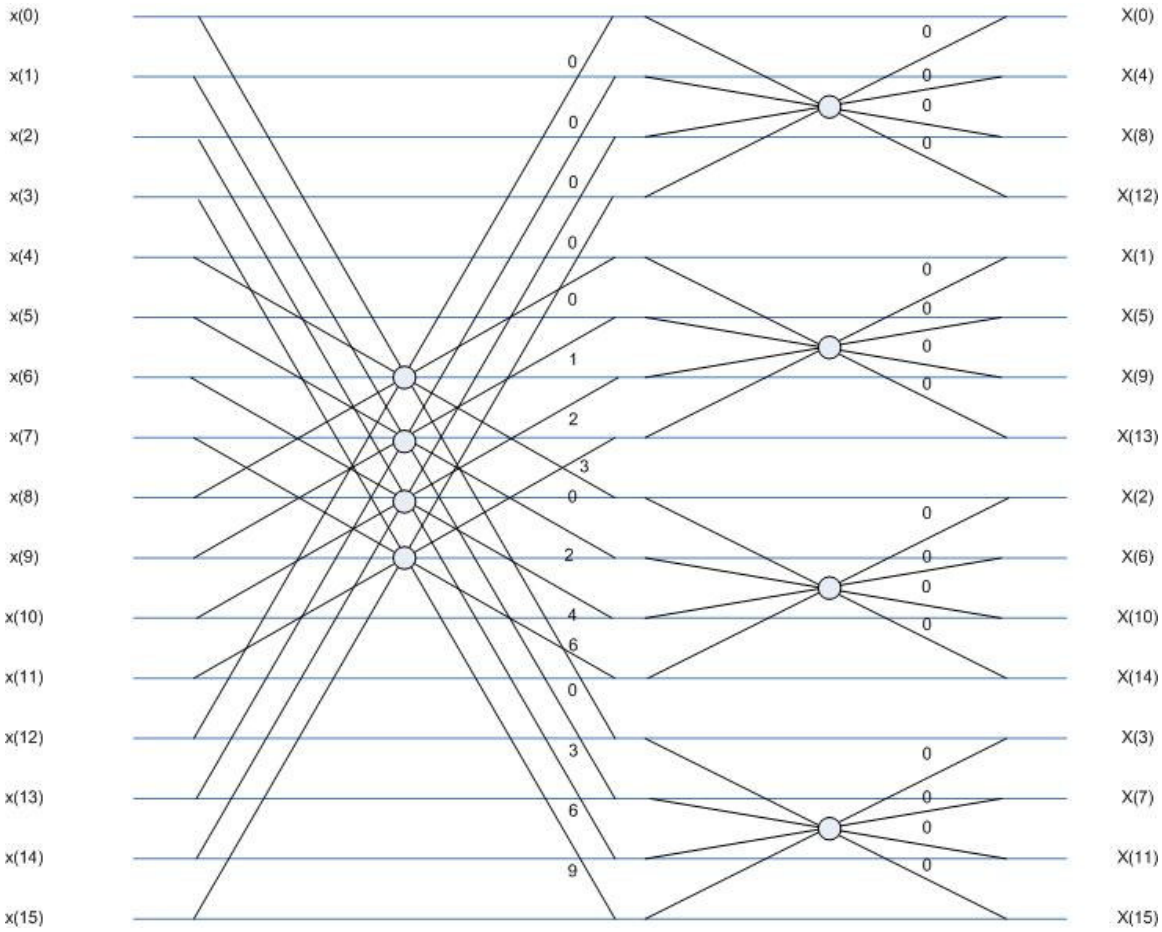


Figure 2.5: Radix – 4 Decimation in Frequency FFT

2.2.2.2 DECIMATION IN TIME

The process of splitting the time domain sequence into (n mod 4) samples gives the algorithm, its name. The block diagram of a Radix-4 Decimation-in-Time algorithm is shown below:

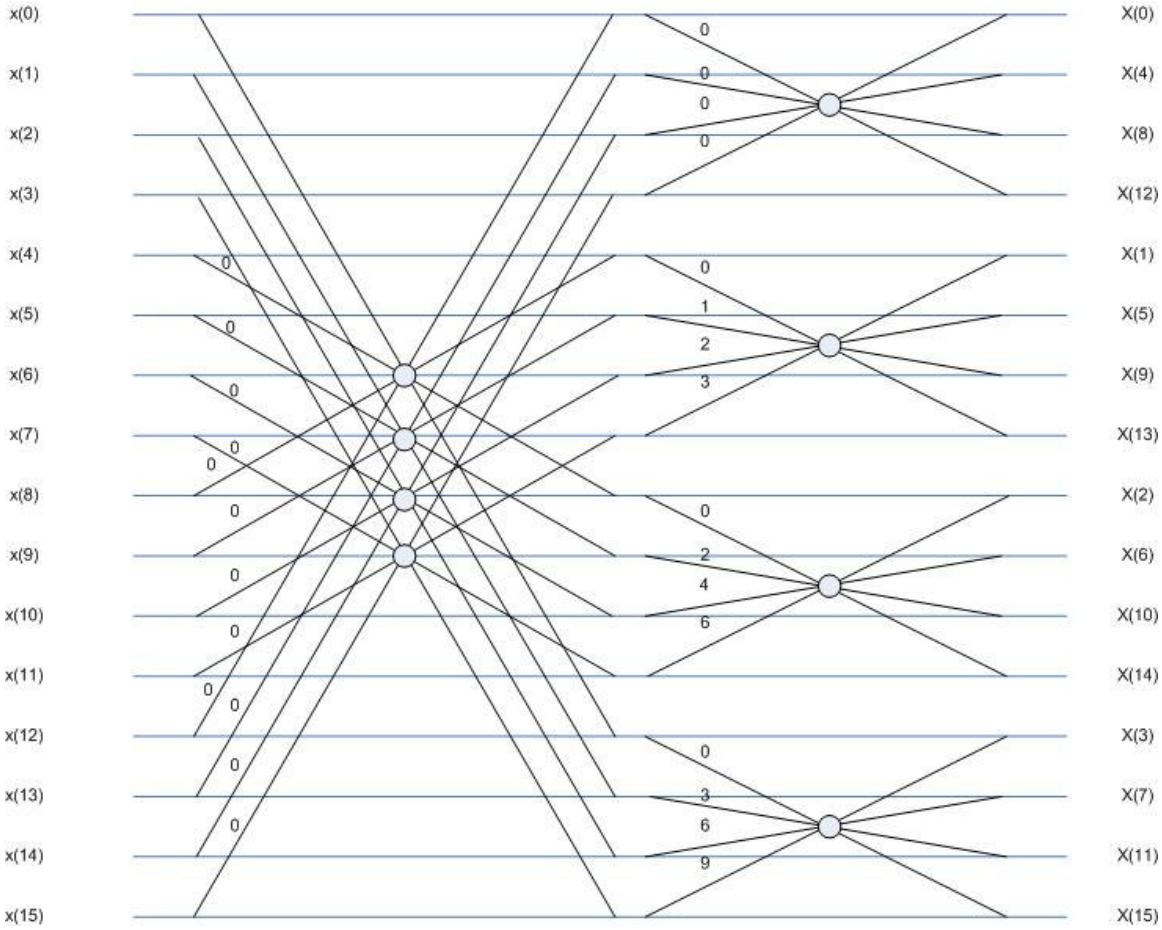


Figure 2.6: Radix – 4 Decimation in Time FFT

2.3 PIPELINING THE FFT STAGES

A Radix – 2 FFT has $\log_2 N$ stages in its transformation of data into frequency domain. These stages can be pipelined and each of these stages can work on separate sets of data in parallel (the data is fed to the successive stages). This reduces the execution time to $\log_2 N$ times of single pass FFT, if there are $\log_2 N$ arithmetic units to perform the computation. In our design, there is a single butterfly for every stage of FFT that sequentially works on all the input data and performs the computations in-place. Once the every stage is done processing, the controller is informed which keeps track of the stages and regulates the data flow.

2.4 COMPUTATIONAL QUANTIZATION

The FFT also takes a lot of processor clock cycles. A Radix-2 butterfly takes a 2 complex additions and a complex multiplication and a Radix-4 butterfly takes 12 complex additions and 3 complex multiplications.

In a real-time signal (high sampling rate), the rate of change of the amplitude of the data is not going to be drastic most of the time. It might happen that the complex multiplications and additions would be performed on almost constant data most of the time. If this is taken into consideration, an experimental quantum value can be applied to the FFT stages, depending on which the butterfly would be computed or not. So, if a set of data is seen by the butterfly, it compares it with the previous value based on the

quantum and either calculates the output or just copies the previously stored output. This would enable the saving of the processor clock cycles as the complex multiplication and addition is not performed. However, the compromise is on the quality (which depends upon the correctness of the output) and memory taken up by the data to be stored.

2.4 MEDIAN FILTER

The median filter helps in reducing the sudden surges in the amplitude of the signal that might occur due to the quantization. It helps in reducing the noise in the output. If this can be added as a analog filter, this would help in accomplishing the noise reduction, reducing the overhead. [26], explains such an approach. The median filter performs better than the mean filter, because the noise is not averaged, but eliminated most of the times due to the sorting of the samples. A similar module was built and simulated in the digital domain to test the working and experiment real-time data.

2.5 COMPLEX ARITHMETIC USING SIMD

This section shows the complex arithmetic using AMD's 3DNow! SIMD instruction set. The 3DNow! enhanced instruction set provides DSP instructions to effectively perform fast DSP complex math.

A 3DNow!/MMX register is 64-bits wide and can store two packed single-precision floating point numbers in a single register. This allows us to load a complex number with

its real and imaginary parts into a single register for computation. The following tables show addition and multiplication using Enhanced 3DNow! instruction set.

MOVQ	MM0, MMWORD PTR a
PFADD	MM0, MMWORD PTR b
MOVQ	MMWORD PTR a, MM0

Table 2.1 COMPLEX ADDITION USING 3DNow!

MOVQ	MM0, MMWORD PTR a
PSWAPD	MM1, MM0
PFMUL	MM0, MMWORD PTR b
PFMUL	MM1, MMWORD PTR b
PFPNACC	MM0, MM1
MOVQ	MMWORD PTR a, MM0

Table 2.2 COMPLEX MULTIPLICATION USING 3DNow!

If we do not consider the move instructions, a complex addition takes only 1 clock cycle and a complex multiplication takes only 4 clock cycles. So, this is used for calculating the time taken in the experiments.

3 FORMAL MODELS

This chapter shows the formalization of models used to build the system. The input ports, output ports, states, internal and external transitions and outputs have been shown for all the models that have been used.

3.1 BUTTERFLY

The butterfly model takes r complex numbers as input depending on the radix and performs the computations as shown in chapter 2. This is the elementary unit used for building up the stages of the FFT.

3.1.1 MODEL (ATOMIC)

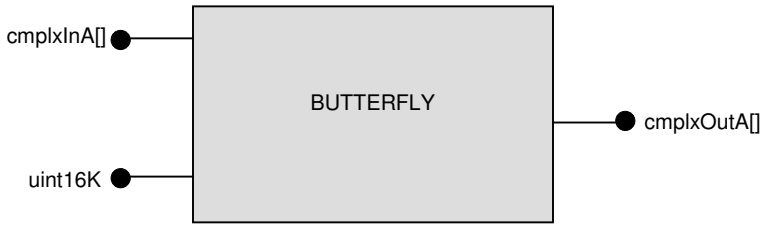


Figure 3.1 Butterfly Model

3.1.2 PARAMETERS

N : The N-point Fast Fourier Transform where N denotes the block size

Quantum : The quantum for comparisons of inputs with stored inputs

3.1.3 INPUT PORTS

X = { cmplxInA[], uint16InK }

cmplxInA[] is an array of Complex inputs. It is of size 2 when it is a Radix-2 butterfly of size –4 if it is a Radix-4 butterfly and uint16K is an 16-bit unsigned integer input.

3.1.4 OUTPUT PORTS

Y = { cmplxOutA[] }

cmplxOutA is an array of complex outputs. It is of size 2 when it is a Radix-2 butterfly of size –4 if it is a Radix-4 butterfly.

3.1.5 STATES

S = {passive (S0), calculate (S1)} x R_0^+

There are only two states. The model always stays passive, unless there is an input on all the three ports that transitions it to the "calculate" phase. The comparisons and calculations are performed and the model passivates after comparison time or calculation time, depending on whether the outputs are actually calculated.

3.1.6 EXTERNAL TRANSITION

δ_{ext}(passive, ∞, cmplxInA[], uint16InK, e)

/* if (cmplxInA[] equals (prevCmplxInA[], quantum)) */

= (calculate, TIME_COMPARE)

/* else */

= (calculate, TIME_COMPARE + TIME_CALCULATE)

If the current set of inputs fall within the quantum of the previous set then the state if held for the comparison time otherwise for the comparison and calculation time.

3.1.7 INTERNAL TRANSITION

δ_{int}(calculate) = (passive, ∞)

The model passivates whenever there is an internal transition.

3.1.8 OUTPUTS

λ(calculate, σ) = cmplxOutA[]

The calculation phase outputs the values before the model passivates in the internal transition.

3.2 STAGE

A stage model couples to a butterfly model and takes care of the data flow. There is a single butterfly model for every stage which is used by the stage to perform its computations.

3.2.1 MODEL (ATOMIC)

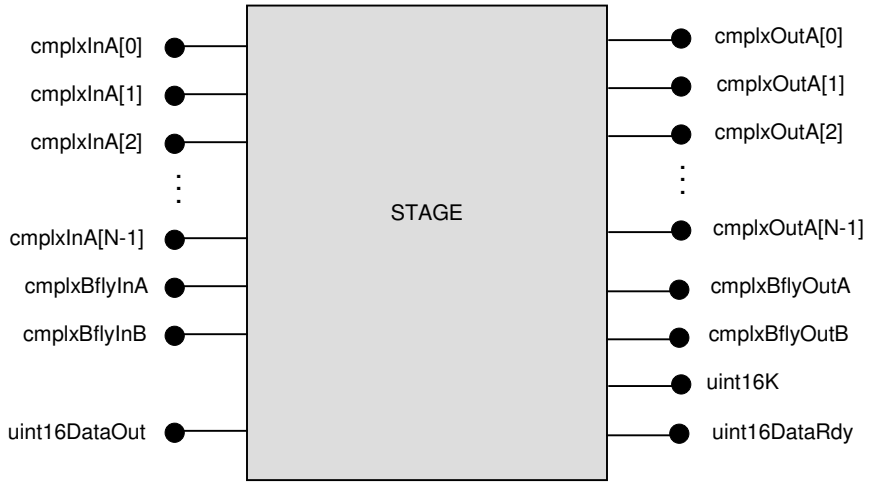


Figure 3.2 Stage Model

3.2.2 PARAMETERS

N : The N-point Fast Fourier Transform where N denotes the block size

Id : The stage's id (0 based index) in an array of stages

3.2.3 INPUT PORTS

X = { cmplxInA[0 : N-1], cmplxBflyInA, cmplxBflyInB, uint16DataOut }

cmplxInA[0 : N-1] are the Complex inputs on which the data for processing come in.

cmplxBflyInA and cmplxBflyInB are Complex inputs that are connected to the Butterfly unit to accept the outputs of the butterfly.

uint16DataOut is a 16-bit unsigned integer input, on receiving which the processed data is sent out.

3.2.4 OUTPUT PORTS

Y = { cmplxOutA[0 : N-1],cmplxBflyOutA,cmplxBflyOutB,uint16InK,uint16DataRdy }

cmplxOutA[0 : N-1] are the Complex outputs which send the processed data out.

cmplxBflyOutA and cmplxBflyOutB are complex outputs, that send the data out to the Butterfly unit.

uint16InK and uint16DataRdy are 16-bit unsigned integer ports for sending the value of "K" to the butterfly and signaling Data Ready to the Controller.

NOTE: There cmplxBflyOutA, cmplxBflyOutB and uint16K are removed in the revised model.

3.2.5 STATES

S = {passive (S0), send(S1), dataOut(S2)} x R_0^+

The model always stays passive, unless there is an input on all the Complex data input ports that transitions it to the "send" phase. The "send" phase sends pairs of data to the butterfly and passivates. The data on the butterfly input ports "send" phase and at the last

receive of data from butterfly signals uint16DataRdy. When uint16DataOut is received at this point the data is sent out.

3.2.6 EXTERNAL TRANSITION

δ_{ext}(passive, ∞, cmplxInA[0 : N – 1], e, state == 0)

= (send, 0)

δ_{ext}(passive, ∞, cmplxBflyInA, cmplxBflyInB, e, state <= N/2)

= (send, 0)

δ_{ext}(passive, ∞, uint16DataOut, e, state == N/2 + 1)

= (dataOut, 0)

3.2.7 INTERNAL TRANSITION

δ_{int}(send) = (passive, ∞)

δ_{int}(dataOut) = (passive, ∞)

The model passivates whenever there is an internal transition.

3.2.8 OUTPUTS

λ(send, σ) = cmplxBflyOutA, cmplxBflyOutB, uint16K

λ(dataOut, σ) = cmplxOutA[0 : N – 1]

3.3 STAGE MODULE

3.3.1 MODEL (DIGRAPH)

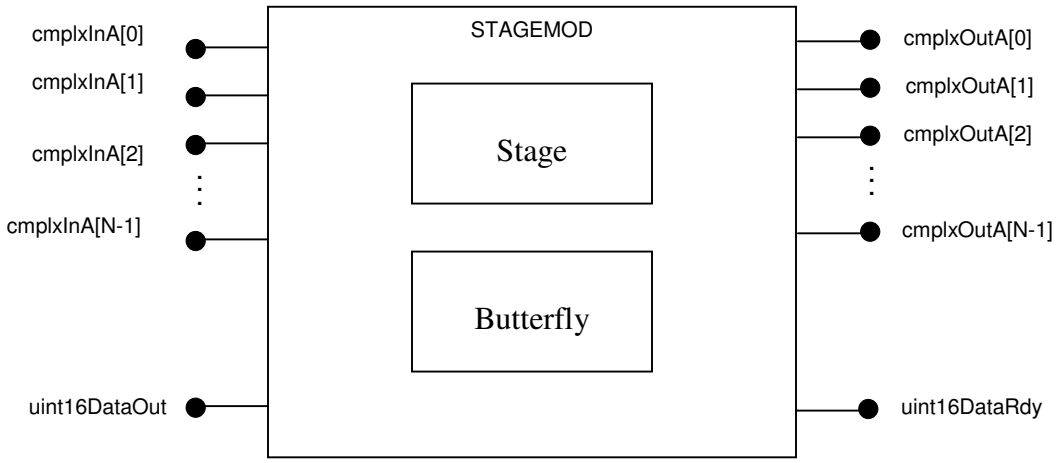


Figure 3.3 Stage Module

3.3.2 PARAMETERS

N : The N-point Fast Fourier Transform where N denotes the block size

Id : The stage's id (0 based index) in an array of stages

Quantum : The quantum for comparisons of inputs with stored inputs

3.3.3 INPUT PORTS

X = { cmplxInA[0 : N-1], uint16DataOut }

cmplxInA[0 : N-1] are the Complex inputs on which the data for processing come in. It is coupled to the Stage Atomic Model.

uint16DataOut is a 16-bit unsigned integer input, on receiving which the processed data is sent out. It is also coupled to the corresponding input port on the Stage Atomic Model.

3.3.4 OUTPUT PORTS

Y = { cmplxOutA[0 : N-1], uint16DataRdy }

cmplxOutA[0 : N-1] are the Complex outputs which send the processed data out.

uint16DataRdy is16-bit unsigned integer port for signaling Data Ready to the Controller.

3.4 QFFT MODULE

This model has $\log_r N$ stages coupled together serially to form a pipelined transform. This model accepts N inputs and works on the inputs to transform them to frequency domain. The heart of this unit is the stages that are coupled together and are responsible for the data processing.

3.4.1 MODEL (DIGRAPH)

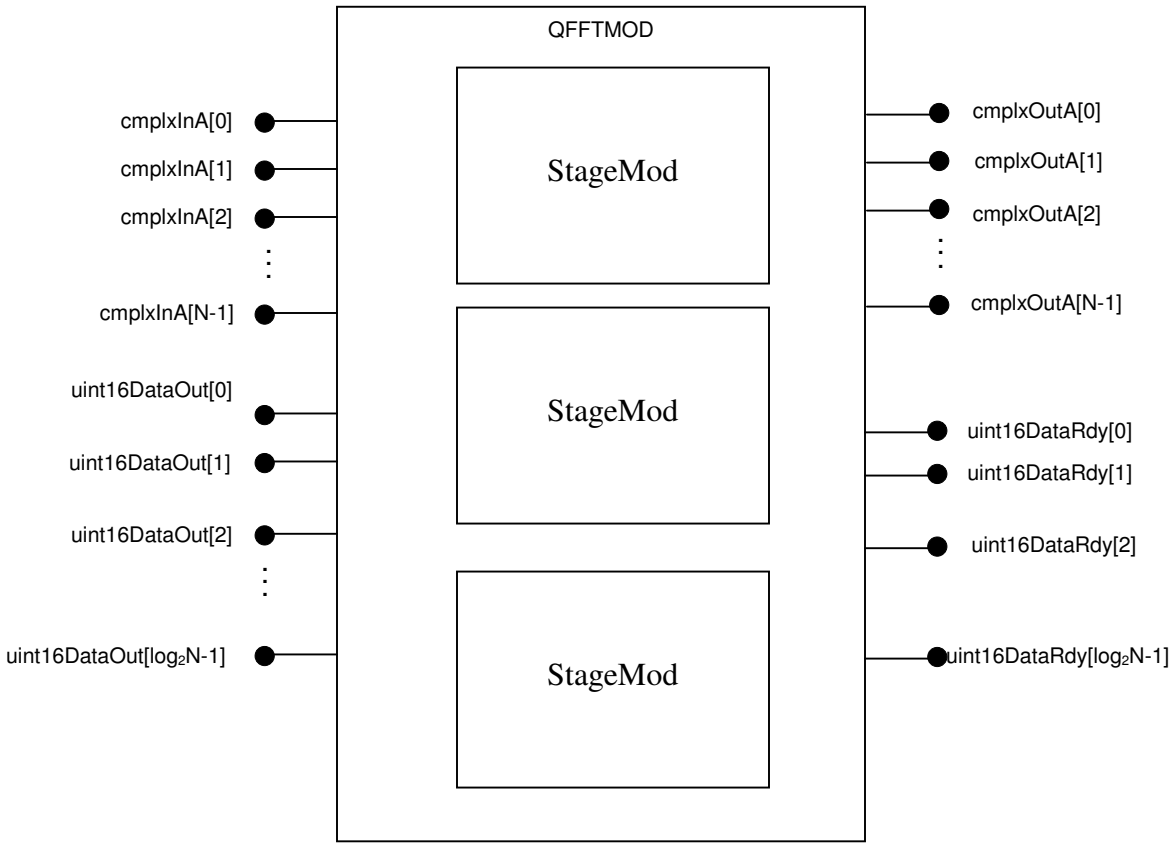


Figure 3.4 QFFTMod Model

3.4.2 PARAMTERS

N : The N-point Fast Fourier Transform where N denotes the block size

Quantum : The quantum for comparisons of inputs with stored inputs

b_inverse_transform : This parameter is used to calculate progressive quantum for the stages depending on Forward or Inverse transform.

s8_pq_type : This parameter denotes the progressive-quantization type.

3.4.3 INPUT PORTS

X = { cmplxInA[0 : N-1], uint16DataOut[0 : $\log_R$N-1] }

cmplxInA[0 : N-1] are the Complex inputs on which the data for processing come in. It is coupled to the Stage Module.

uint16DataOut[0 : $\log_R$N-1] are 16-bit unsigned integer inputs, which are coupled to the respective stage modules based on the id.

3.4.4 OUTPUT PORTS

Y = { cmplxOutA[0 : N-1], uint16DataRdy[0 : $\log_R$N-1] }

cmplxOutA[0 : N-1] are the Complex outputs which send the processed data out.

uint16DataRdy[0 : $\log_R$N-1] are 16-bit unsigned integer ports for signaling Data Ready to the Controller and are coupled to the respective stage modules based on the id.

3. 5 SYSTEM CONTROLLER

System Controller controls the data flow between the stages. This system has a forward transform and an inverse transform module. The controller keeps track of the status of every stage and regulates them to send the data out and accepts an input from the stages once they are ready for the next set for data.

3.5.1 MODEL (ATOMIC)

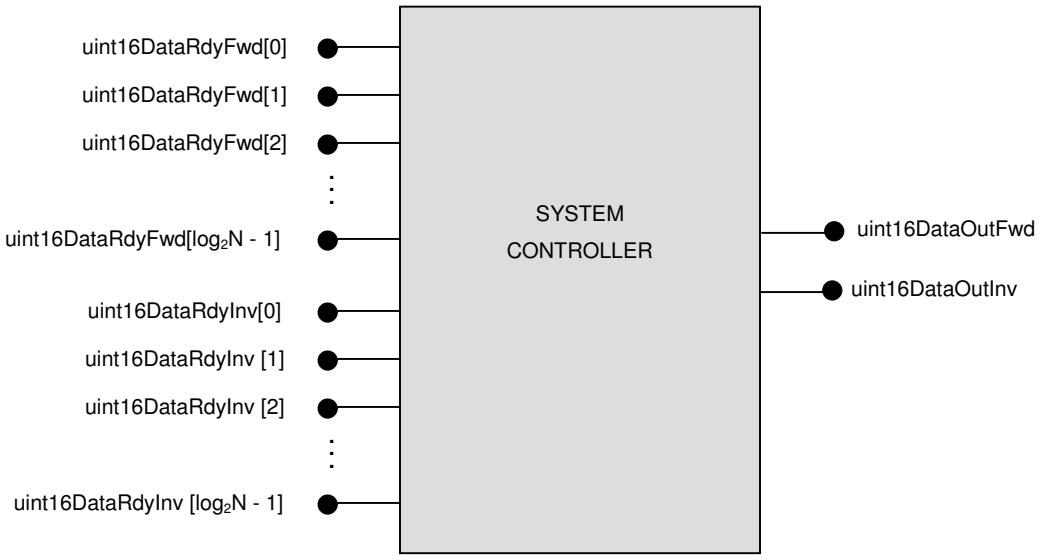


Figure 3.5 System Controller Model

3.5.2 PARAMETERS

N : The N-point Fast Fourier Transform where N denotes the block size

3.5.3 INPUT PORTS

X = { uint16DataRdyFwd[0 : $\log_2$N -1], uint16DataRdyInv[0 : $\log_2$N -1] }

uint16DataRdyFwd[0 : $\log_2$N -1] and uint16DataRdyInv[0 : $\log_2$N -1] are data ready inputs from the forward and inverse transform modules respectively.

3.5.4 OUTPUT PORTS

Y = { uint16DataOutFwd, uint16DataOutInv }

uint16DataOutFwd, uint16DataOutInv are the dataOut signals to the forward and inverse transform blocks.

3.5.5 STATES

S = {passive (S0), dataOut(S1) } x R_0^+

The model always stays passive, and counts the inputs on the forward and inverse transforms and after a simple logic to determine if all the stages are ready for next input, it sends the dataOut signals on the respective ports.

3.5.6 EXTERNAL TRANSITION

δ_{ext}(passive, ∞, uint16DataRdyFwd[0 : $\log_2$N -1], uint16DataRdyInv[0 : $\log_2$N -1], e)

= (passive, 0) /* if (!dataOut) */

δ_{ext}(passive, ∞, uint16DataRdyFwd[0 : $\log_2$N -1], uint16DataRdyInv[0 : $\log_2$N -1], e)

= (dataOut, 0) /* if (dataOut) */

3.5.7 INTERNAL TRANSITION

δ_{int}(dataOut) = (passive, ∞)

The model passivates whenever there is an internal transition.

3.5.8 OUTPUTS

λ(dataOut, σ) = uint16DataRdyFwd, uint16DataRdyInv

3.6 SYSTEM

3.6.1 MODEL (DIGRAPH)

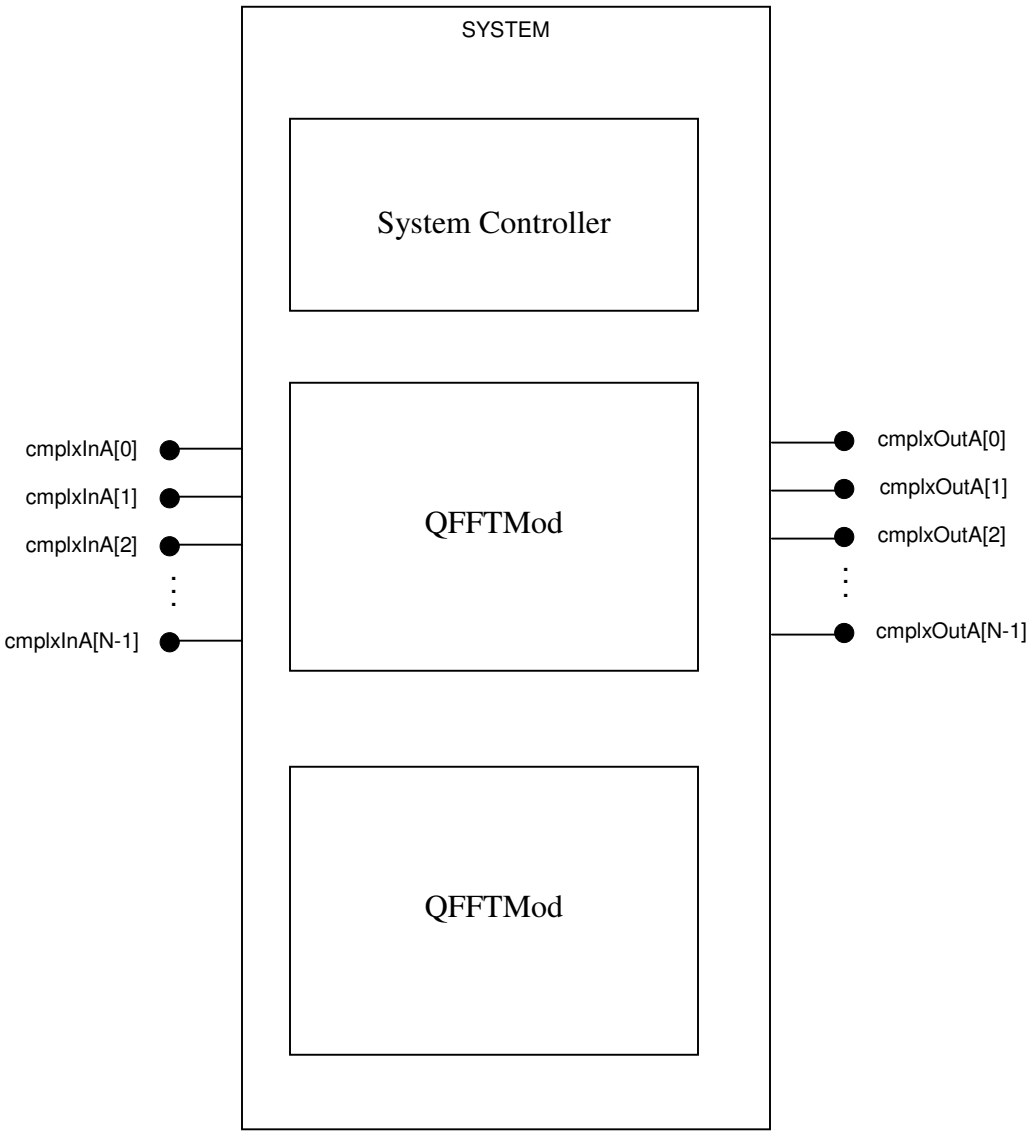


Figure 3.6 - System Model

3.6.2 PARAMETERS

N : The N-point Fast Fourier Transform where N denotes the block size

Quantum : The quantum for comparisons of inputs with stored inputs

3.6.3 INPUT PORTS

X = { cmplxInA[0 : N-1] }

cmplxInA[0 : N-1] are the Complex inputs on which the data for processing come in. It is coupled to the QFFT Module.

3.6.4 OUTPUT PORTS

Y = { cmplxOutA[0 : N-1], uint16DataOutGenr}

cmplxOutA[0 : N-1] are the Complex outputs which send the processed data out.

uint16DataOutGenr is coupled to the System Controller's uint16DataOutFwd. It can be used as an input to the generator to signal the send of next set of data.

3.7 GENERATOR

3.7.1 MODEL (ATOMIC)

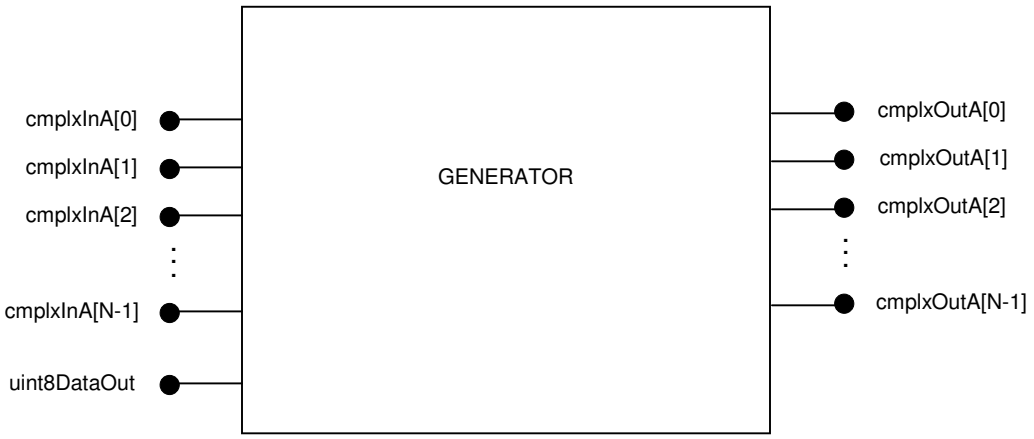


Figure 3.7 Generator Model

3.7.2 PARAMETERS

N : The N-point Fast Fourier Transform where N denotes the block size

3.7.3 INPUT PORTS

X = { cmplxInA[0 : N-1], uint16DataOut }

cmplxInA[0 : N-1] are the Complex inputs on which processed data is received.

uint16DataOut is a 16-bit unsigned integer input, on receiving which the next set of data is sent out.

3.7.4 OUTPUT PORTS

Y = { cmplxOutA[0 : N-1]}

cmplxOutA[0 : N-1] are the Complex outputs which send the data out.

3.7.5 STATES

S = {passive (S0), send(S1) } x R_0^+

The model always stays passive, unless there is an input on all the Complex data input ports that transitions it to the "send" phase. The "send" phase sends pairs of data to the butterfly and passivates. The data on the butterfly input ports "send" phase and at the last receive of data from butterfly signals uint16DataRdy. When uint16DataOut is received at this point the data is sent out.

3.7.6 EXTERNAL TRANSITION

δ_{ext}(passive, ∞, uint16DataOut, e) = (send, 0)

δ_{ext}(passive, ∞, cmplxInA[0 : N – 1], e) = (passive, 0)

3.7.7 INTERNAL TRANSITION

δ_{int}(send) = (passive, ∞)

The model passivates whenever there is an internal transition.

3.7.8 OUTPUTS

λ(send, σ) = cmplxBflyOutA[0 : N – 1]

3.8 AUDIO PLAYER

3.8.1 MODEL (ATOMIC)

Figure 3.8 Audio Player Model

3.8.2 PARAMETERS

N : The N-point Fast Fourier Transform where N denotes the block size

3.8.3 INPUT PORTS

X = { RealData[0 : N-1], releaseBuffer }

RealData [0 : N-1] are the real inputs on which data to be output to the sound card is received. The releaseBuffer port accepts the free buffer at the input port and adds it to the buffer queue.

3.8.4 OUTPUT PORTS

Y = { queueBuffer}

"queueBuffer" port is used to send the prepared data to the sound card.

3.8.5 STATES

S = {passive (S0), play(S1) } x R_0^+

The model stays passive, unless there are inputs on all the real data input ports that transitions it to the "play" phase. When the buffer to be freed is received at the releaseBuffer port, the model continues in the same state with an elapsed time, where-in the freed buffer is enqueued in the buffer queue.

3.8.6 EXTERNAL TRANSITION

δ_{ext}(passive, ∞, RealData[0 : N – 1], e) = (play, 0)

δ_{ext}(phase, σ, releaseBuffer, e) = (phase, σ – e, queue.Enqueue(releaseBuffer))

where phase = passive or play

When the model is passive and there is data in the input ports, it prepares the data for output and transitions to the "play" phase where a buffer is dequeued from the buffer queue. When there is an input in the releaseBuffer port, the model continues in the same phase and the released buffer is enqueued onto the buffer queue.

3.8.7 INTERNAL TRANSITION

δ_{int}(play, σ, queue) = (passive, ∞, queue.Dequeue())

The model passivates whenever there is an internal transition after dequeuing a buffer from the buffer queue.

3.8.8 OUTPUTS

λ(play, σ, queue) = queue.Front

3.9 SPECTRUM ANALYZER

3.9.1 MODEL (ATOMIC)

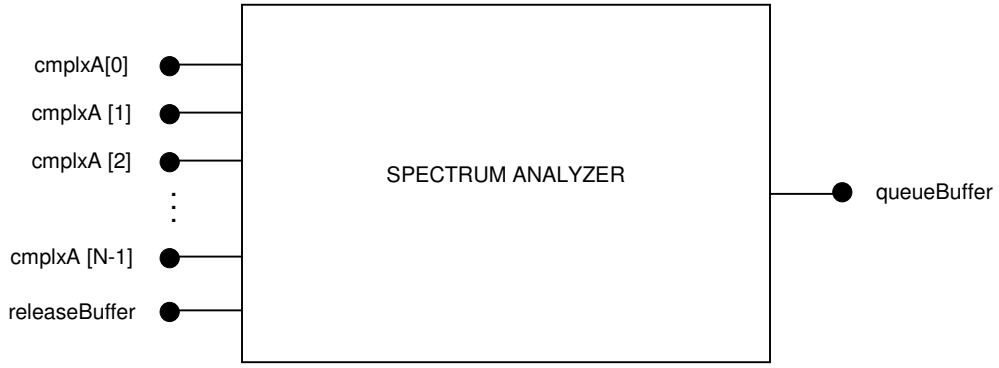


Figure 3.9 Spectrum Analyzer Model

3.9.2 PARAMETERS

N : The N-point Fast Fourier Transform where N denotes the block size

3.9.3 INPUT PORTS

X = { cmplxInA[0 : N-1] }

cmplxInA[0 : N-1] are the Complex inputs on which the frequency data is received.

3.8.4 OUTPUT PORTS

Y = { queueBuffer}

"queueBuffer" port is used to send the prepared data to the display.

3.9.5 STATES

S = { passive (S0), display(S1) } x R_0^+

The model always passive, unless there are inputs on all the data input ports that transitions it to the "display" phase. When the buffer to be freed is received at the releaseBuffer port, the model continues in the same state with an elapsed time, where-in the freed buffer is enqueued in the buffer queue.

3.9.6 EXTERNAL TRANSITION

δ_{ext}(passive, ∞, RealData[0 : N – 1], e) = (display, 0)

δ_{ext}(phase, σ, releaseBuffer, e) = (phase, σ – e, queue.Enqueue(releaseBuffer))

where phase = passive or display

When the model is passive and there is data in the input ports, it prepares the data for output and transitions to the "display" phase where a buffer is dequeued from the buffer queue. When there is an input in the releaseBuffer port, the model continues in the same phase and the released buffer is enqueued onto the buffer queue.

3.9.7 INTERNAL TRANSITION

δ_{int}(display, σ, queue) = (passive, ∞, queue.Dequeue())

The model passivates whenever there is an internal transition after dequeuing a buffer from the buffer queue.

3.9.8 OUTPUTS

λ(display, σ, queue) = queue.Front

3.10 MEDIAN FILTER

3.10.1 MODEL (ATOMIC)

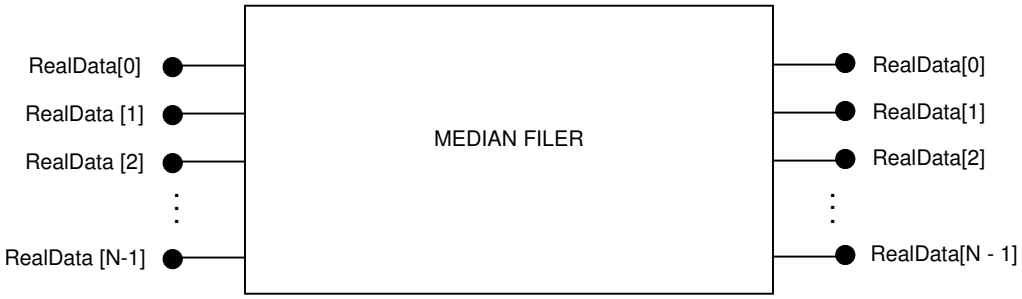


Figure 3.10 Median Filter Model

3.10.2 PARAMETERS

N : The N-point Fast Fourier Transform where N denotes the block size

3.10.3 INPUT PORTS

X = { IRealData[0 : N-1] }

RealData [0 : N-1] are the real inputs on which the data is received.

3.10.4 OUTPUT PORTS

Y = { ORealData[0 : N-1] }

RealData [0 : N-1] are the real median filtered outputs.

3.10.5 STATES

S = {passive (S0), filter(S1) } x R_0^+

The model always stays passive, unless there are inputs on all the data input ports that transitions it to the "filter" phase. The "filter" filters the data as per the median filter algorithm and passivates.

3.10.6 EXTERNAL TRANSITION

δ_{ext}(passive, ∞, IRealData[0 : N – 1], e) = (filter, 0)

3.10.7 INTERNAL TRANSITION

δ_{int}(filter) = (passive, ∞)

The model passivates whenever there is an internal transition.

3.10.8 OUTPUTS

λ(filter, σ) = ORealData[0 : N-1]

3.11 BLOCK DIAGRAM OF THE SYSTEM

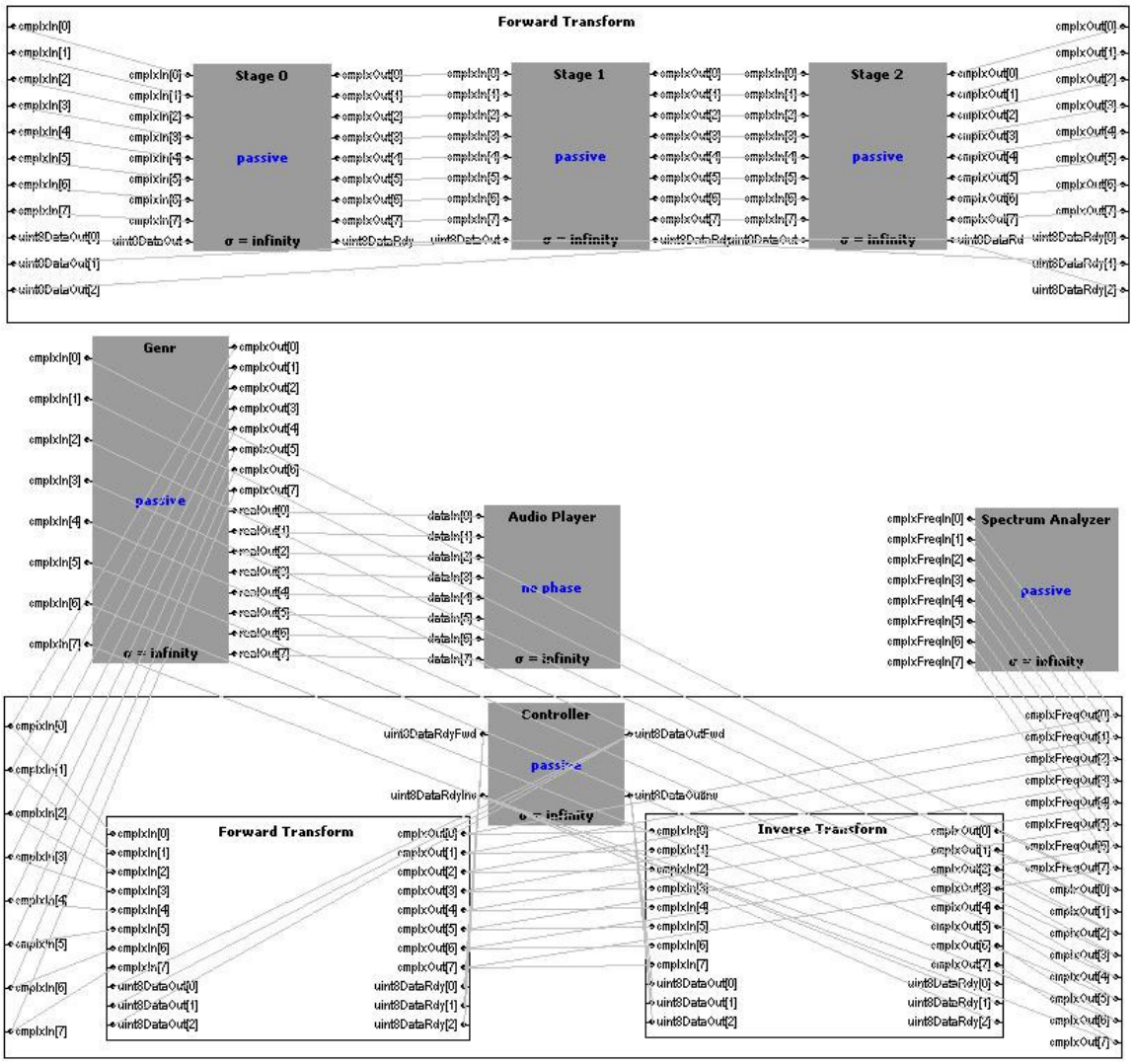


Figure 3.11 SIMVIEW CAPTURE OF SYSTEM

Figure 3.11 shows the simulated system screen captured from simView. It is the same system as intended (shown in Figure 1.2). The various components shown are the models that have been designed in the previous sections of this chapter.

4 IMPLEMENTATION

4.1 PERFORMANCE

One of the most essential criterions for design is to achieve a performance level when the implementation is done. Few of the design issues and constraints and performance enhancements are discussed below.

4.1.1 CIRCULAR ZERO COPY SHARED BUFFERS

There is a lot of data transfer involved between the different models in the simulation. Copying the data and allocating memory dynamically on the heap (scalable models), would consume a lot of time in this high data rate application. To achieve better performance, the data is put on a circular buffer that would also move along the different stages of the pipeline. There is no data copying involved right from the generator, where it comes from. The generator can retrieve/refill its buffers in similar circular fashion.

4.1.2 LOOK UP TABLES

Few of the models that would need the same set of data (for example : twiddle factors) can be put in LUTs. This is make the retrieval time for these values as O(1), which otherwise would prove very costly for the math involved. Bit-reversal (for Radix - 2) and Digit-Reversal (for Radix – 4) are similar ones that require LUTs for efficient implementation.

4.1.3 NUMBER OF PORTS

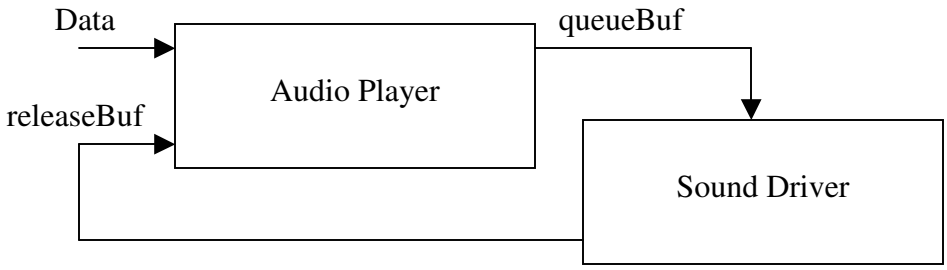
The number of ports in the model seems to affect the performance, since the simulator-coordinator has to look-up the ports and route the events/data. So, the N-ports for data were replaced by 1-port that can carry all the data. This enhanced the performance considerably for simulations.

4.2 ACTUATORS

The audio player model and spectrum analyzer model are the models that fall out of conventions. The audio player needs interaction with the sound card driver, to share its buffers and retrieve its buffers from the sound card. The spectrum analyzer on the other hand needs to interact with a display thread that would display the data onto the screen as spectrum bars. These situations demanded the usage the following models.

4.2.1 DISCRETE EVENT MODEL WITH CALLBACK EVENTS

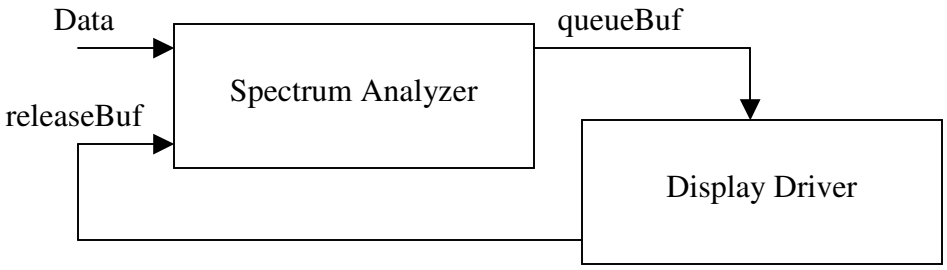
Callback events are required for the situation mentioned above. Buffer management between sound card driver and audio player model is handled through a callback function that would be called when the driver finishes up using the buffer. This event would then get a mutually exclusive access to mark the buffers as free to be used by the model. A typical setup of the model is shown below:



This model is coupled to the sound card and interacts with it through the sound card driver. The sound data buffers are received through the data port at an external transition and are queued for playing. These buffers are consumed by the sound card and returned through callback event. The callback event releases the buffers between external transitions and the next external transition retrieves these buffers for preparing the next set of data.

4.2.2 DISCRETE EVENT MODELS WITH SELF-TIMED EVENTS

The Spectrum Analyzer model needs to display the data in sync with the player. So, a dedicated thread of the model, self-times itself to display the data in its buffers after periodic intervals of time. Typically, the sleep time of the thread would be (N/ (sampling rate)) – (display calculation time).



The model prepares the display buffers during the external transition when the raw data is received. The prepared buffers are queued for playing. A display thread coordinated by the model is responsible for displaying the queued buffers at periodic intervals by interacting with the display card driver using GDI (Graphics Device Interface). The buffers are freed once they are consumed and are retrieved at the next external transition.

4.3 SCALABILITY AND REUSABILITY

This is handled by the very concept of object-oriented modeling and simulations. But configuration options have been added at places where they were required.

5 STATE CHARTS AND SEQUENCE DIAGRAMS

5.1 SYSTEM SEQUENCE

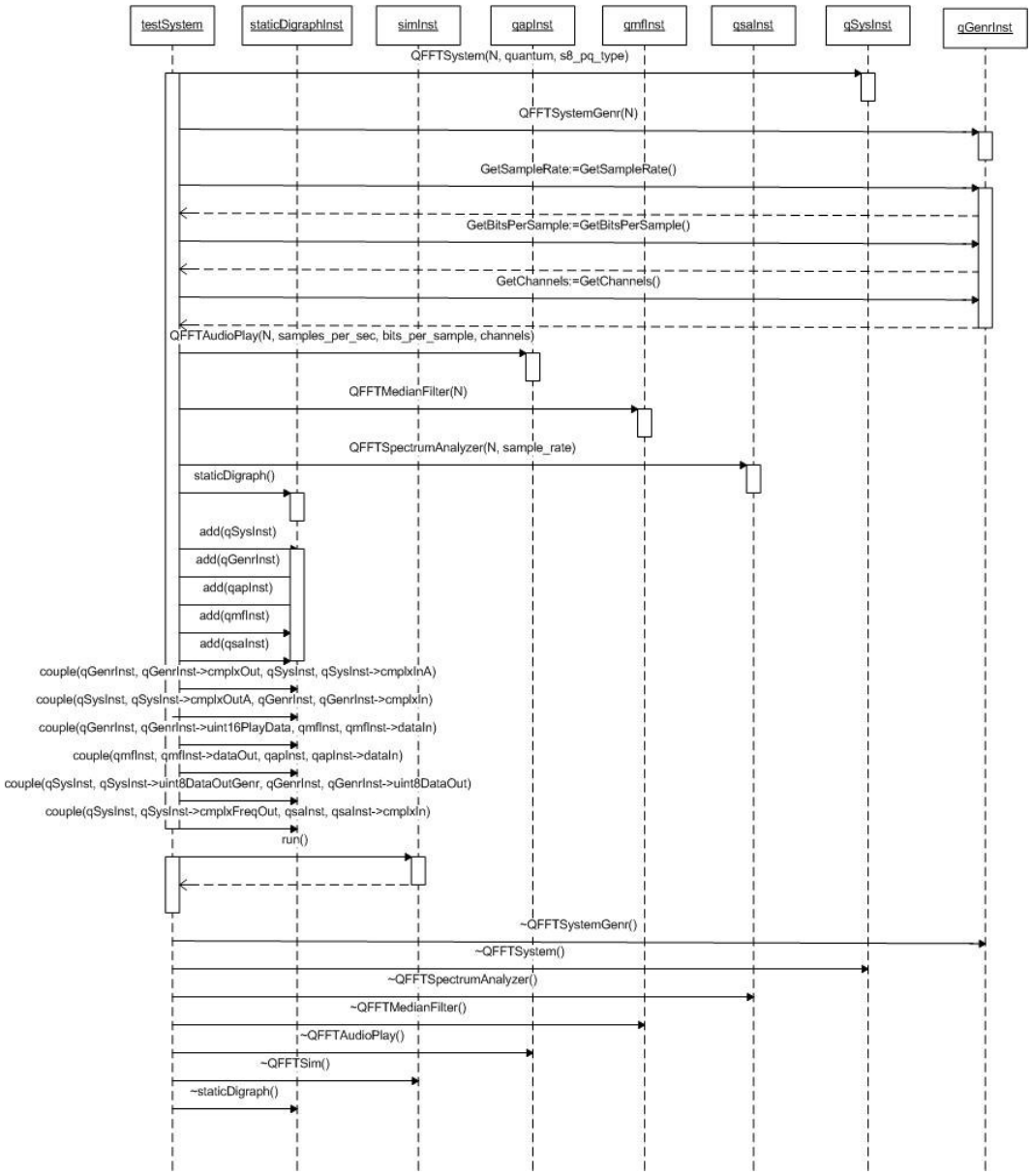


Figure 5.1 System Sequence Diagram

Figure 5.1 shows the sequence diagram of the system and the interaction between the different objects and the simulation sequence.

5.2 BUTTERFLY STATE CHART

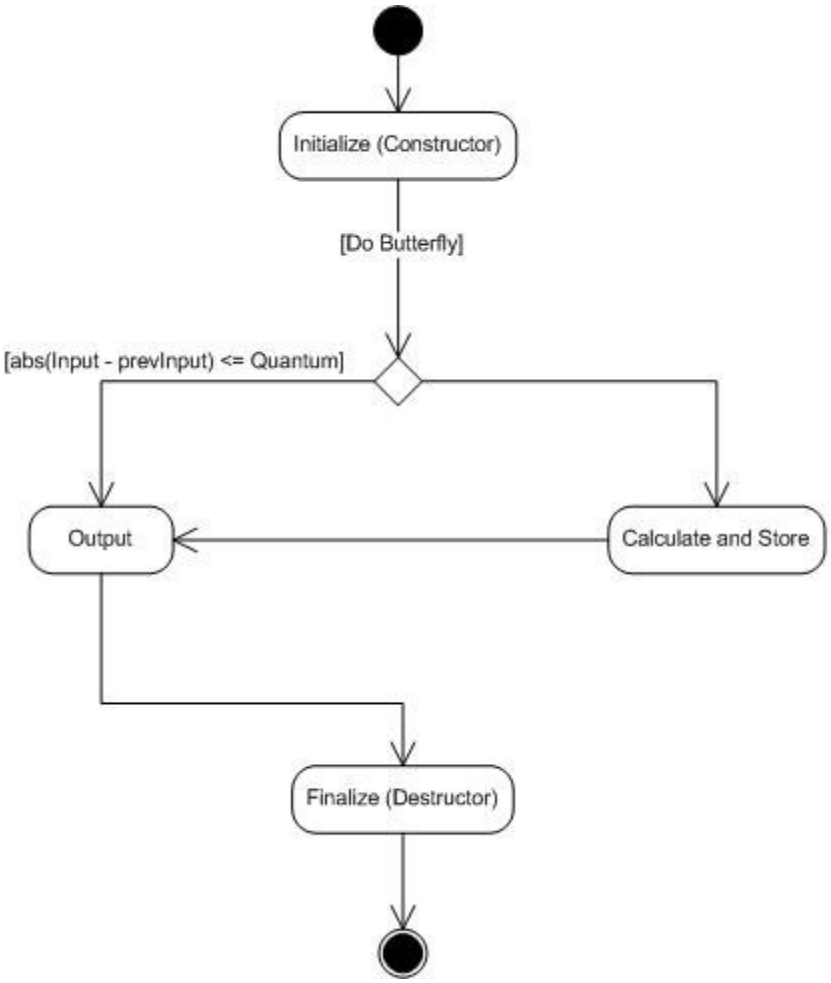


Figure 5.2 Butterfly State Chart

The above figure shows the state chart of a FFT Butterfly. It is either a Radix - 2 Butterfly or Radix – 4 butterfly based on the simulation settings. The butterfly either

calculates the output in-place or copies the previous output depending on whether the current inputs fall within the stage's quantum or not.

5.3 PLAYER STATE CHART

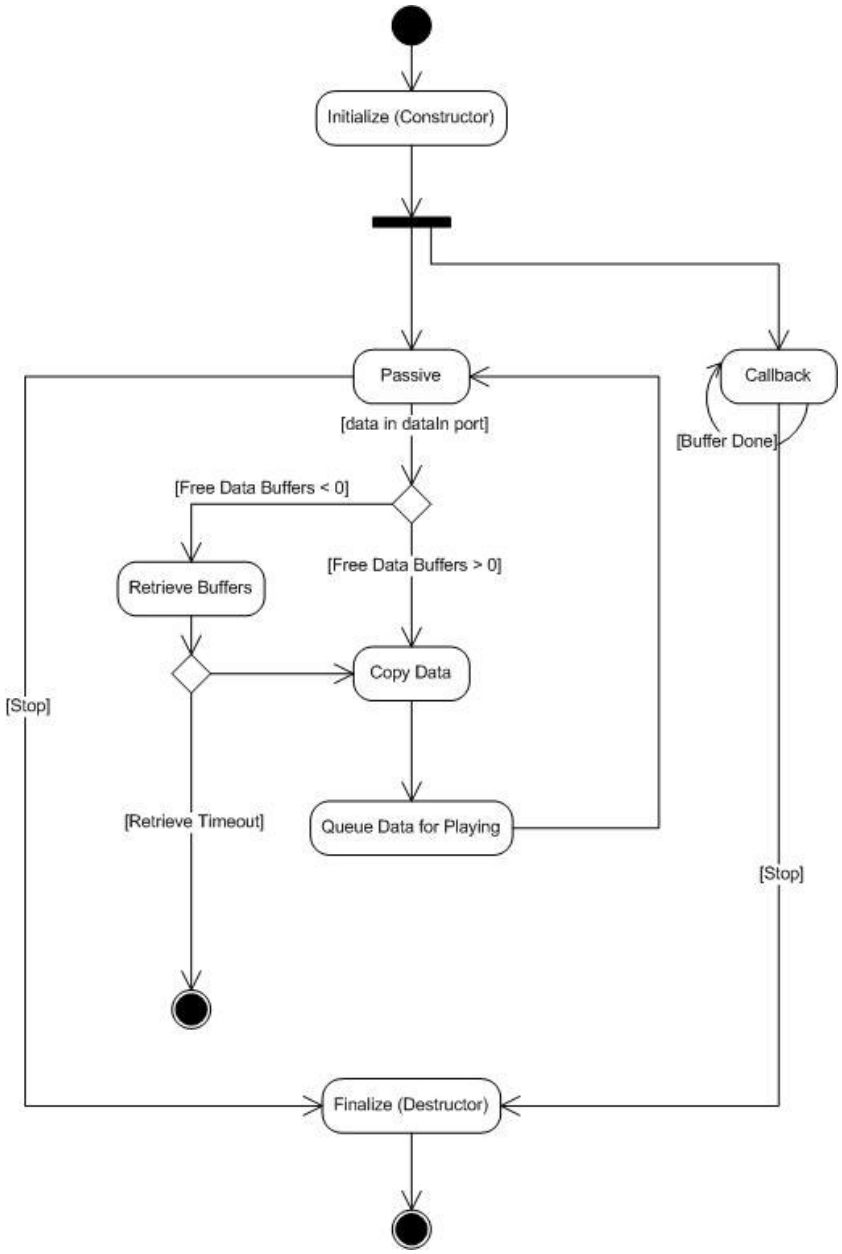


Figure 5.3 Player State Chart

Figure 5.3 shows the state chart of an Audio Player object. Whenever there is an input, it is copied and queued for playing. If there are not enough buffers, an intermediary stage tries to retrieve the buffers. There is also a parallel state, which on a callback event from the sound driver, marks the returned buffer as free and ready for refill.

5.4 MEDIAN FILTER STATE CHART

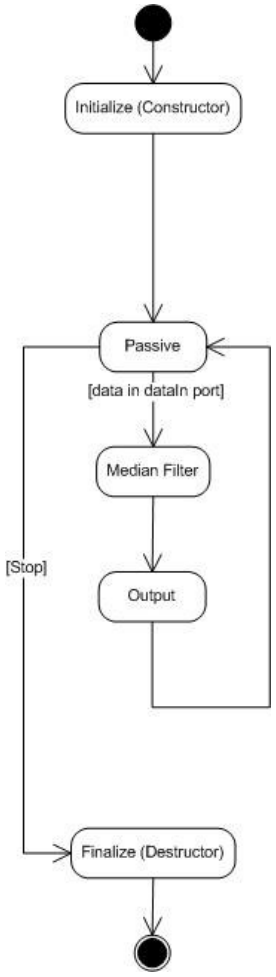


Figure 5.4 Median Filter State Chart

A median filter gets the input and adds it to the input queue and moves the head pointer of the circular queue by one. Then it serially sorts the input data by using binary search to insert the data into an already sorted array. The median of these values is the projected output for the serial input.

5.5 SPECTRUM ANALYZER STATE CHART

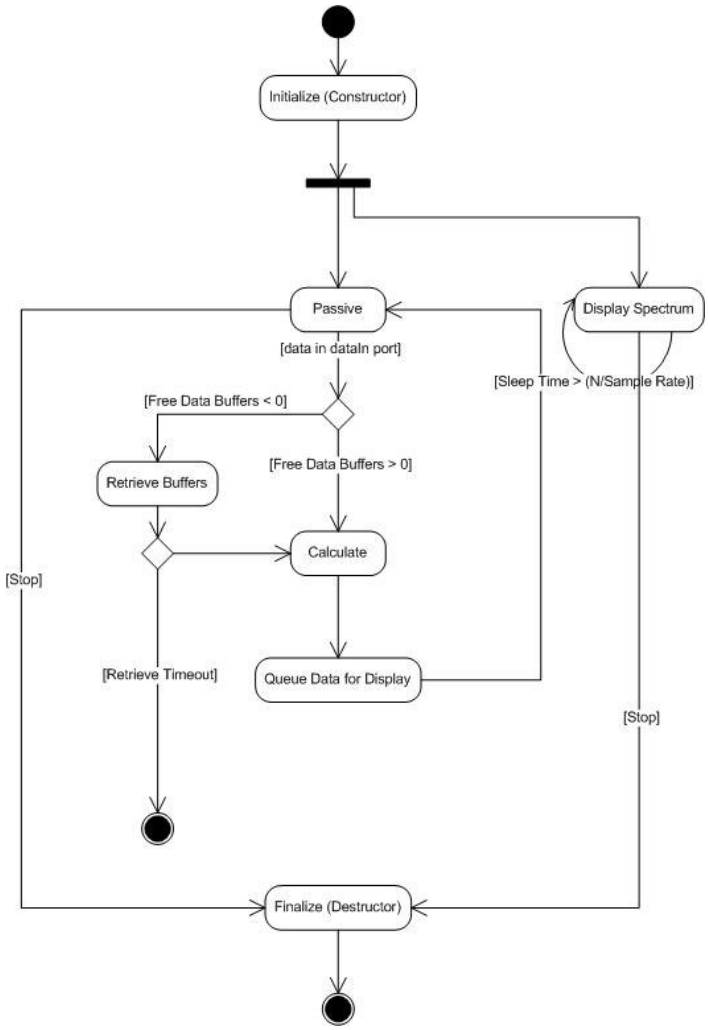


Figure 5.5 Spectrum Analyzer State Chart

The spectrum analyzer states are shown in Figure 5.5. The model, when it gets an input, copies it onto the display buffer. When there are not enough buffers available, wait states are introduced to retrieve the buffers. Now the data is queued for display. There is another parallel state which causes self-critically timed events to display. The time gap for the event is the block size/sample rate + the time taken for display. This causes the display to be synchronized with the audio player's output.

5.6 STAGE STATE CHART

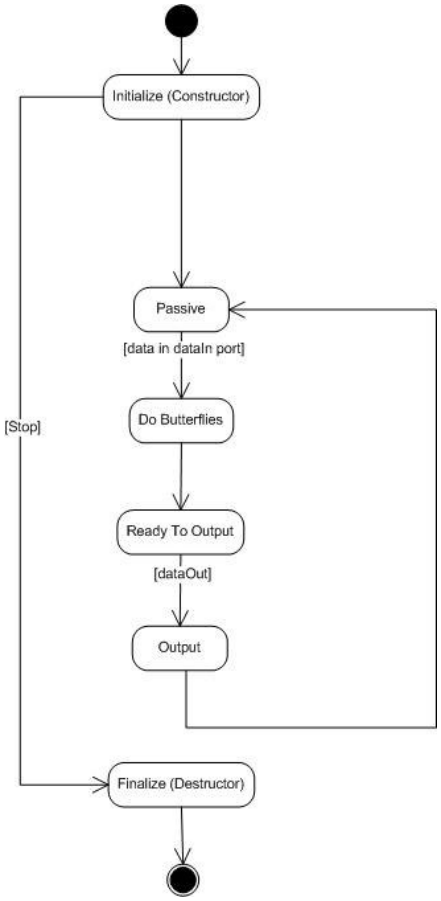


Figure 5.6 Stage State Chart

A stage model, typically calculates its butterflies sequentially over the set of input data and stays passive until there is next input. It outputs the data when the controller signals data-out event in the corresponding port.

5.7 CONTROLLER STATE CHART

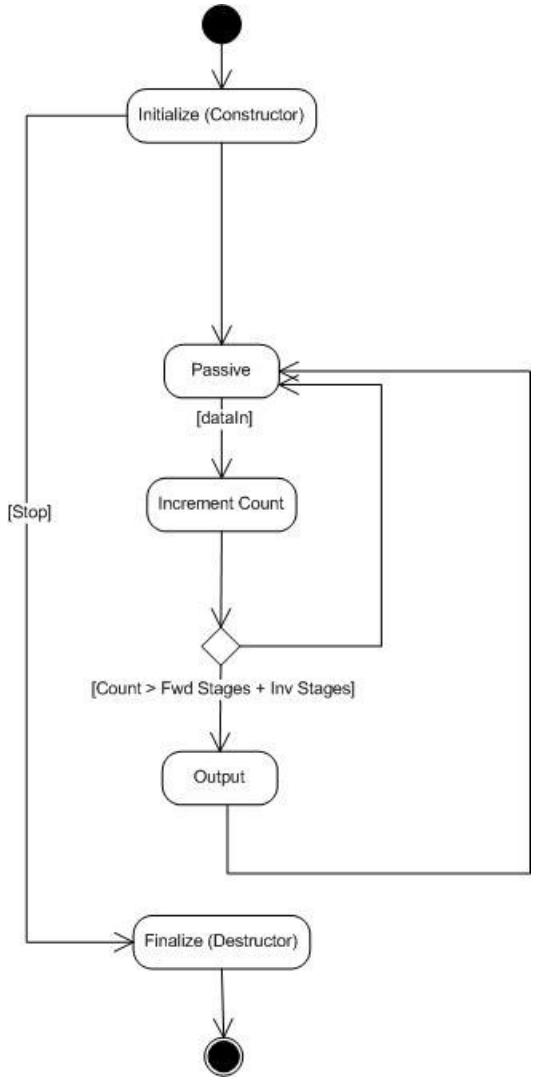


Figure 5.7 Controller State Chart

A controller is system-specific. For this system with a forward transform and an inverse transform module, the controller keeps track of which stages are ready to output and signals the events correspondingly.

5.8 GENERATOR STATE CHART

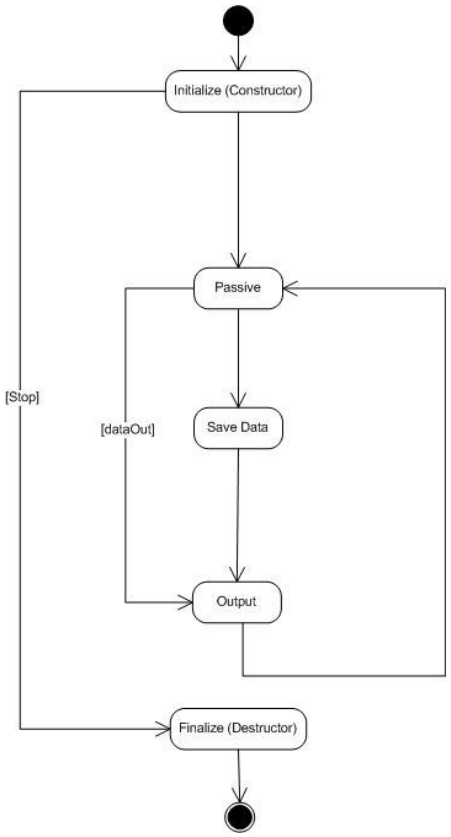


Figure 5.8 Generator State Chart

The generator inputs data to the system and gets processed data from the system. It uses the Au Encoder/Decoder module to Encode/Decode Au Formatted data.

6 EXPERIMENTS

6.1 WORKING

The experiments in the following sub-sections test the working of the models. So, a forward transform and inverse transform on a data has to produce the same output as the input.

6.1.1 Radix – 2 Forward Transform and Inverse Transform – 1024 point

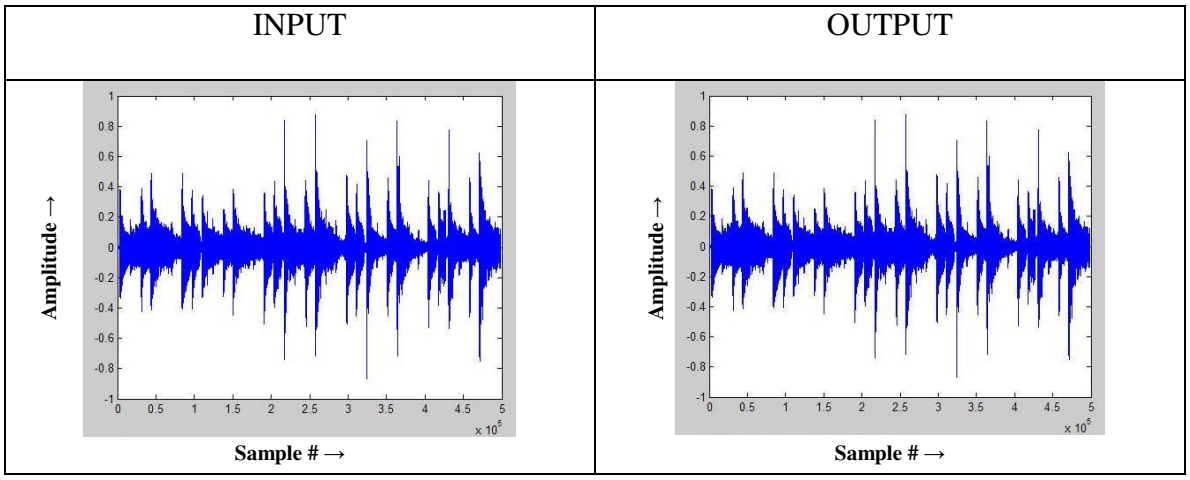


Figure 6.1 Radix – 2 Forward Transform and Inverse Transform – 1024 point

The output signal is same as the input signal (Figure 6.1). This confirms the working of the Radix – 2 FFT and the associated models.

6.1.2 Radix – 4 Forward Transform and Inverse Transform – 1024 point

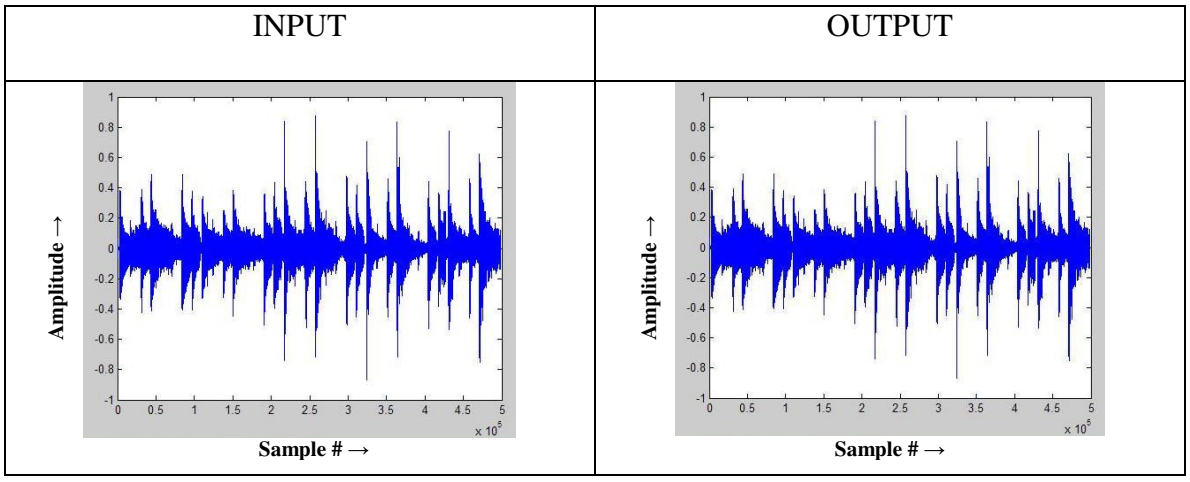


Figure 6.2 Radix – 4 Forward Transform and Inverse Transform – 1024 point

The output signal is same as the input signal (Figure 6.2). This confirms the working of the Radix – 4 FFT and the associated models.

6.2 N-POINT TRANSFORMS (PIPELINED vs. NON-PIPELINED)

This experiment tests the N-point transforms for different values of N and the corresponding time taken. The time taken is based on the values from the section 2.5.

6.2.1 RADIX – 2

DATA SIZE	N-POINT	TIME TAKEN (DFT)	TIME TAKEN (CLOCK CYCLES) (PIPELINED)	TIME TAKEN (CLOCK CYCLES) (NON-PIPELINED)
4194304	4	79691776	12582924	25165824
4194304	8	163577856	12582960	37748784
4194304	16	331350016	12583056	50331888
4194304	32	666894336	12583296	62915424
4194304	64	1337982976	12583872	75500160
4194304	128	2680160256	12585216	88088064
4194304	256	5364514816	12588288	100684032
4194304	512	10733223936	12595200	113299968
4194304	1024	21470642176	12610560	125964288
4194304	2048	42945478656	12644352	138743808
4194304	4096	85895151616	12718080	151793664
4194304	8192	171794497536	12877824	165470208
4194304	16384	343593189376	13221888	180584448
4194304	32768	687190573056	13959168	198967296

TABLE 6.1 N-POINT TRANSFORMS (PIPELINED vs. NON-PIPELINED)

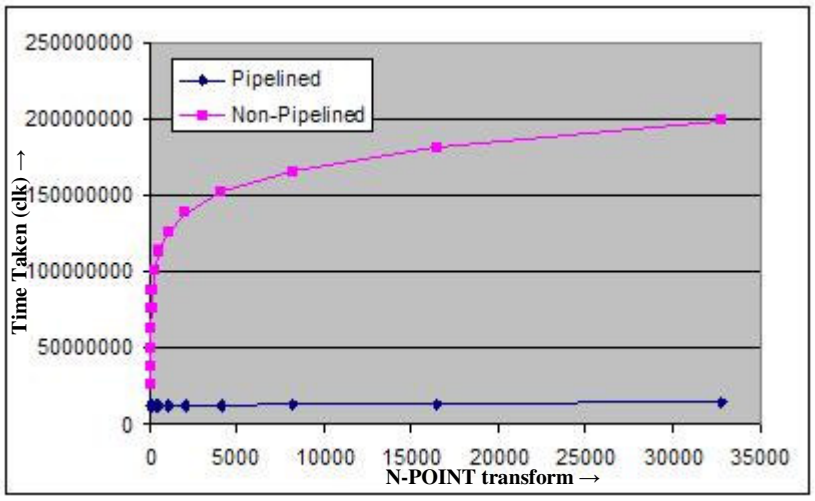


Figure 6.3 N-POINT TRANSFORMS (PIPELINED vs. NON-PIPELINED)

Table 6.1 and Figure 6.3 shows the speed-up of pipelined implementation over the non-pipelined implementation for Radix – 4 FFT. There are $\log_2 N$ stages in a pipelined implementation whereas there is only one stage in a non-pipelined implementation. So, the speedup observed should be

Time taken for pipelined implementation T_P = Time taken for non-pipelined implementation(T_{NP})/$\log_2 N$.

But, there is a zero quantum comparison that takes place, i.e. if the input is same as the previous input the calculations will not be performed, but the previously stored output values are copied. So,

$$T_P * \log_2 N >= T_{NP}$$

From Table 6.1, this can be verified. All the observed values hold to the equation.

6.2.2 RADIX – 4

DATA SIZE	N-POINT	TIME TAKEN (DFT)	TIME TAKEN (CLOCK CYCLES) (PIPELINED)	TIME TAKEN (CLOCK CYCLES) (NON-PIPELINED)
4194304	16	331350016	25165920	50331648
4194304	64	1337982976	25166592	75498240
4194304	256	5364514816	25170432	100670976
4194304	1024	21470642176	25190400	125884416
4194304	4096	85895151616	25288704	151339008
4194304	16384	343593189376	25755648	178126848

TABLE 6.2 N-POINT TRANSFORMS (PIPELINED vs. NON-PIPELINED)

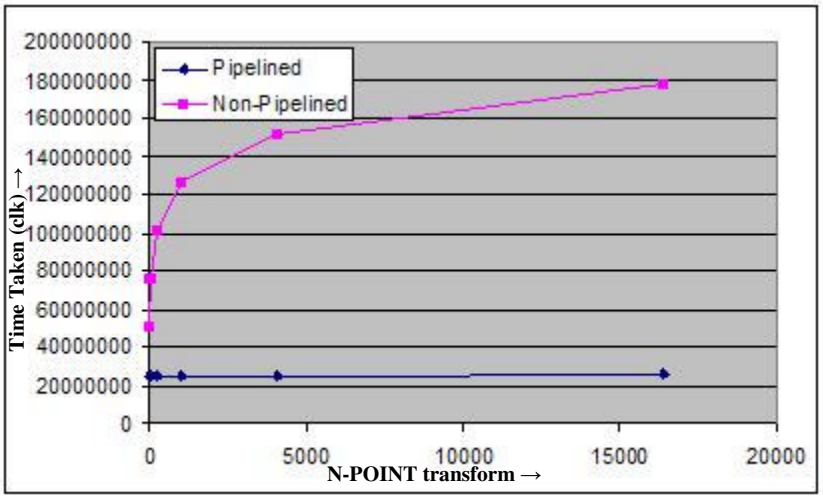


Figure 6.4 N-POINT TRANSFORMS (PIPELINED vs. NON-PIPELINED)

Table 6.2 and Figure 6.4 shows the speed-up of pipelined implementation over the non-pipelined implementation for Radix – 4 FFT. There are $\log_4 N$ stages in a pipelined implementation whereas there is only one stage in a non-pipelined implementation. Following the similar derivation as in section 6.2.1,

$$T_P * \log_4 N >= T_{NP}$$

From Table 6.2, this can be verified. All the observed values hold to the equation.

6.2.3 RADIX – 2 vs RADIX – 4

DATA SIZE	N-POINT	RADIX – 2 TIME TAKEN (PIPELINED)	RADIX – 2 TIME TAKEN (NON-PIPELINED)	RADIX – 4 TIME TAKEN (PIPELINED)	RADIX –4 TIME TAKEN (NON-PIPELINED)
4194304	16	12583056	50331888	25165920	50331648
4194304	64	12583872	75500160	25166592	75498240
4194304	256	12588288	100684032	25170432	100670976
4194304	1024	12610560	125964288	25190400	125884416
4194304	4096	12718080	151793664	25288704	151339008
4194304	16384	13221888	180584448	25755648	178126848

TABLE 6.3 N-POINT TRANSFORMS (PIPELINED vs. NON-PIPELINED)

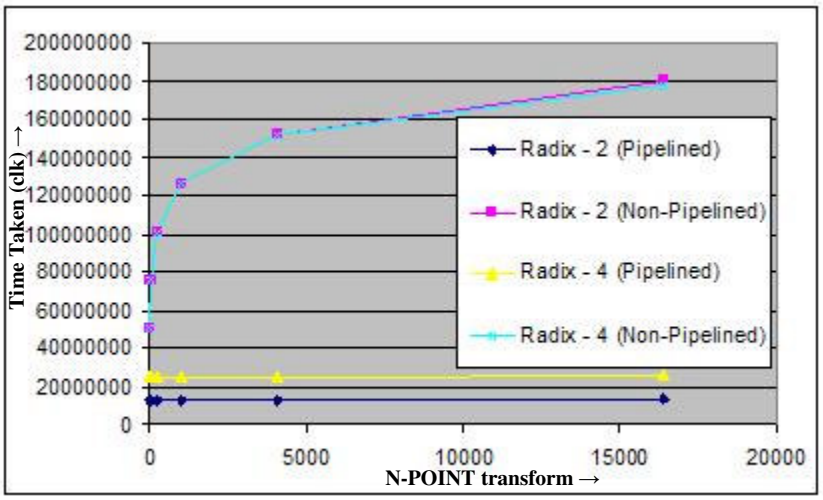


Figure 6.5 (RADIX – 2 vs RADIX - 4) (PIPELINED vs. NON-PIPELINED)

Figure 6.5 shows that the Radix – 2 and Radix – 4 non – pipelined implementation takes the same amount of time. This is because, there are 2 complex additions and one complex multiplication in a Radix – 2 butterfly and there are 12 complex additions and 3 complex

multiplications in a Radix – 4 butterfly and the values chosen for the time taken sum up to equally. There are $\log_2 N$ processing stages in Radix – 2 and there are $\log_4 N$ processing stages in Radix – 4. So, Radix – 2 performs fasters in this experiment as shown in the figure.

6.3 COMPUTATIONAL QUANTIZATION

The following sub-sections show the effects of computational quantization when applied to different sets of data in the Radix – 2 as well as Radix – 4. The Mean Square Error is also shown between the output signal (out of forward and inverse transform) and the input signal.

6.3.1 RADIX – 2

6.3.1.1 INPUT Sine wave f = 100 Hz, fs = 6000Hz

In this experiment same quantum is applied for all the stages of the forward and inverse transform. The quantum value is tabulated in the "Quantum" column.

N-POINT	Quantum	Time Taken (Clock Cycles)	$\Sigma(D_s - D_{sq})^2/(N)$
1024	0	3186176	0
1024	0.05	2290376	0.156932
1024	0.1	2185766	0.17657
1024	0.15	2108726	0.484297
1024	0.2	2069534	0.240643
1024	0.25	2090432	0.239156
1024	0.3	1980278	0.204524
1024	0.35	1947398	0.131614
1024	0.4	1872668	0.158622
1024	0.45	1822946	0.161584

TABLE 6.4 SPEED IMPROVEMENT – RADIX – 2 – SINE INPUT – SAME QUANTUM

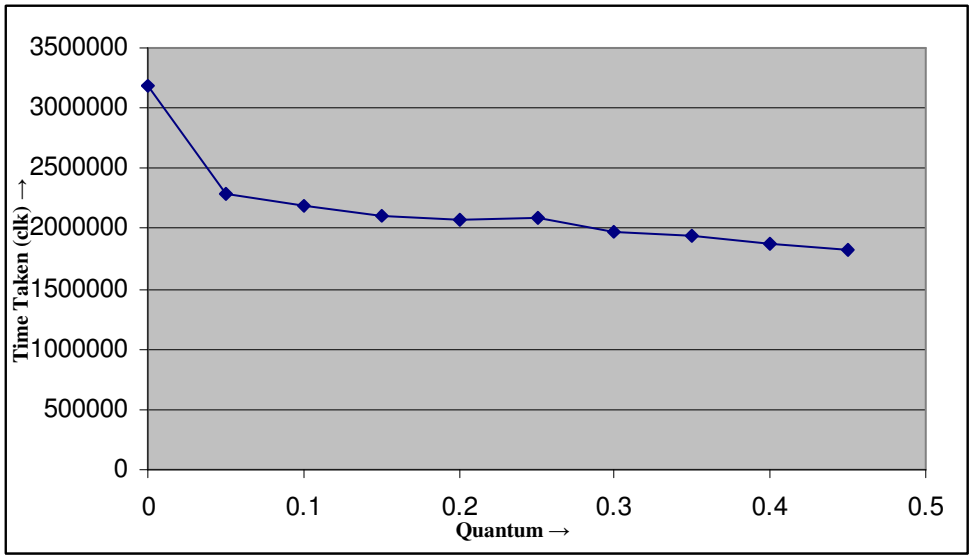


Figure 6.6 SPEED IMPROVEMENT – RADIX – 2 – SINE INPUT – SAME QUANTUM

Figure 6.6 shows the speed improvement graph with time on y-axis and quantum on x-axis.

6.3.1.2 INPUT Sine wave f = 100 Hz, fs = 22050Hz

This is another experiment of the same type with a higher sample rate.

N-POINT	Quantum	Time Taken (clock cycles)	$\sum D_s - D_{sq})^2/(N)$
1024	0	3186176	0
1024	0.05	2245934	0.158939
1024	0.1	2143814	0.274074
1024	0.15	2072570	0.368139
1024	0.2	1996046	0.311567
1024	0.25	2018618	0.363661
1024	0.3	1975952	0.326492
1024	0.35	1975592	0.345531
1024	0.4	1932302	0.30662
1024	0.45	1891154	0.31129

TABLE 6.5 SPEED IMPROVEMENT – RADIX – 2 – SINE INPUT – SAME QUANTUM

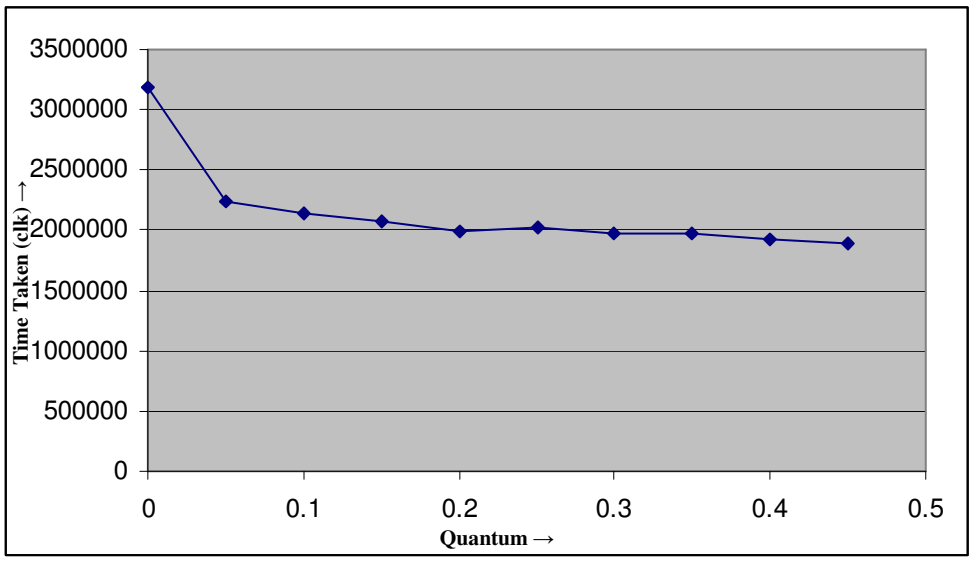


Figure 6.7 SPEED IMPROVEMENT – RADIX – 2 – SINE INPUT – SAME QUANTUM

6.3.1.3 INPUT Sine wave f = 100 Hz, fs = 44100Hz

This is another experiment of the same type with a higher sample rate.

N-POINT	Quantum	Time Taken (clock cycles)	$\sum D_s - D_{sq})^2/(N)$
1024	0	3186176	0
1024	0.05	2139362	0.212919
1024	0.1	1975628	0.261253
1024	0.15	2020676	0.274961
1024	0.2	1960538	0.251832
1024	0.25	1937594	0.205307
1024	0.3	1906676	0.220355
1024	0.35	1892708	0.215292
1024	0.4	1866182	0.263561
1024	0.45	1835672	0.310884

TABLE 6.6 SPEED IMPROVEMENT – RADIX – 2 – SINE INPUT – SAME QUANTUM

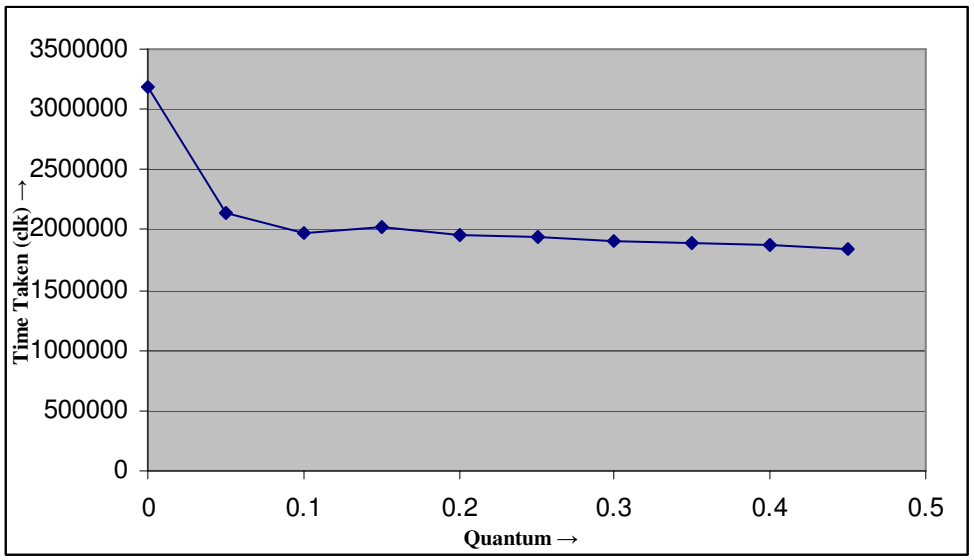


Figure 6.7 SPEED IMPROVEMENT – RADIX – 2 – SINE INPUT – SAME QUANTUM

6.3.1.4 INPUT Sine wave f = 5000 Hz, fs = 22050Hz

N-POINT	Quantum	Time Taken (clock cycles)	$\sum D_s - D_{sq})^2/(N)$
1024	0	3186176	0
1024	0.05	2342726	0.025257
1024	0.1	2379674	0.038308
1024	0.15	2415728	0.057775
1024	0.2	2409938	0.104773
1024	0.25	2400668	0.14315
1024	0.3	2390288	0.186954
1024	0.35	2378300	0.243059
1024	0.4	2367110	0.311485
1024	0.45	2401376	0.399582
1024	0.5	2393114	0.437922

TABLE 6.7 SPEED IMPROVEMENT – RADIX – 2 – SINE INPUT – SAME QUANTUM

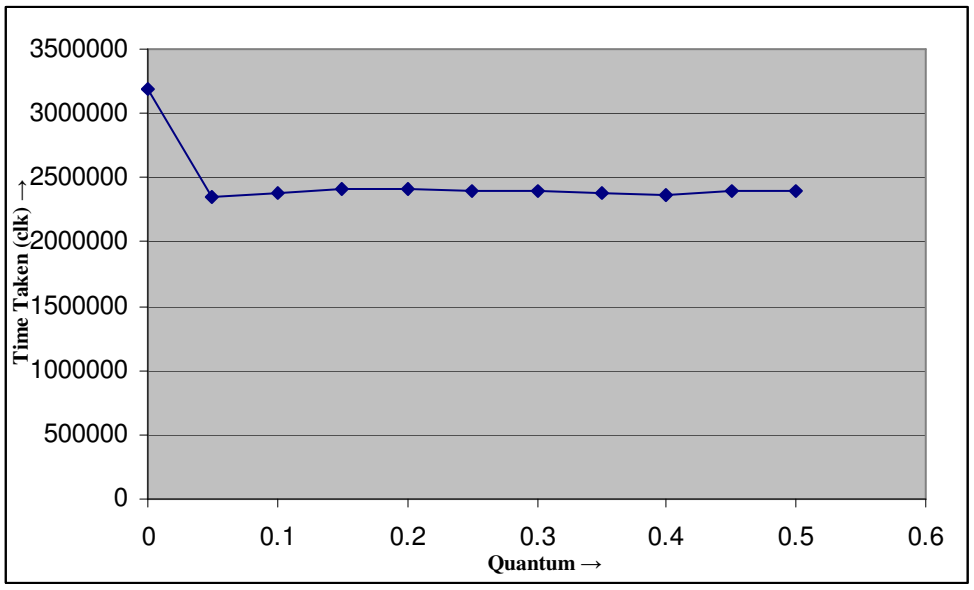


Figure 6.8 SPEED IMPROVEMENT – RADIX – 2 – SINE INPUT – SAME QUANTUM

6.3.1.5 INPUT Sine wave f = 5000 Hz, fs = 44100Hz

N-POINT	Quantum	Time Taken (clock cycles)	$\sum D_s - D_{sq})^2/(N)$
1024	0	3186176	0
1024	0.05	2409158	0.016258
1024	0.1	2385026	0.058052
1024	0.15	2381552	0.226532
1024	0.2	2320274	0.348745
1024	0.25	2312840	0.467876
1024	0.3	2285078	0.4867
1024	0.35	2267582	0.488488
1024	0.4	2252426	0.431518
1024	0.45	2187902	0.331102

TABLE 6.8 SPEED IMPROVEMENT – RADIX – 2 – SINE INPUT – SAME QUANTUM

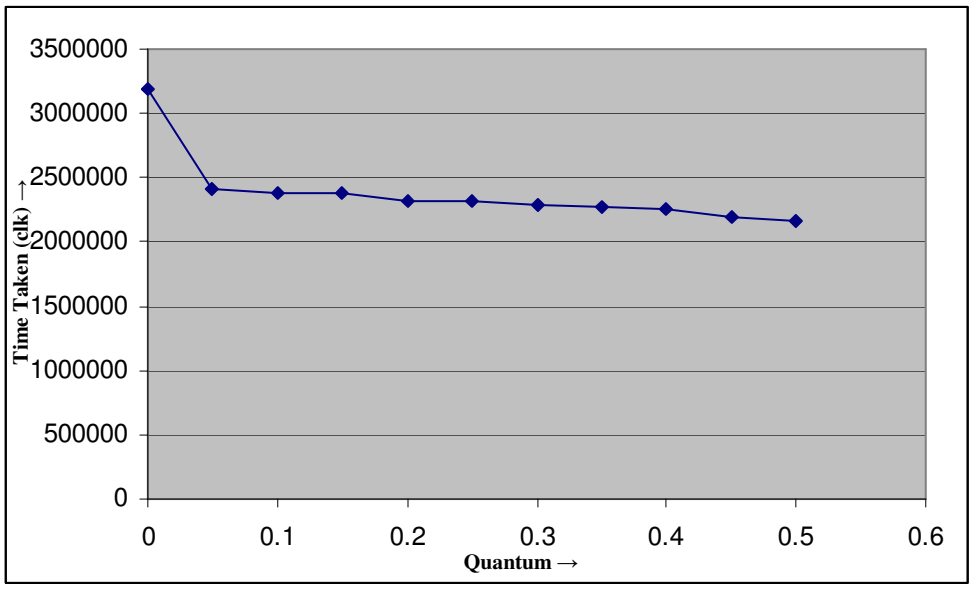


Figure 6.9 SPEED IMPROVEMENT – RADIX – 2 – SINE INPUT – SAME QUANTUM

6.3.1.6 INPUT Sine wave f = 16000 Hz, fs = 44100Hz

N-POINT	Quantum	Time Taken (clock cycles)	$\sum D_s - D_{sq})^2/(N)$
1024	0	3186176	0
1024	0.05	2363168	0.019887
1024	0.1	2328554	0.036828
1024	0.15	2306576	0.051304
1024	0.2	2291534	0.058094
1024	0.25	2279636	0.063437
1024	0.3	2281280	0.068193
1024	0.35	2310722	0.064777
1024	0.4	2332568	0.082279
1024	0.45	2329244	0.104613
1024	0.5	2327588	0.125317

TABLE 6.9 SPEED IMPROVEMENT – RADIX – 2 – SINE INPUT – SAME QUANTUM

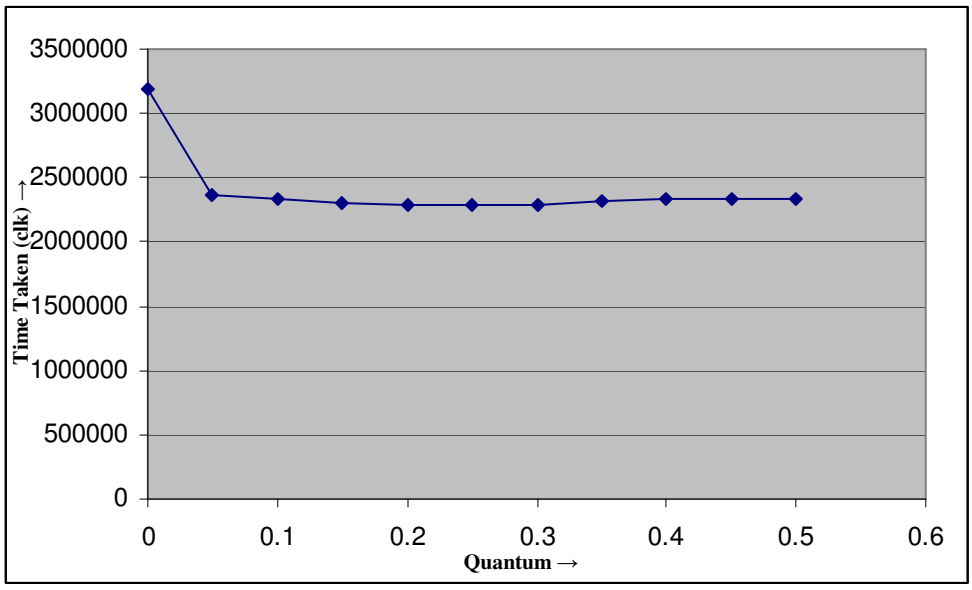


Figure 6.10 SPEED IMPROVEMENT – RADIX – 2 – SINE INPUT – SAME QUANTUM

6.3.1.7 MUSIC SIGNAL fs = 11025 Hz

The following experiment is conducted with a music signal whose spectrum is shown in the figure below. The maximum frequency in the input signal is limited to 5000Hz after passing it through a Low Pass Filter.

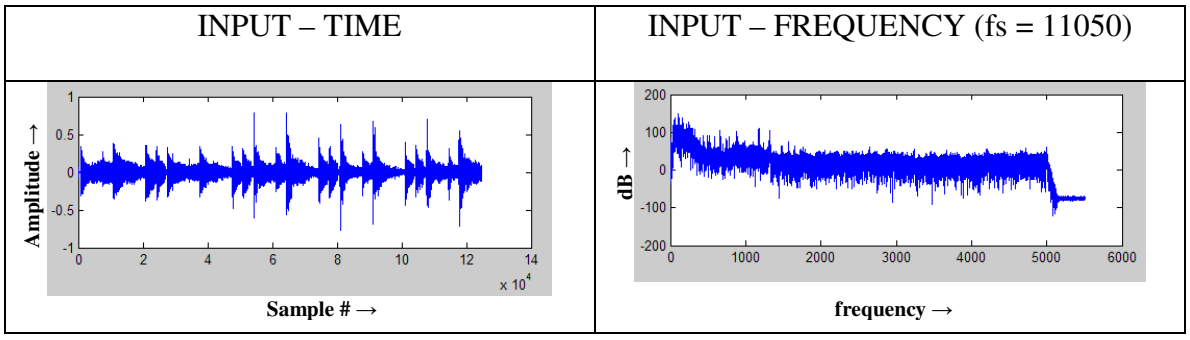


Figure 6.11 MUSIC SIGNAL

N-POINT	Quantum	Time Taken (clock cycles)	$\sum D_s - D_{sq})^2/(N)$
1024	0	9565696	0
1024	0.05	7680826	0.000661
1024	0.1	6716368	0.002599
1024	0.15	5612686	0.004885
1024	0.2	4793194	0.007438
1024	0.25	3929146	0.009523
1024	0.3	3355060	0.014715
1024	0.35	2920420	0.015464
1024	0.4	2276710	0.014773
1024	0.45	2130496	0.015426
1024	0.5	2024986	0.020765

TABLE 6.10 SPEED IMPROVEMENT – RADIX – 2 – MUSIC INPUT – SAME QUANTUM

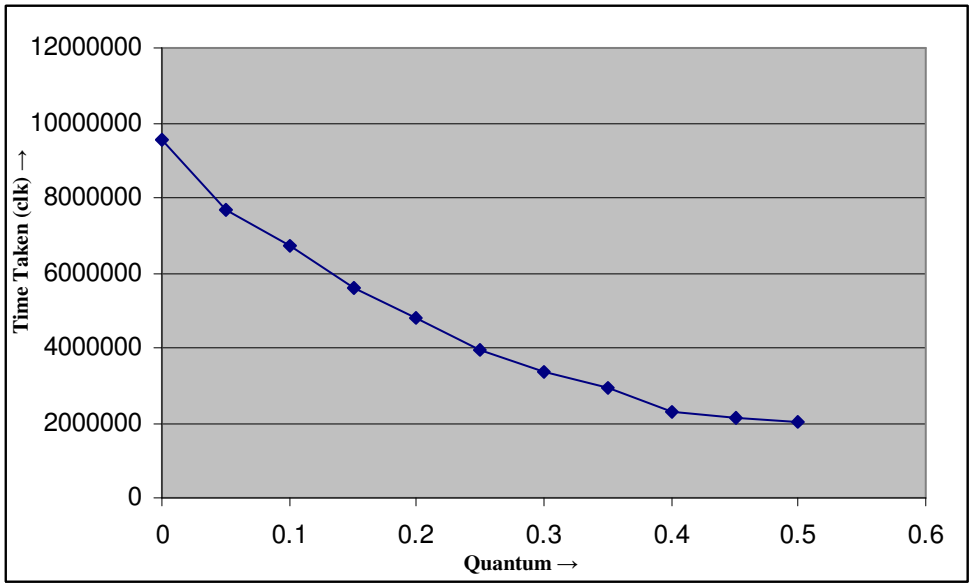


Figure 6.12 SPEED IMPROVEMENT – RADIX – 2 – MUSIC INPUT – SAME QUANTUM

6.3.1.8 MUSIC SIGNAL fs = 22050 Hz

The following experiment is conducted with a music signal whose spectrum is shown in the figure below. The maximum frequency in the input signal is limited to 10000Hz after passing it through a Low Pass Filter.

N-POINT	Quantum	Time Taken (clock cycles)	$\sum D_s - D_{sq})^2/(N)$
1024	0	18023936	0
1024	0.05	14331332	0.000842
1024	0.1	12034358	0.002959
1024	0.15	10022600	0.005347
1024	0.2	7941032	0.007916
1024	0.25	6623210	0.011146
1024	0.3	5596778	0.012665
1024	0.35	5163182	0.013522
1024	0.4	4451984	0.017959
1024	0.45	4146056	0.01849
1024	0.5	3510176	0.013889

TABLE 6.11 SPEED IMPROVEMENT – RADIX – 2 – MUSIC INPUT – SAME QUANTUM

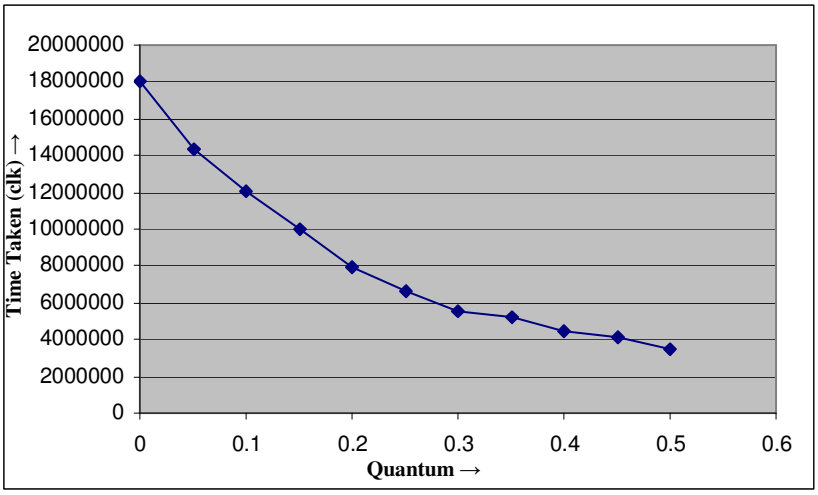


Figure 6.13 SPEED IMPROVEMENT – RADIX – 2 – MUSIC INPUT – SAME QUANTUM

6.3.1.9 MUSIC SIGNAL fs = 44100Hz

The following experiment is conducted with a music signal whose spectrum is shown in the figure below. The maximum frequency in the input signal is limited to 20000Hz after passing it through a Low Pass Filter.

N-POINT	Quantum	Time Taken (clock cycles)	$\sum D_s - D_{sq})^2/(N)$
1024	0	35728896	0
1024	0.05	27809478	0.001322
1024	0.1	22979556	0.003733
1024	0.15	18048522	0.005643
1024	0.2	14112540	0.008289
1024	0.25	12136536	0.010167
1024	0.3	10431780	0.01261
1024	0.35	9358932	0.014871
1024	0.4	8680242	0.022046
1024	0.45	7784202	0.024811
1024	0.5	7028190	0.022555

TABLE 6.12 SPEED IMPROVEMENT – RADIX – 2 – MUSIC INPUT – SAME QUANTUM

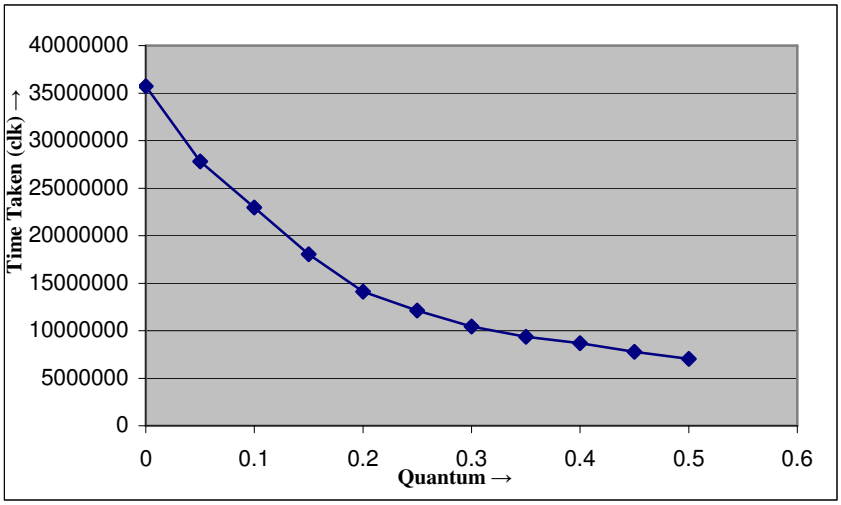


Figure 6.14 SPEED IMPROVEMENT – RADIX – 2 – MUSIC INPUT – SAME QUANTUM

6.3.1.10 MUSIC SIGNAL fs = 11025Hz

The following experiment is conducted with a progressive quantization scheme where the quantum varies as Quantum[i] = (q *$\log_2$N)/(i + 1)

N-POINT	Quantum	Time Taken (clock cycles)	$\sum D_s - D_{sq})^2/(N)$
1024	0	9565696	0
1024	0.005	8744014	0.000002
1024	0.01	8517760	0.000015
1024	0.015	8280820	0.000048
1024	0.02	8079052	0.000108
1024	0.025	7882684	0.000193
1024	0.03	7685512	0.000302
1024	0.035	7462780	0.000437
1024	0.04	7327378	0.000603
1024	0.045	7174150	0.000782
1024	0	9565696	0

TABLE 6.13 SPEED IMPROVEMENT – RADIX – 2 – MUSIC INPUT – PROGRESSIVE QUANTUM

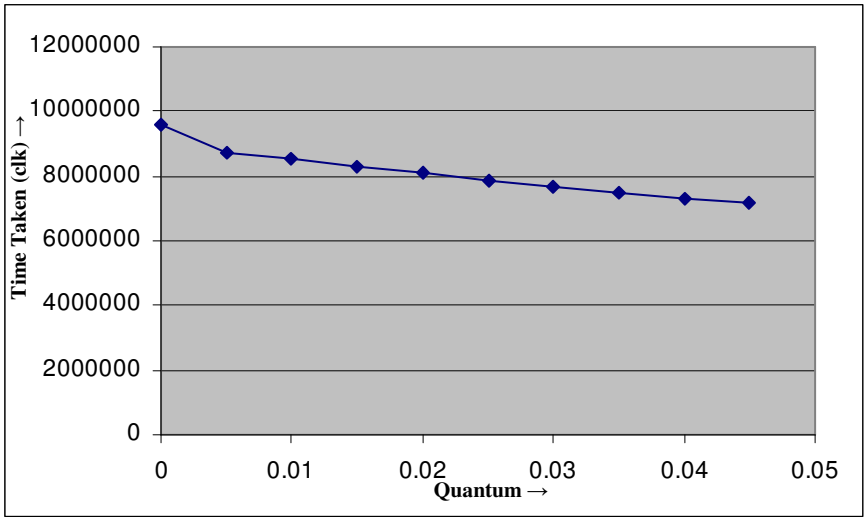


Figure 6.15 SPEED IMPROVEMENT – RADIX – 2 – MUSIC INPUT – PROGRESSIVE QUANTUM

6.3.1.11 MUSIC SIGNAL fs = 22050Hz

N-POINT	Quantum	Time Taken (clock cycles)	$\sum(D_s - D_{sq})^2/(N)$
1024	0	18023936	0
1024	0.005	16869896	0.000003
1024	0.01	16279688	0.000023
1024	0.015	15733088	0.000072
1024	0.02	15229154	0.000153
1024	0.025	14800058	0.000267
1024	0.03	14323640	0.000416
1024	0.035	13863350	0.000598
1024	0.04	13511174	0.000817
1024	0.045	13210040	0.00101

TABLE 6.14 SPEED IMPROVEMENT – RADIX – 2 – MUSIC INPUT – PROGRESSIVE QUANTUM

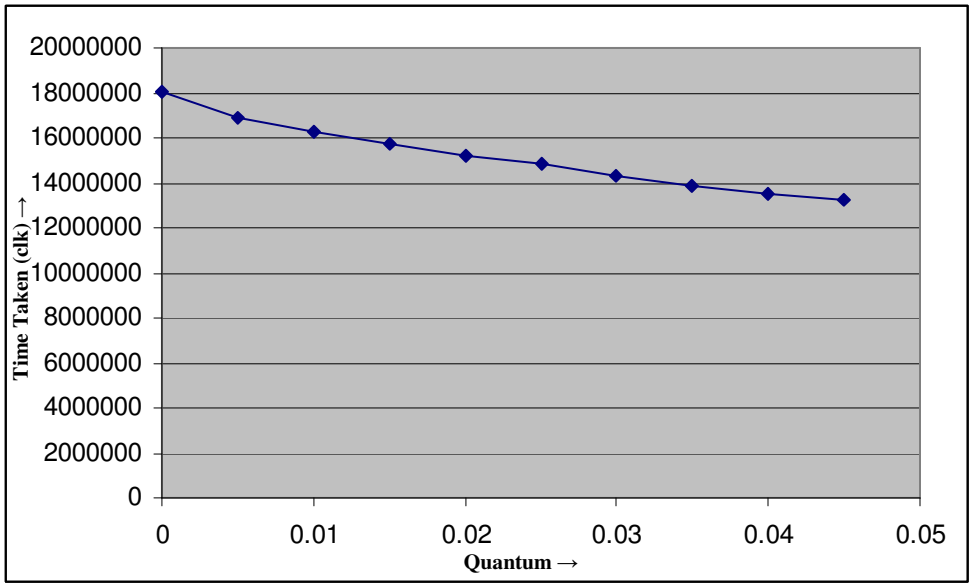


Figure 6.16 SPEED IMPROVEMENT – RADIX – 2 – MUSIC INPUT – PROGRESSIVE QUANTUM

6.3.1.12 MUSIC SIGNAL fs = 44100Hz

N-POINT	Quantum	Time Taken (clock cycles)	$\sum D_s - D_{sq})^2/(N)$
1024	0	35728896	0
1024	0.005	33324222	0.000006
1024	0.01	31727166	0.000044
1024	0.015	30439896	0.000129
1024	0.02	29318196	0.000271
1024	0.025	28364682	0.000457
1024	0.03	27470478	0.000689
1024	0.035	26652288	0.000958
1024	0.04	25849494	0.001225
1024	0.045	25112094	0.001501
1024	0	35728896	0

TABLE 6.15 SPEED IMPROVEMENT – RADIX – 2 – MUSIC INPUT – PROGRESSIVE QUANTUM

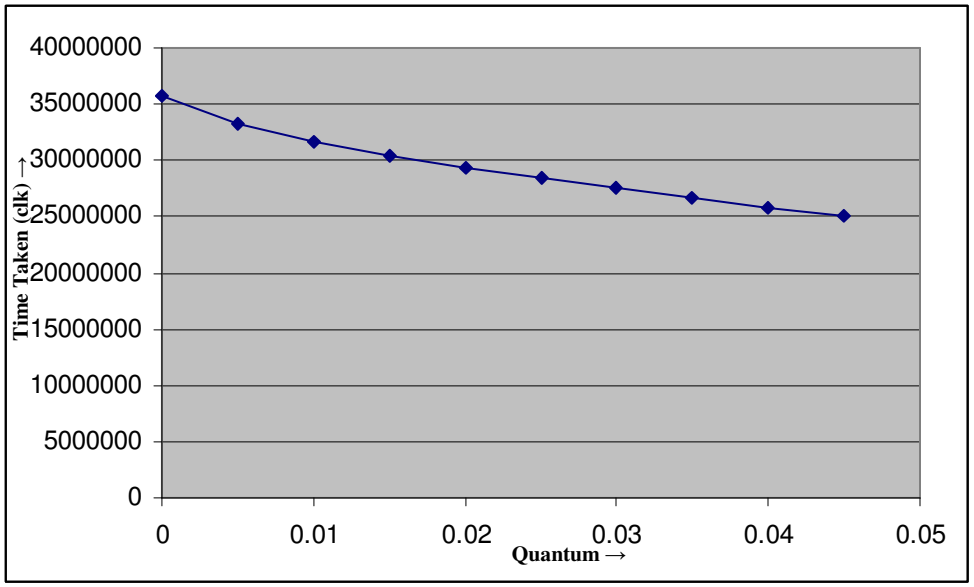


Figure 6.17 SPEED IMPROVEMENT – RADIX – 2 – MUSIC INPUT – PROGRESSIVE QUANTUM

6.3.1.13 MUSIC SIGNAL fs = 44100Hz

The following experiment shows the real-time speed improvement observed while running the simulations. Experiment 6.3.1.12 is repeated to obtain the real-time execution speed.

N-POINT	Quantum	Time Taken (seconds)	$\Sigma(D_s - D_{sq})^2/(N)$
1024	0	1.559313	0
1024	0.005	1.558073	0.000006
1024	0.01	1.462919	0.000044
1024	0.015	1.428628	0.000129
1024	0.02	1.39524	0.000271
1024	0.025	1.370745	0.000457
1024	0.03	1.347872	0.000689
1024	0.035	1.31875	0.000958
1024	0.04	1.297712	0.001225
1024	0.045	1.282039	0.001501

TABLE 6.16 SPEED IMPROVEMENT – RADIX – 2 – MUSIC INPUT – PROGRESSIVE QUANTUM

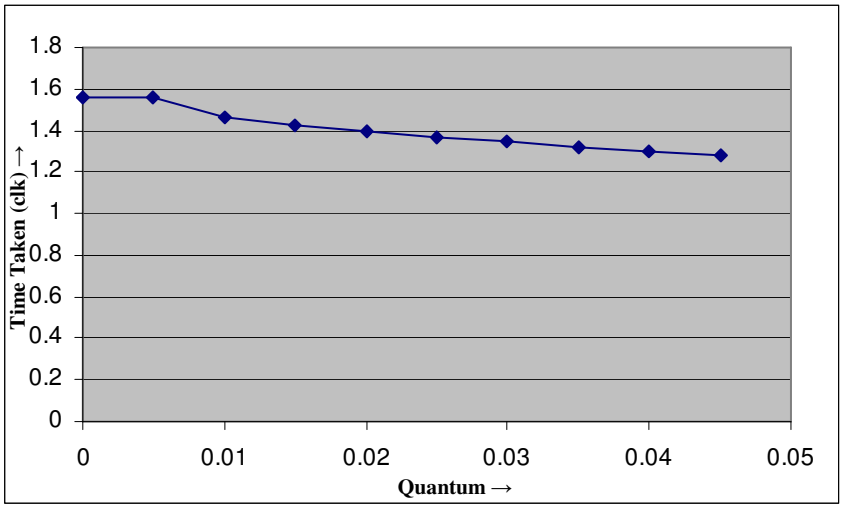


Figure 6.18 SPEED IMPROVEMENT – RADIX – 2 – MUSIC INPUT – PROGRESSIVE QUANTUM

6.3.2 RADIX – 4

The following experiments are conducted with Radix – 4 FFT for different values of computational quantization.

6.3.2.1 MUSIC INPUT fs = 44100Hz

In the following experiment, same quantum has been applied to all the stages of the transform.

N-POINT	Quantum	Time Taken (clock cycles)	$\sum D_s - D_{sq})^2/(N)$
1024	0	31577600	0
1024	0.05	25901072	0.000888
1024	0.1	22484772	0.002932
1024	0.15	18829248	0.005883
1024	0.2	15412843	0.00753
1024	0.25	13589773	0.008654
1024	0.3	11624130	0.011853
1024	0.35	11052455	0.012841
1024	0.4	10209779	0.014848
1024	0.45	10061919	0.026239

TABLE 6.17 SPEED IMPROVEMENT – RADIX – 4 – MUSIC INPUT – SAME QUANTUM

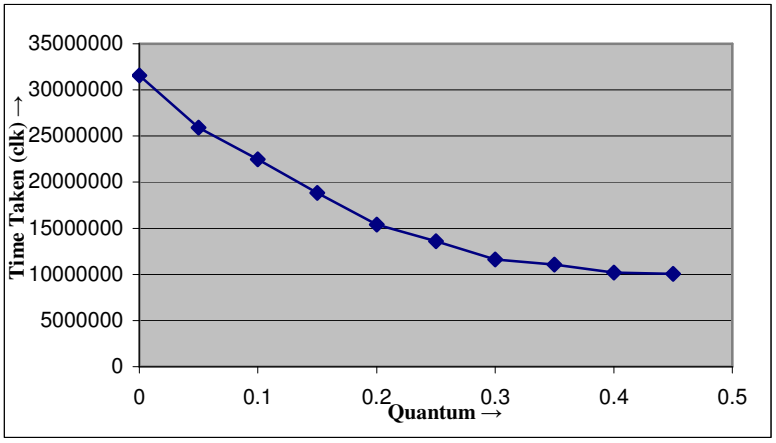


Figure 6.19 SPEED IMPROVEMENT – RADIX – 4 – MUSIC INPUT – SAME QUANTUM

6.3.2.2 MUSIC INPUT fs = 44100Hz

In the following experiment, progressive quantum has been applied to the stages of the transform.

N-POINT	Quantum	Time Taken (clock cycles)	$\sum(D_s - D_{sq})^2/(N)$
1024	0	31577600	0
1024	0.005	30508428	0.000003
1024	0.01	29414420	0.000021
1024	0.015	28460188	0.000067
1024	0.02	27675681	0.000139
1024	0.025	26994645	0.000246
1024	0.03	26350972	0.000382
1024	0.035	25776725	0.000527
1024	0.04	25259840	0.000726
1024	0.045	24711914	0.000902

TABLE 6.18 SPEED IMPROVEMENT – RADIX – 4 – MUSIC INPUT – PROGRESSIVE QUANTUM

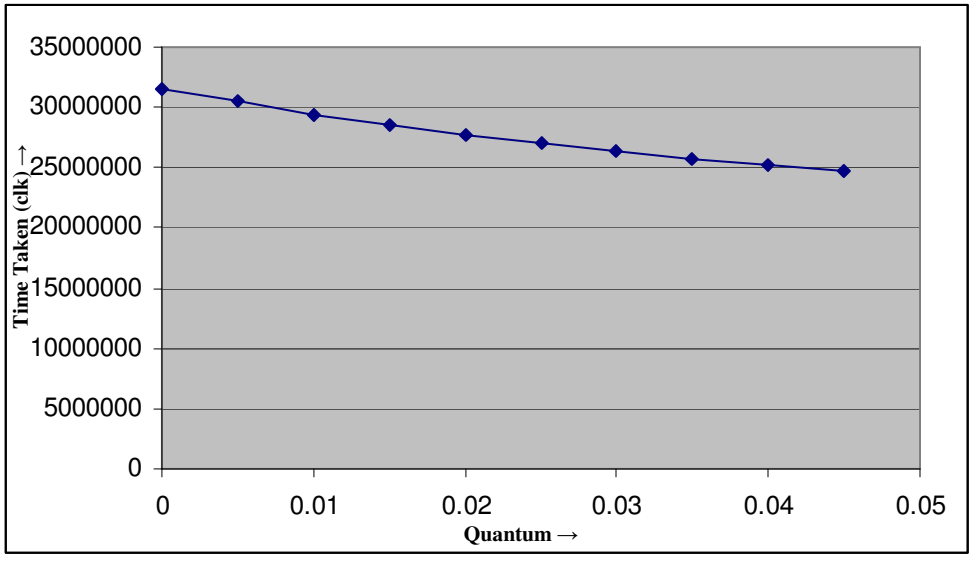


Figure 6.20 SPEED IMPROVEMENT – RADIX – 4 – MUSIC INPUT – PROGRESSIVE QUANTUM

6.3.3 MEAN SQUARE ERROR VARIATION WITH N

The following experiment is conducted to observe the errors for different values of N. So, a particular quantum is kept constant and N is varied for the observation.

N-POINT	Quantum	$\sum(D_s - D_{sq})^2/(N)$
256	0.02	0.000467
512	0.02	0.000385
1024	0.02	0.000271
2048	0.02	0.000182
4096	0.02	0.000124

TABLE 6.18 MEAN SQUARE ERROR VARIATION WITH N

N decides the block size of the transform. For a Radix – 2 transform there are $\log_2 N$ stages. For example, if a block size of 1024 is taken, there are 10 stages in the transform. For a block size of 2048 stages there are 11 stages in the transform. So, the error introduced by processing a bigger block size would be lesser because, the data undergoes a less number of passes. For a data size of 2048, a 1024 point transform has to do 20 passes, whereas a 2048 point transform does only 11 passes.

7 ANALYSIS

Experiments were conducted on various waveforms and different experiment setup and some of them have been shown above. The models work correctly and the simulations are fast & real-time and provide a solid framework for DSP modeling and simulation using Discrete Events. The Spectrum Analyzer model and Audio Player model are a great addition to the package which allow viewing the spectrum and hearing the output real-time during the simulation.

7.1 PIPELINING

The time taken for calculating the Radix – 2 FFT of a block is

$T_{NP} = (N/2)*\log_2 N*(2*TIME\_ADDITION + 1*TIME\_MULTIPLICATION)$

With zero quantum applied, the outputs are not computed if the current inputs are same as the previous inputs. So,

$T_{NP} = (N/2)*\log_2 N* (2*TIME\_ADDITION + 1*TIME\_MULTIPLICATION) +$
$(N/2)*\log_2 N *TIME\_COMPARE + N_{C2EQ} *TIME\_COMPARE –$
$N_{CEQ}*(2*TIME\_ADDITION + 1*TIME\_MULTIPLICATION)]$

where N_{C2EQ} is the Number of comparisons for the second input in the butterfly

$N_{C2EQ} <= (N/2)*\log_2 N$

where N_{CEQ} is the Number of comparisons that results in current input being equal to the previous input.

$N_{CEQ} <= N_{C2EQ}$

(there are a maximum of N_{C2EQ} butterflies that are not computed)

So, if T_C is taken as calculation time and T_{CMP} is taken as comparison time,

$T_{NP} = T_C [(N/2)*\log_2 N - N_{CEQ}] + T_{CMP}[(N/2)*\log_2 N - N_{C2EQ}]$

For pipelined implementation, the time taken is

$T_P >= T_{NP}/\log_2 N$

Because, all the stages wait to output till the stage which takes the longest time finishes processing. It is observed that all the values from Table 6.1 conform to these equations.

The same holds true for the Radix – 4 case where there are N/4 butterflies and $\log_4 N$ stages. Observations from Table 6.2 conforms to the equations modified for the Radix – 4.

In the Radix- 2 case there are $\log_2 N$ execution units and in Radix- 4 case there are $\log_4 N$ execution units. So, for the same block sizes, Radix – 2 would perform faster than Radix – 4 using the values from section 2.5.

7.2 COMPUTATIONAL QUANTIZATION

Computational quantization reduces the time taken for the quality trade-off. The Mean Square Error column tabulates the corresponding quality trade-offs with the speed improvement.

The observed speed improvements for the music signal from the tables are given below:

RADIX / QUANTUM TYPE	Fs	QUANTUM	TIME TAKEN (% = ratio of rime taken for the quantum to zero quantum)	MSE
2 – SAME	11025	0.2	50.11	0.007438
2 – SAME	22050	0.2	44.06	0.007916
2 – SAME	44100	0.2	39.50	0.008289
2 – PROG	11025	0.02	84.46	0.000108
2 – PROG	22050	0.02	84.49	0.000153
2 – PROG	44100	0.02	82.06	0.000271
4 – SAME	44100	0.2	48.81	0.00753
4 – PROG	44100	0.02	87.64	0.000139

TABLE 7.1 TIME TAKEN FOR VARIOUS QUANTUMS

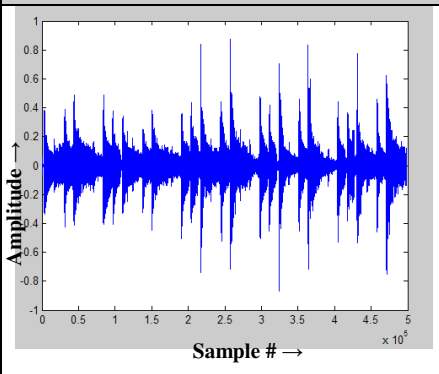
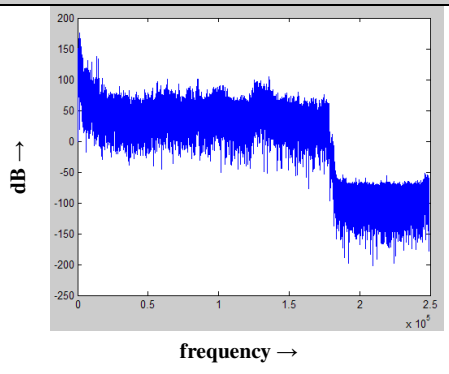
Using same quantum for all the stages of the transform produces better speed-up but the error is higher. Using progressive quantization reduces the Mean Square Error (MSE), but the time taken for the calculation is higher.

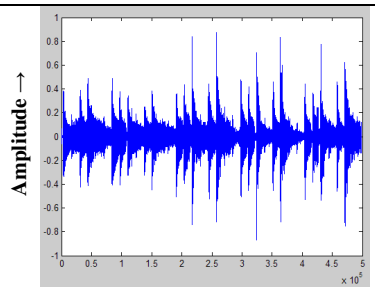
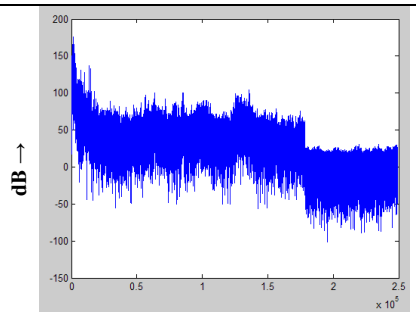
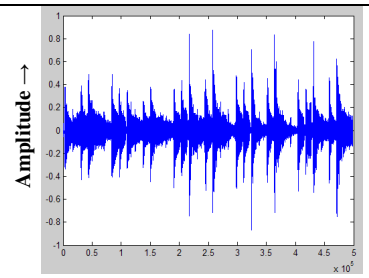
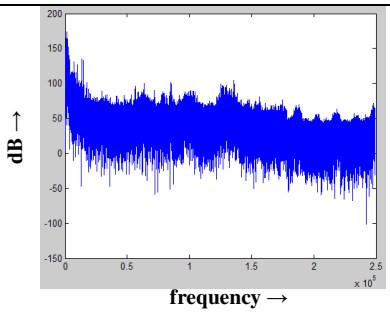
Increasing the sampling rate of the input signal, reduces the time taken for the computation, but correspondingly the error introduced is also higher. This is the same for same quantization or progressive quantization and also Radix – 2 and Radix – 4.

Radix – 4, for the same quantum and quantization type as Radix – 2, produces lesser speed-up and lesser error. This is because the number of comparisons also increase in Radix – 4.

The time taken for the computation is derived in Section 2.5. It shows that complex multiplication takes 4 clock cycles and complex addition and subtraction in 1 clock cycle each. The speed-up due to the computational quantization is analogous to saving power on systems that take up a lot of power performing floating point operations.

The following figures show the music input in time and frequency for various quantum:

Quantum (Progressive)	**Input(Time)**	**Input(Frequency)**
0		

0.005	 Amplitude → Sample # →	 dB → frequency →
0.02	 Amplitude → Sample # →	 dB → frequency →

8 CONCLUSION AND FUTURE WORK

DEVS serves as a good framework for modeling and simulating the system as software or hardware. In this thesis, we have shown that Discrete Event Simulation provides a solid framework for design and analysis of DSP based systems. Models for FFT (Butterfly, Stages, Controller and Quantizer) have been built for Radix – 2 and Radix – 4 transforms for Decimation in Frequency and Decimation Time Transforms. Audio Player, Spectrum Analyzer, Audio Generator (with Au format Decoder/Encoder) are the other models that have been built for the system. Models built using DEVS are fast, scalable, reusable and configurable. They allow real-time simulations of high sampling rate systems as shown in the thesis. The models built are of immense use for hierarchical systems based on them.

Various experiments were performed on the system and analyzed as shown in chapter 6 and chapter 7. Quantization when applied during the computation of FFT reduces the time taken for the computation. Table 7.1 shows the time taken for different quantization types and quantums. A quantum of 0.02 takes only 82% of the time taken for 0 quantum with a MSE of 0.000271 at a sampling rate of 44100 Hz. Noise due to quantization starts getting audible at this level.

Systems can be built in hardware where quantizers can be embedded with processing elements. In this case, even if a zero quantum is applied to the processing elements, the successive values, if equal, in case of constant amplitude signals, would benefit a lot from such a design. The future work is vast and promising. Quantizers can be built at hardware level and software systems can use them.

9 REFERENCES

1. James W. Cooley and John W. Tukey, *"An algorithm for the machine calculation of complex Fourier series",* Math. Comput. 19, 297–301 (1965).

2. Bernard P. Zeigler, *"Discrete Event Abstraction: An Emerging Paradigm For Modeling Complex Adaptive Systems",* to appear in "Adaptation and Evolution" (Festschrift for John H.Holland) to be published by Santa Fe Institute, Oxford Press, England.

3. Bernard P. Zeigler, *"Object Oriented Simulation with Hierarchical, Modular Models: Selected Chapters Updated for DEVS-C++"*, (1995) Originally published by Academic Press, 1990.

4. James J. Nutaro, *"Adevs: User manual and API documentation"*.

5. Bernard P. Zeigler, *"DEVS Today: Recent Advances in Discrete Event-Based Information Technology"*.

6. P. Duhamel and M. Vetterli, *"Fast Fourier transforms: a tutorial review and a state of the art"* Signal Processing 19, 259–299 (1990).

7. John G. Proakis and Dimitris Manolakis, *"Digital Signal Processing: Principles, Algorithms and Applications (3rd Edition)"*.

8. A. V. Oppenheim and R. W. Schafer, *"Discrete-Time Signal Processing"*.

9. Wan-Teh Chang, Soonhoi Ha, Edward A. Lee, *"Heterogeneous Simulation—Mixing Discrete-Event Models with Dataflow"*, Journal of VLSI Signal Processing 15, 127–144 (1997).

10. Duhamel, P. & Vetterli M. *"Fast Fourier Transforms: A Tutorial Review and a State of the Art,"* Digital Signal Processing Handbook Ed. Vijay K. Madisetti and Douglas B. Williams Boca Raton: CRC Press LLC, 1999.

11. *"AMD Athlon™ Processor x86 Code Optimization Guide"*, Publication No. 22007 Revision K Date February 2002.

12. *"AMD Extensions to the 3DNow! and MMX Instruction Sets Manual"*.

13. Lothar Thiele, *"Discrete Event Systems - Introduction - Computer Engineering and Networks Laboratory"*, Swiss Federal Institute of Technology (ETH) Zurich.

14. I.S.Uzun and A.Amira A.Bouridane, *"FPGA Implementations of Fast Fourier Transforms for Real-Time Signal and Image Processing"*, Field-Programmable Technology (FPT), 2003. Proceedings. 2003 IEEE International Conference on 15-17 Dec. 2003, p 102- 109.

15. Shousheng He and Mats Torkelson, *"A new approach to pipeline FFT processor"*, IPPS 1996.

16. Kofman, E. (2002), *"Discrete Event Simulation of Hybrid Systems"*, Technical Report LSD0205, LSD, Universidad Nacional de Rosario. To appear in SIAM Journal on Scientific Computing.

17. Zarka Cvetanovic, *"Performance analysis of the FFT-algorithm on a sharedmemory parallel architecture"*, IBM Journal of Research and Development(1987).

18. Chao-Kai Chang; Chung-Ping Hung; Sau-Gee Chen, *"An efficient memory-based FFT architecture, Circuits and Systems"*, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on Page(s): II-133- II-136 vol.2.

19. J.H. Son, B.S. Jo, B.G. Sunwoo, M.H. Oh, S.K. , *"A continuous flow mixed-radix FFT architecture with an in-place algorithm"*, Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on 25-28 May 2003 p II-133- II-136 vol.2.

20. Thomas Lenart and Viktor Öwall, *"A Pipelined FFT Processor using Data Scaling with Reduced Memory Requirements"*, In Proc. NORCHIP, 2002.

21. Joo, T.H.; Oppenheim, A.V., *"Effects of FFT coefficient quantization on sinusoidal signal detection"*, Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on , 11-14 April 1988 Pages:1818 - 1821 vol.3.

22. Sukhsawas, S.; Benkrid, K., *"A high-level implementation of a high performance pipeline FFT on Virtex-E FPGAs"*, VLSI, 2004. Proceedings. IEEE Computer society Annual Symposium on , 19-20 Feb. 2004 Pages:229 – 232.

23. Agarwal, R.C.; Gustavson, F.G.; Zubair, M., "A high performance parallel algorithm for 1-D FFT", Supercomputing '94. Proceedings , 14-18 Nov. 1994 Pages:34 – 40.

24. D'Abreu, M.C.; Wainer, G.A., *"Models for continuous and hybrid system simulation, Simulation Conference"*, 2003. Proceedings of the 2003 Winter , Volume: 1 , 7-10 Dec. 2003 Pages:641 - 649 Vol.1.

APPENDIX A CLASS DIAGRAMS

A.1 AU DECODER/ENCODER

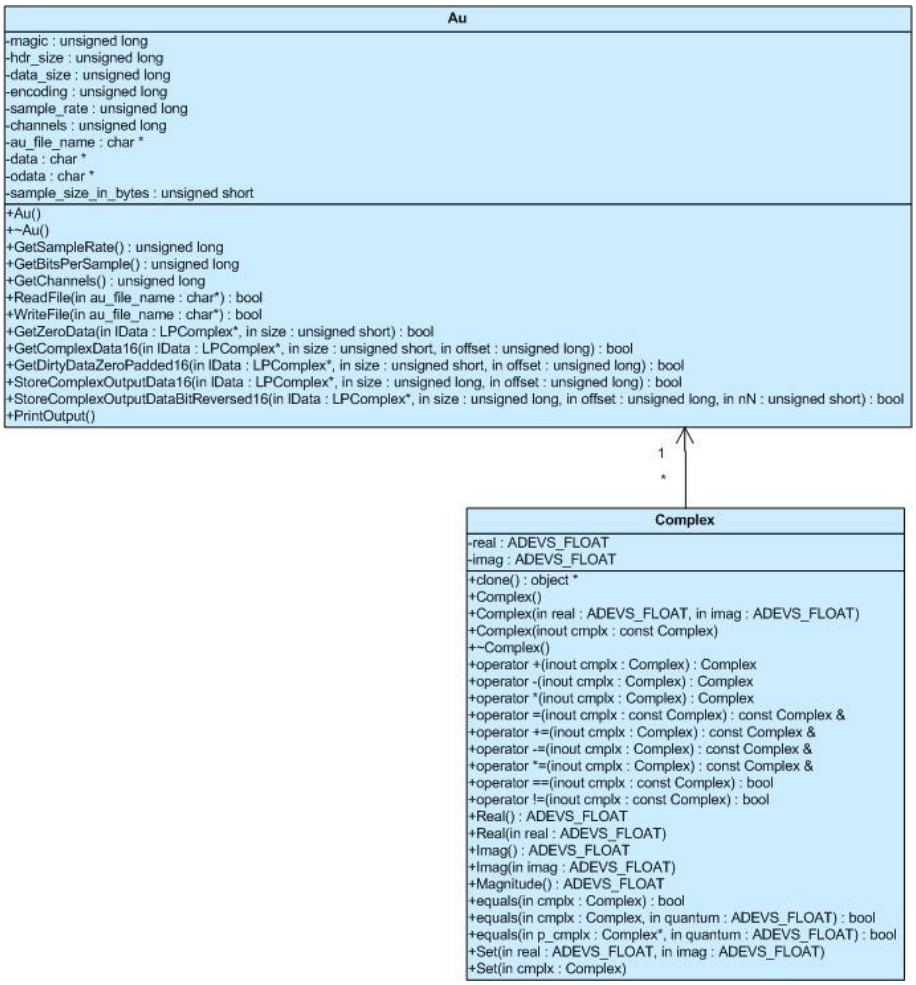


Figure A.1 – AU Decoder/Encoder Class Diagram

A.2 AUDIO PLAYER

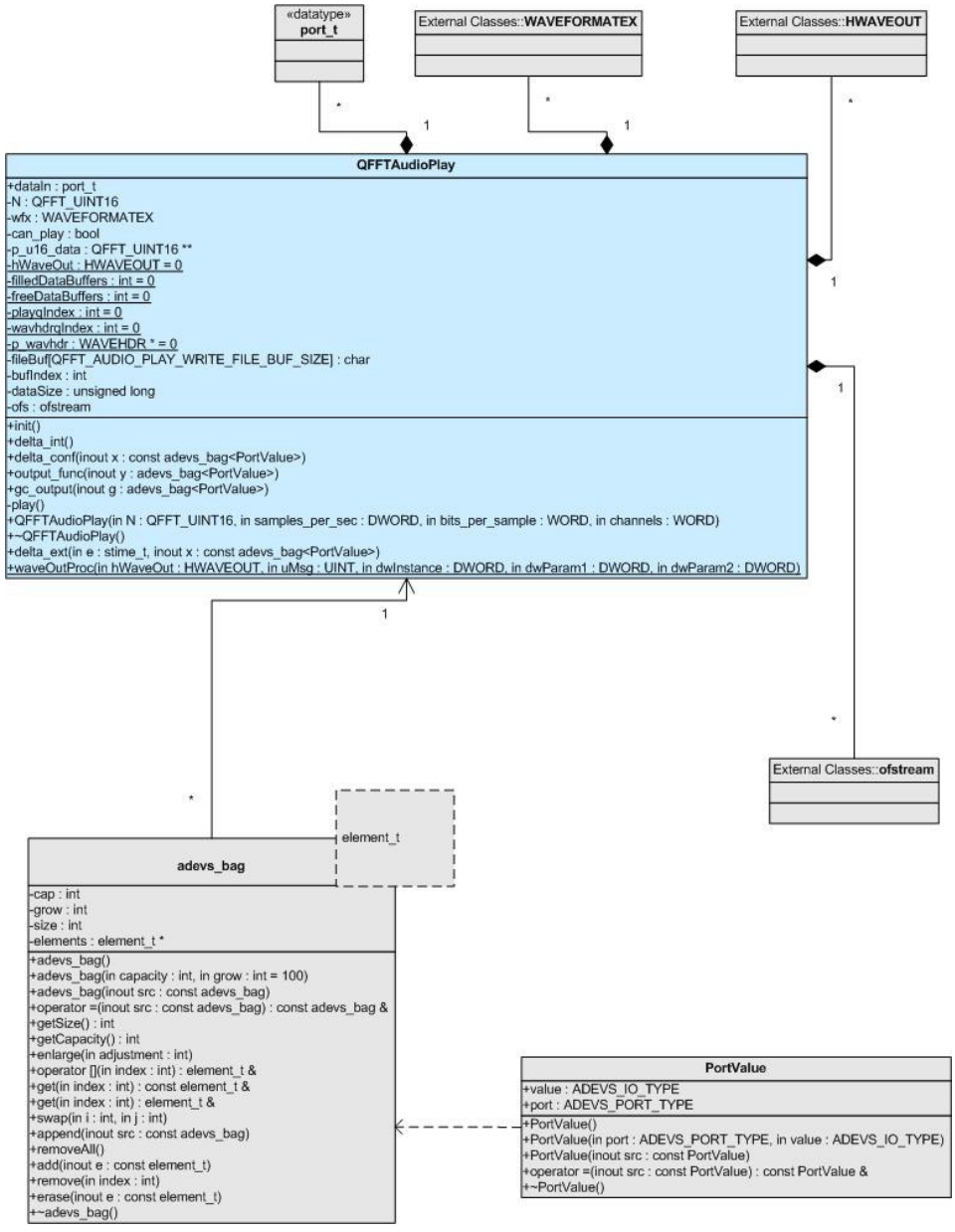


Figure – A.2 Audio Player Class Diagram – 1

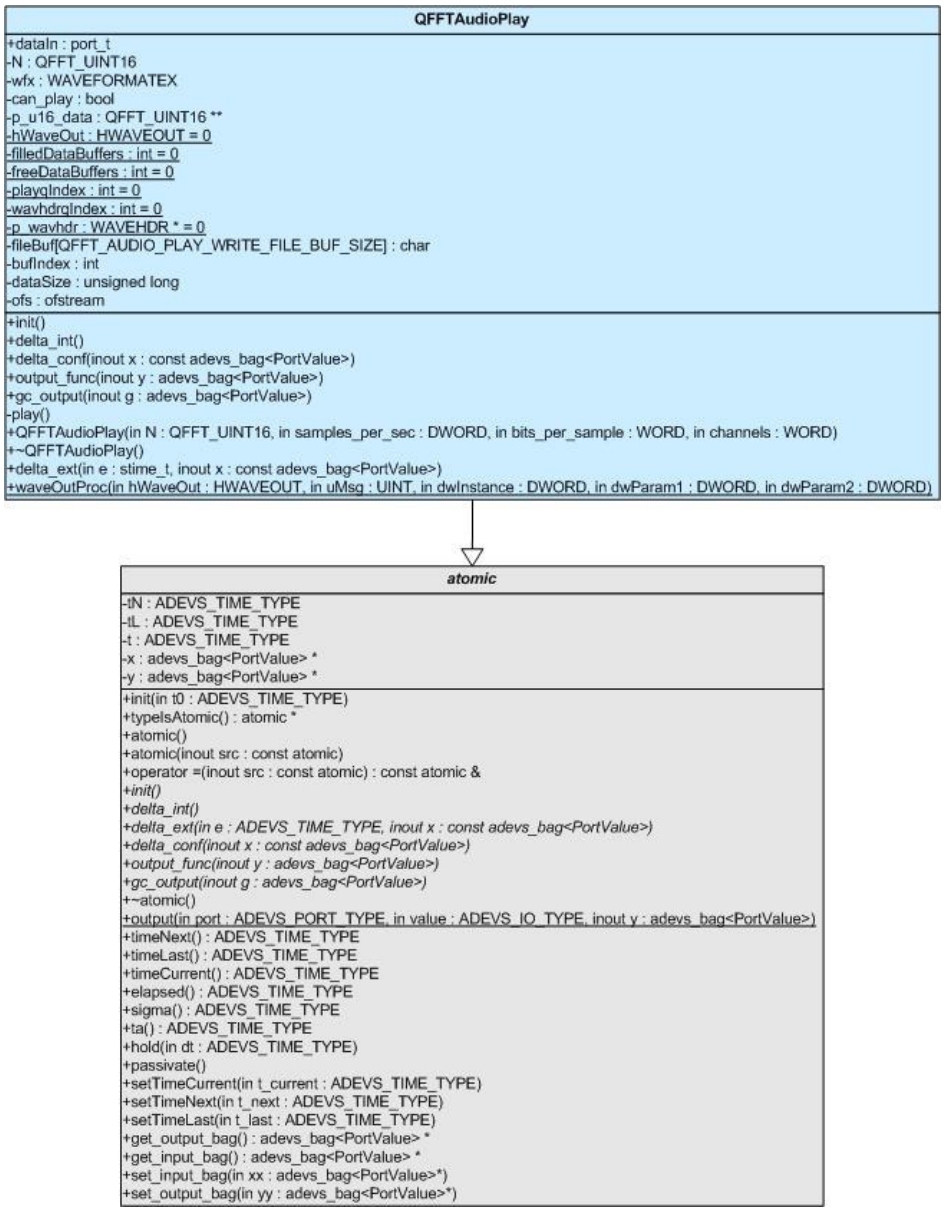


Figure – A.3 Audio Player Class Diagram – 2

A.3 BUTTERFLY

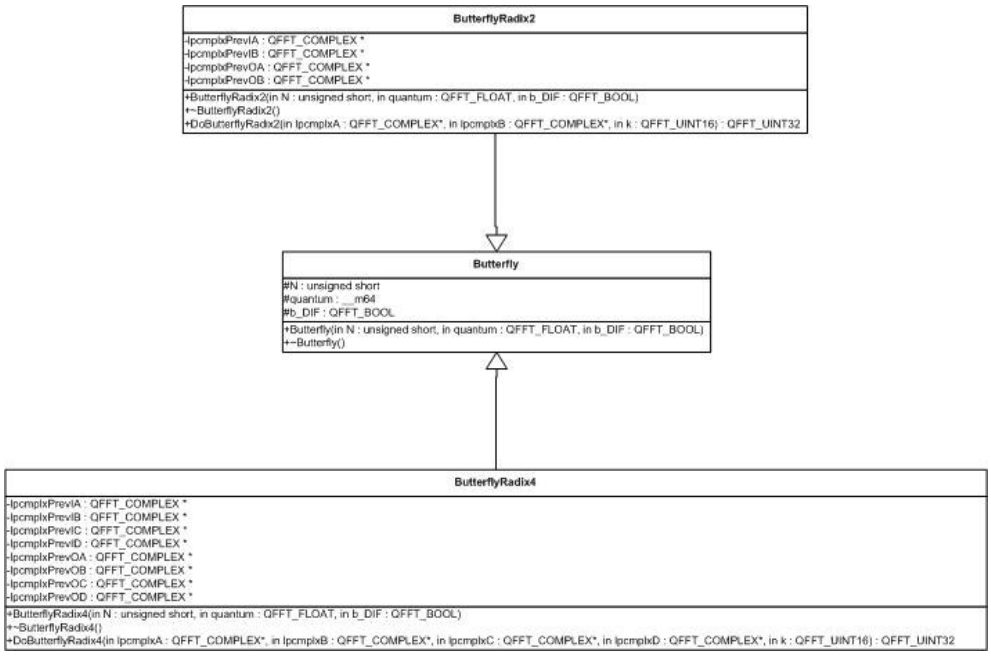


Figure – A.4 Butterfly

A.4 GENERATOR

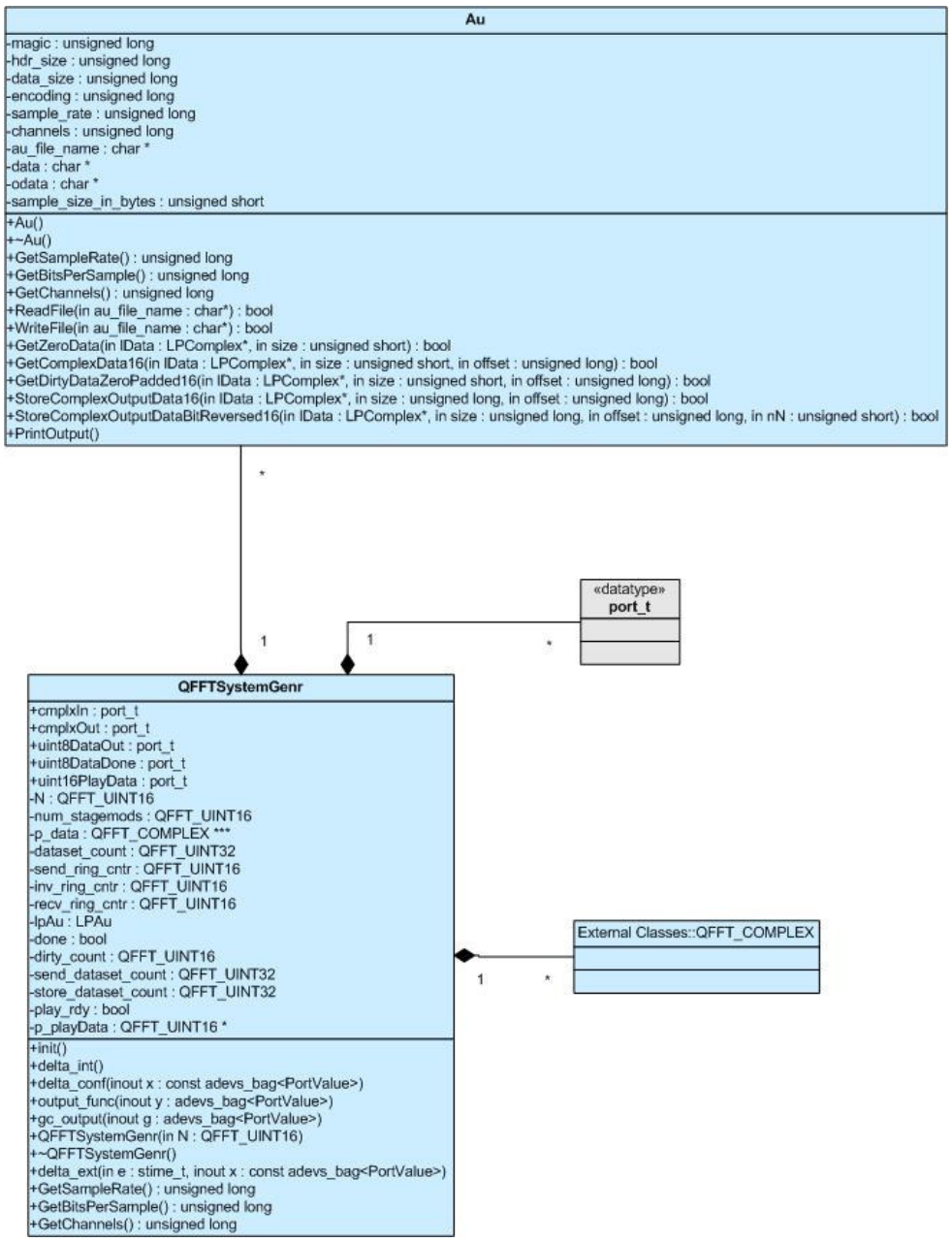


Figure – A.5 Generator Class Diagram – 1

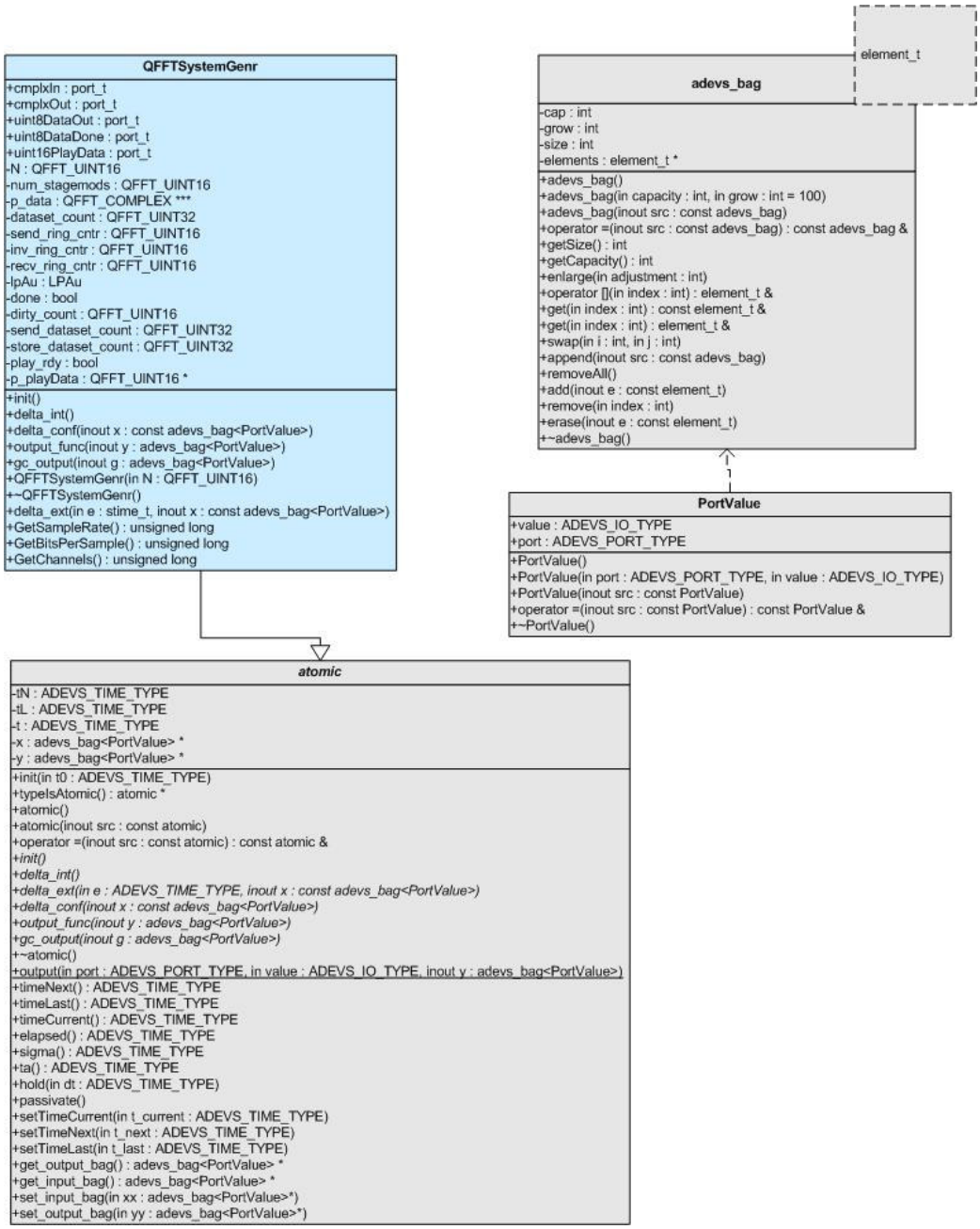


Figure – A.6 Generator Class Diagram – 2

A.5 MEDIAN FILTER

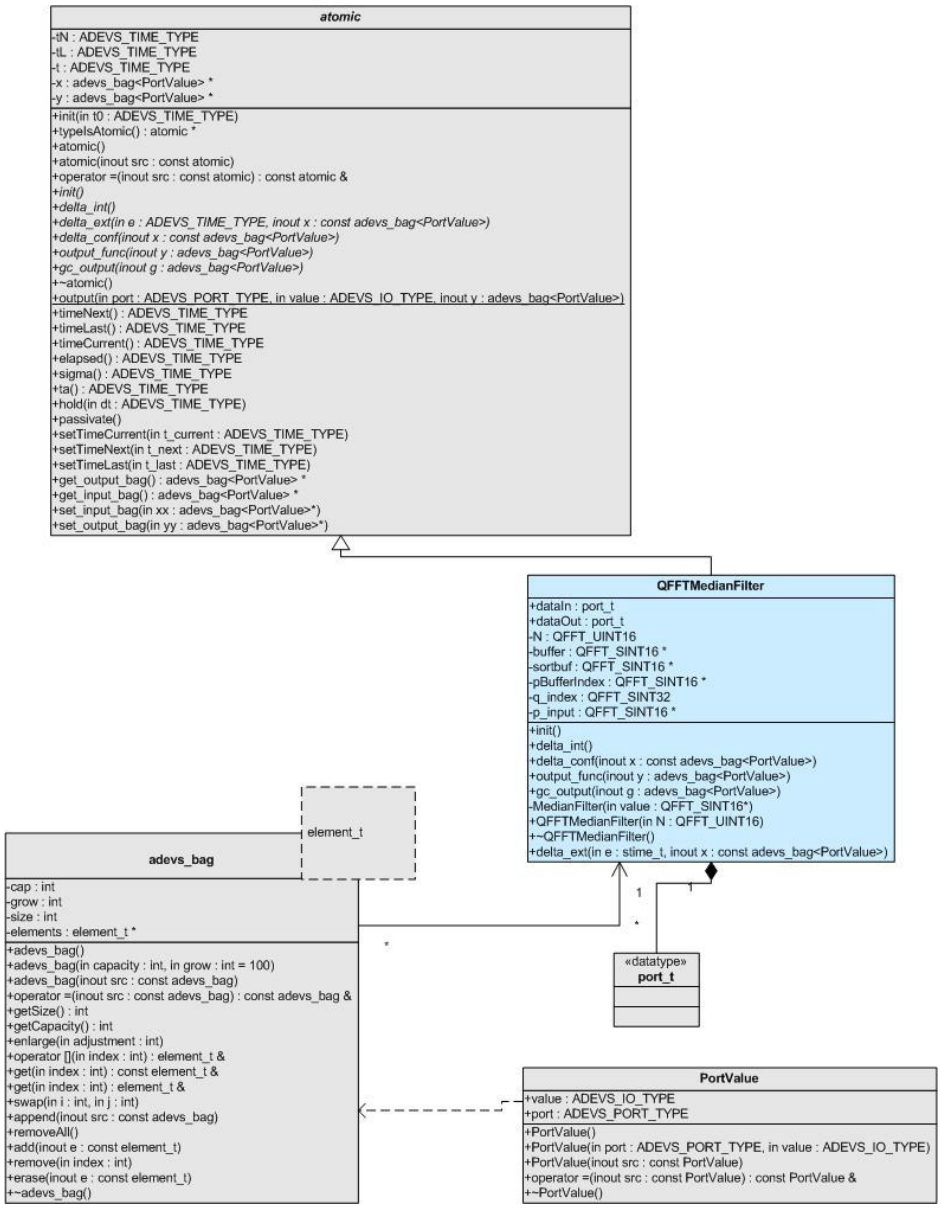


Figure – A.7 Median Filter Class Diagram

A.6 MODULE

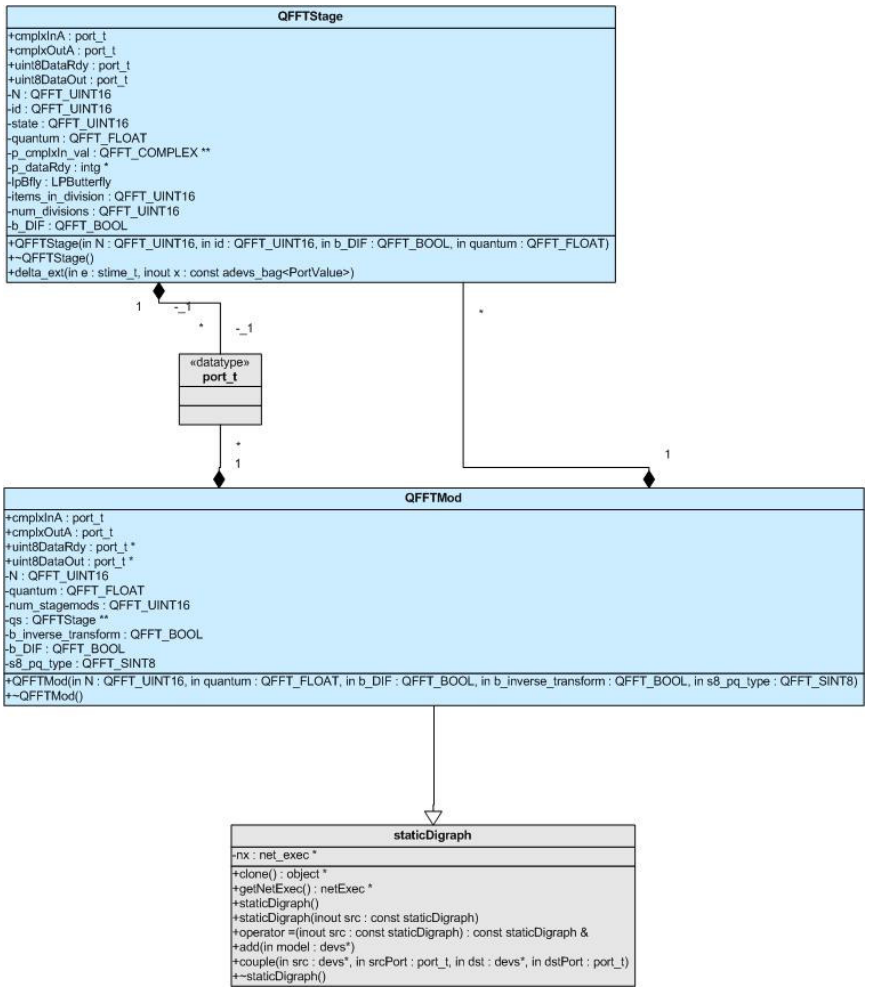


Figure – A.8 Module Class Diagram

A.7 SIMULATOR

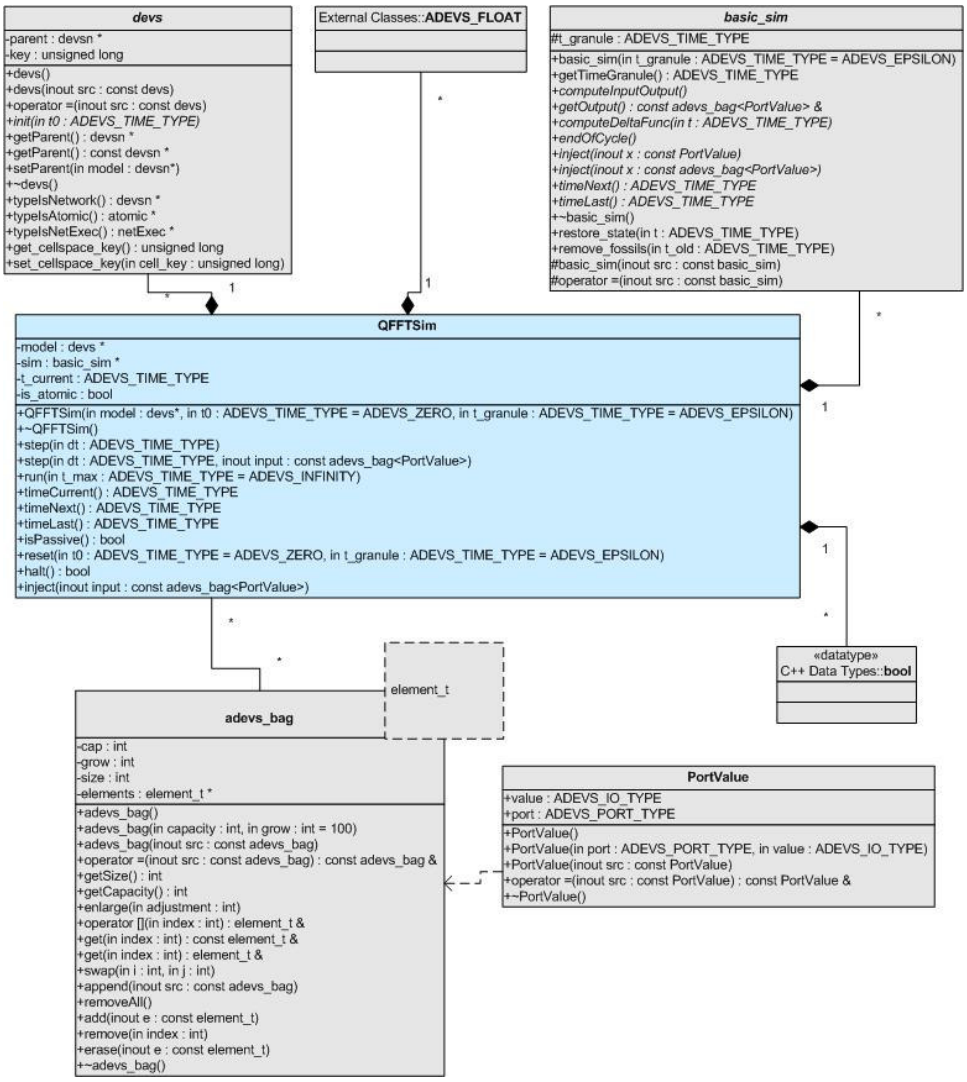


Figure – A.9 Simulator Class Diagram

A.8 SPECTRUM ANALYZER

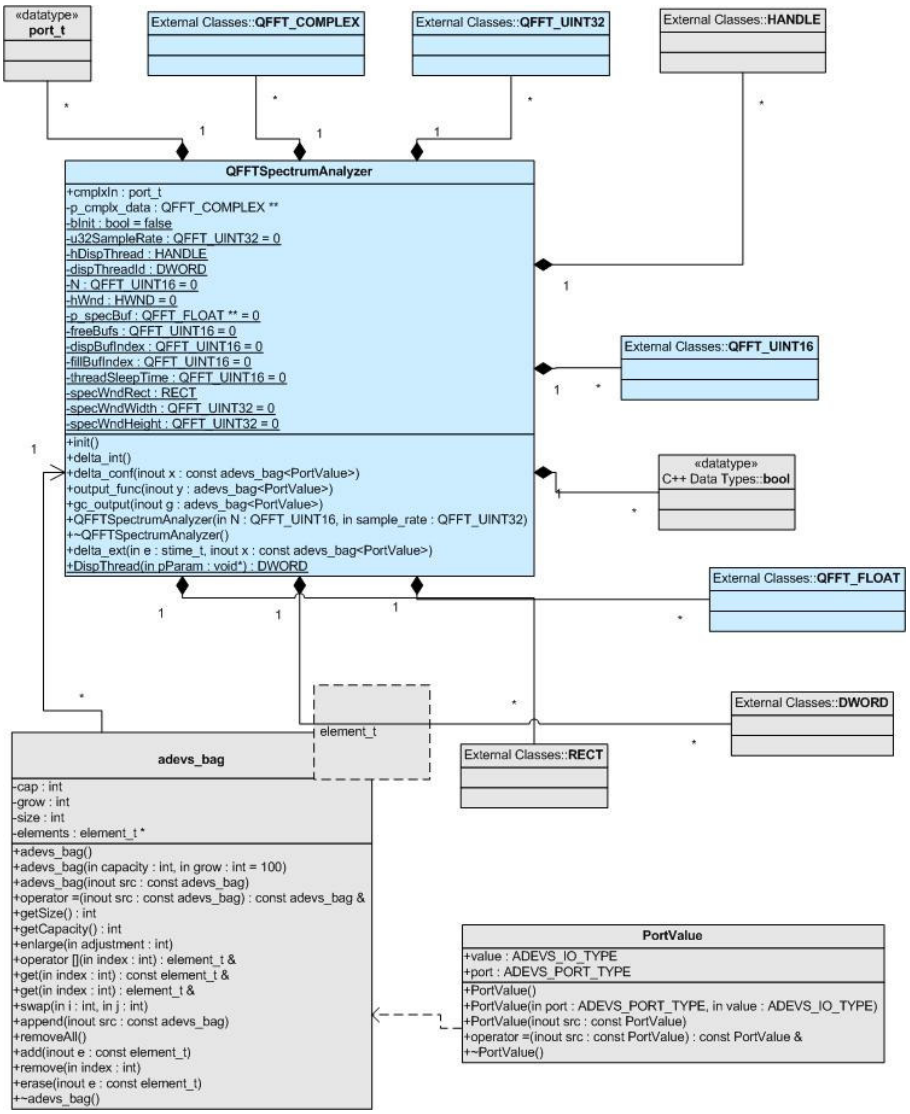


Figure – A.10 Spectrum Analyzer Class Diagram – 1

Figure – A.11 Spectrum Analyzer Class Diagram – 2

A.9 STAGE

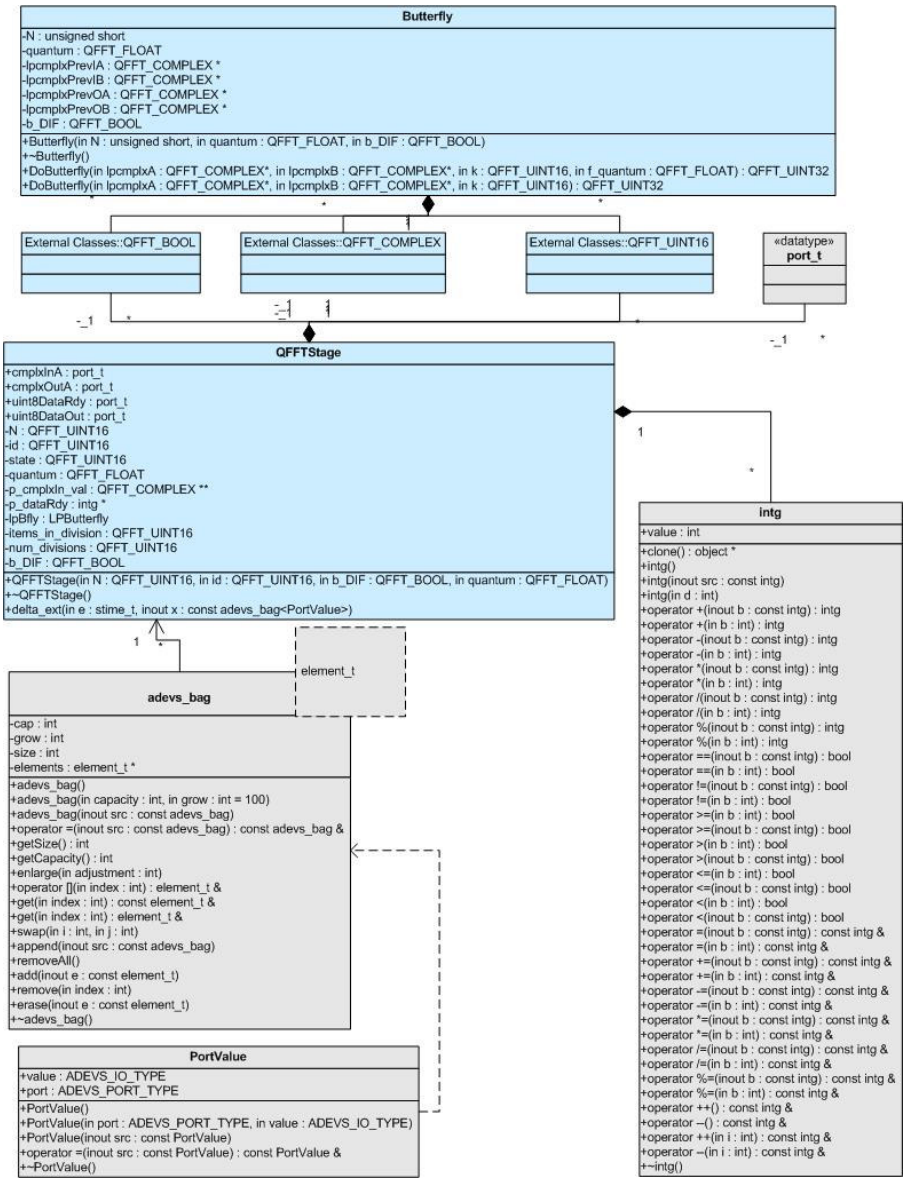


Figure – A.12 Stage Class Diagram – 1

Figure – A.13 Stage Class Diagram – 2

A.10 SYSTEM

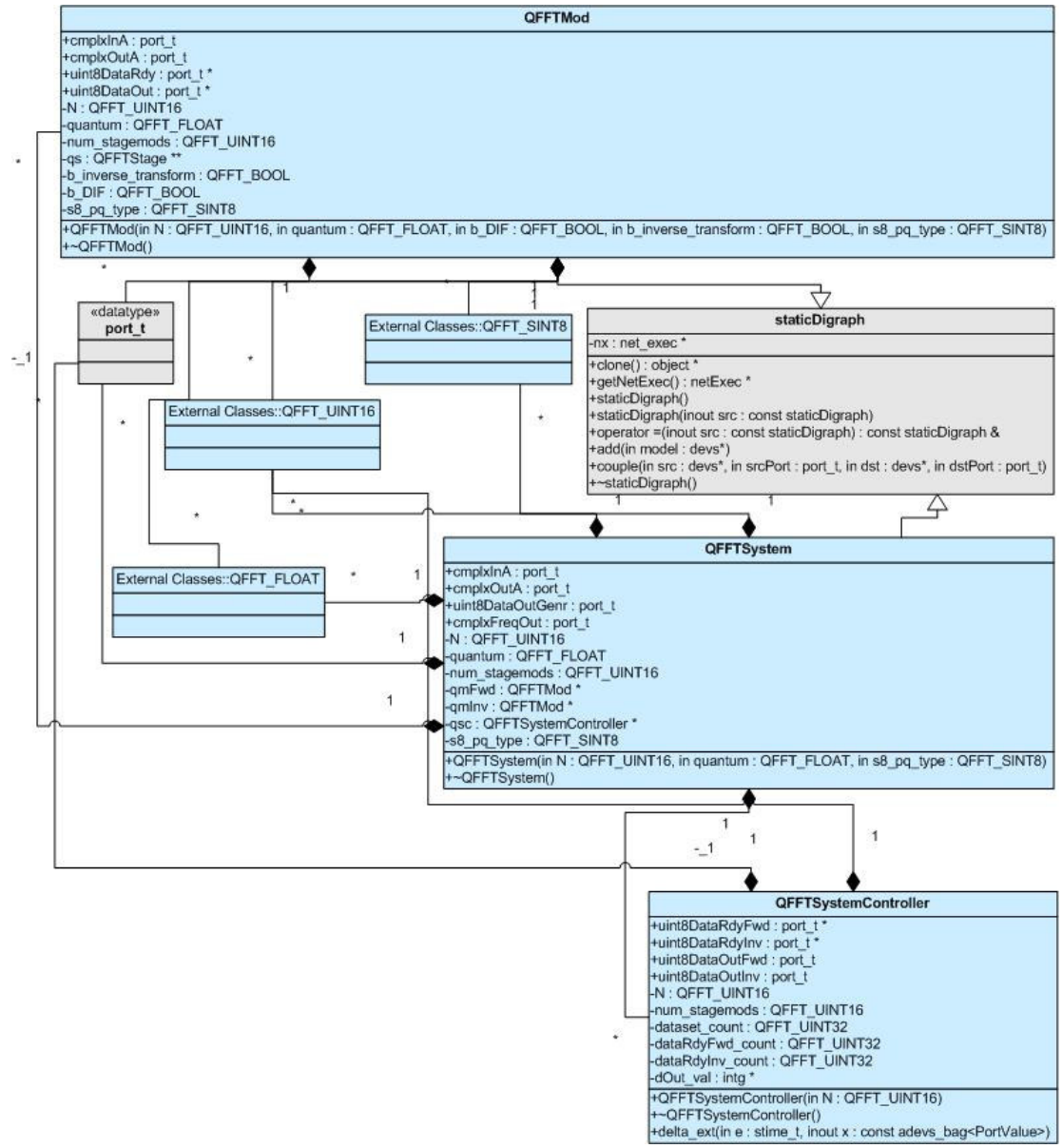


Figure – A.14 System Class Diagram

A.11 SYSTEM CONTROLLER

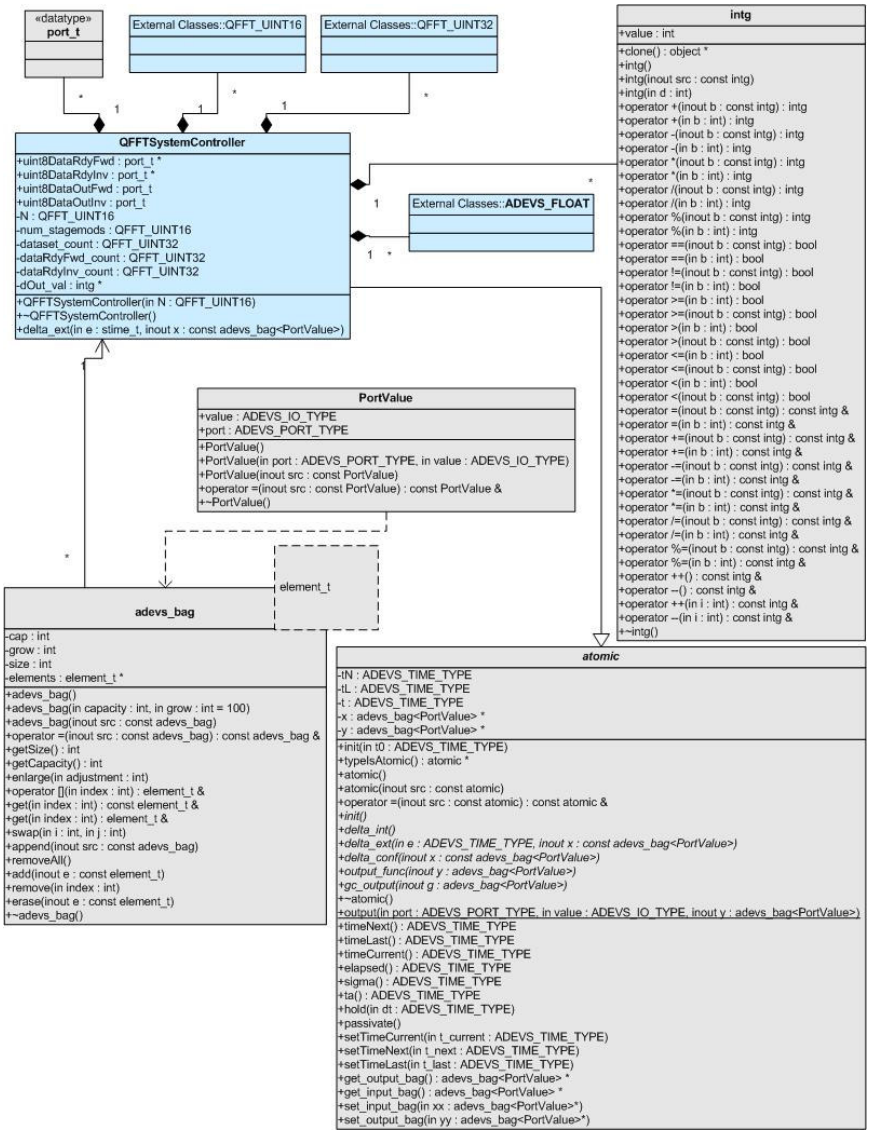


Figure – A.14 System Controller Class Diagram

APPENDIX B SIMULATION ENVIRONMENT

The ADEVS package and QFFT package were compiled on

1. Microsoft Visual Studio.NET 2003
    a. ADEVS as a library
    b. QFFT as a console application
    c. For the experiments both the projects were configured for release mode with Global Optimization, Full Optimization, Default Floating Point consistency, CPU for P4 and above and SSE2 code optimization.

The test machines were:

1. AMD Athlon XP 2800+/ 512MB RAM/ 64 MB ATI Radeon IGP.
2. Intel Pentium 4 2.8 GHz with HyperThreading/ 512 MB RAM / 128 MB ATI Radeon 9200.

APPENDIX C SIMD SUPPORT FOR SIMULATION

The following explains SIMD support for the simulation using AMD's 3DNow! instruction set and its extensions for DSP. Figure 2.2 shows a basic butterfly unit for Radix – 2 Decimation in Frequency FFT and Figure 2.4 shoes a basic butterfly for Radix – 2 Decimation in Time FFT. The complex mathematics involved for the operation can be simplified very much by using the SIMD instruction set and the DSP extensions. The instructions that simplify the calculation are:

INSTRUCTION	EXPLANATION
PSWAPD	Swaps upper and lower halves of the source operand
PFADD	Performs packed floating-point, single-precision addition
PFSUB	Performs packed floating-point, single-precision subtraction
PFMUL	Performs packed floating-point, single-precision Multiplication
PFPNACC	Performs packed floating-point, single-precision positive-negative accumulation

The following tables show the code for the butterflies and getting the magnitude of the complex number.

MOV	eax, DWORD PTR [lpcmplxA]
MOV	ebx, DWORD PTR [lpcmplxB]
MOVQ	MM0, [eax + 8]
MOVQ	MM1, MM0
PFADD	MM0, [ebx + 8]
PFSUB	MM1, [ebx + 8]
PSWAPD	MM2, MM1
PFMUL	MM1, MMWORD PTR t
PFMUL	MM2, MMWORD PTR t
PFPNACC	MM1, MM2
MOVQ	[eax + 8], MM0
MOVQ	[ebx + 8], MM1
FEMMS	

Table C.1 Radix-2 DIF Butterfly in AMD's 3DNow! Instruction set

MOV	eax, DWORD PTR [lpcmplxA]
MOV	ebx, DWORD PTR [lpcmplxB]
MOVQ	MM0, [ebx + 8]
PSWAPD	MM1, MM0
PFMUL	MM0, MMWORD PTR t
PFMUL	MM1, MMWORD PTR t
PFPNACC	MM0, MM1
MOVQ	MM1, MM0
MOVQ	MM2, [eax + 8]
PFADD	MM0, MM2
PFSUB	MM2, MM1
MOVQ	[eax + 8], MM0
MOVQ	[ebx + 8], MM2
FEMMS	

Table C.2 Radix-2 DIT Butterfly in AMD's 3DNow! Instruction set

MOV	eax, DWORD PTR value
MOVQ	MM0, MMWORD PTR [eax]
PFMUL	MM0, MM0
PFACC	MM0, MM0
PFRSQRT	MM1, MM0
PFRCP	MM0, MM1
MOV	eax, DWORD PTR magnitude
MOVQ	MMWORD PTR [eax], MM0
FEMMS	

Table C.3 Magnitude of a Complex Number using AMD's 3DNow! Instruction Set