

INFORMATICS & MATHEMATICAL MODELING,
TECHNICAL UNIVERSITY OF DENMARK,
LYNGBY, DENMARK



DESIGN OF AN INTEGRATED GFSK DEMODULATOR FOR A BLUETOOTH RECEIVER

PROJECT REPORT

BY: KASHIF MUNIR VIRK

SUPERVISER: DR. OLE OLSEN

ACKNOWLEDGEMENTS

I would like to acknowledge the technical advice received during this project from Dr. Søren Sennels at Nokia, Denmark and Mr. Ole Hoejrup at Xilinx, Denmark. The financial assistance received from the Centre for Integrated Electronics, DTU, Denmark through Dr. Ole Olesen and Nokia, Denmark through Dr. Dan Rebild during the course of this project is gratefully acknowledged. The cooperation of the staff at CIE, DTU and ASIC R&D, Nokia is highly appreciated. Finally, special thanks to Mr. Flemming Stassen, my counsellor and teacher for his help and advice during my M.Sc. degree.

SYNOPSIS

This report digresses the project work carried out for the design of an integrated GFSK demodulator for a receiver based on the Bluetooth™ specification. Starting from the system specification, the design description is presented as a set of hierarchical steps including block diagram models of the designed system. The specific functional details of each system block are introduced briefly. The models are refined and transformed to generate VHDL code for design prototyping which is then synthesized, implemented and tested on an FPGA. This report emphasizes the integrated system design methods employed using a Bluetooth™ receiver block as a design example.

CONTENTS

CHAPTER 1

Page 11

Introduction

- 1.1 Introduction
 - 1.2 Organization of the Report
 - 1.3 Project Background
-

CHAPTER 2

Page 15

System Design Methodology

- 2.1 Integrated System Design
 - 2.2 Selection of Design Automation Tools
 - 2.2.1 Workstation-based Tools
 - 2.2.2 PC-based Tools
-

CHAPTER 3

Page 21

A/D Conversion

- 3.1 A/D Converter Architectures
- 3.2 Flash A/D Converters
- 3.3 Oversampling A/D Converters

3.4 Considerations for the Selection of Sampling Frequency

CHAPTER 4

Page 25

System Algorithm Design

- 4.1 Spread Spectrum Modulation
 - 4.1.1 Spectrum Spreading Techniques
 - 4.1.2 Direct Sequence Spread Spectrum
 - 4.1.3 Frequency Hopping Spread Spectrum
 - 4.2 GFSK Modulation
 - 4.2.1 Frequency Shift Keying [FSK]
 - 4.2.2 Gaussian Frequency Shift Keying [GFSK]
 - 4.3 Demodulation Algorithms for GFSK
 - 4.3.1 Coherent Demodulation
 - 4.3.2 NonCoherent Demodulation
 - 4.3.3 Matched Filter-based Demodulation
 - 4.3.4 Frequency Discriminator-based Demodulation
 - 4.4 Detector Algorithms for GFSK
 - 4.5 Bluetooth™ Modulation Specification
-

CHAPTER 5

Page 34

System Architecture Design

- 5.1 Demodulator Architectures
 - 5.1.1 Correlator-based Demodulator
 - 5.1.2 Convolver-based Demodulator
- 5.2 Detector Architectures
 - 5.2.1 Envelope Detector
 - 5.2.2 Square Law Detector
- 5.3 System Architecture
- 5.4 Digital Filters
 - 5.4.1 IIR Filters
 - 5.4.2 FIR Filters
- 5.5 Digital Filter Design Flow
- 5.6 Digital Filter Design Algorithms
 - 5.6.1 Bilinear Transformation Method
 - 5.6.2 Impulse Invariant Method
 - 5.6.3 Pole-Zero Placement Method

- 5.6.4 Window Method
- 5.6.5 Optimal Method
- 5.6.6 Frequency Sampling Method

CHAPTER 6 Page 45**System Validation**

- 6.1 Floating-Point Model Validation
 - 6.1.1 Floating-Point Arithmetic
 - 6.1.2 Floating-Point System Model
- 6.2 Fixed-Point Model Validation
 - 6.2.1 Fixed-Point Arithmetic
 - 6.2.2 Fixed-Point System Model
 - 6.2.2.1 Filter Design Toolbox

CHAPTER 7 Page 63**System Realization**

- 7.1 Digital Filter Realization Structures
- 7.2 Finite Word Length/Quantization Effects
- 7.3 Filter Realization/Synthesis
 - 7.3.1 Filter Realization Wizard

CHAPTER 8 Page 73**FPGA Implementation**

- 8.1 FPGA-based Rapid Prototyping
- 8.2 VHDL Code Generation
 - 8.2.1 Xilinx™ Blockset-based System Model
- 8.3 Design Synthesis
- 8.4 Design Translation, Mapping, Placement & Routing

CHAPTER 9 Page 85**System Testing**

- 9.1 FPGA Programming
- 9.2 FPGA Testing Strategies

CHAPTER 10 Page 88**Summary & Conclusions**

- 10.1 Summary & Conclusions

- 10.1.1 Semi-Custom ASIC Design
 - 10.1.2 System Model Refinement
 - 10.1.3 Low-Power Design
-

APPENDIX A

Data Sheets of the Design Tools

APPENDIX B

System Models & Simulation Waveforms

APPENDIX C

Digital Filter Designs

APPENDIX D

Generated VHDL Code & Data Sheets of the Xilinx™ IP Cores

APPENDIX E

Xilinx™ FPGA Floorplan & Layout

APPENDIX F

Xilinx™ Virtex 1000E Data Sheets and Xilinx™ FPGA Test Board & Test Setup

APPENDIX G

References

FIGURES

- Figure 1.1 Bluetooth™ Reciever Block Diagram
- Figure 1.2 Bluetooth™ Receiver Architecture
- Figure 2.1 System Design Methodology
- Figure 2.2 The MATLAB™ CoDesign Environment for Hardware & Software
- Figure 2.3 The MATLAB™ Deisgn Environment for System & Hardware-level Design
- Figure 2.4 Impact of having a Common Design Environment on System Design Time
- Figure 3.1 A/D Converter Functions
- Figure 3.2 Common A/D Converter Architectures
- Figure 4.1 Direct Sequence Spread Spectrum Modulation
- Figure 4.2 The effect of Gaussian filter bandwidth on the signal frequency spectrum
- Figure 4.3 Classification of Modulation Formats
- Figure 5.1 Correlator-Based Demodulator
- Figure 5.2 Convolver-based Demodulator
- Figure 5.3 Decomposition of an FSK Signal into two ASK Signals
- Figure 5.4 Frequency Discriminator-based FSK Demodulator
- Figure 5.5 Block-level details of the Baseband Coprocessor
- Figure 5.6 Convolver-based Demodulator Architecture

- Figure 5.7 Digital Filter Block Diagram
Figure 5.8 Digital Matched Filter Block Diagram
Figure 5.9 Digital Filter Tolerance Scheme
Figure 5.10 Digital Filter Design Flow
Figure 6.1 Floating Point System Model
Figure 6.2 Settings of the quantization parameters for the Filter Design & Analysis Tool
Figure 6.3 Fixed-Point System Model
Figure 7.1 Direct Form I Filter Structure
Figure 7.2 A Linear Phase Structure for an FIR Filter with 7 Coefficients
Figure 7.3 Graphical User Interface of the Filter Realization Wizard
Figure 7.4 The Floating-Point Filter Block Synthesized by the Filter Realization Wizard
Figure 7.5 The Fixed-Point Filter Block converted from the Floating-Point Filter Block
Figure 7.6 Section of the Synthesized Fixed-Point Direct Form II Realization Structure
Figure 7.7 The Encapsulated Fixed-Point Delay Block (shown in red in Figure 7.6)
Figure 7.8 The Synthesized 49-Tap Direct Form II Bandpass FIR Filter Structure
Figure 7.9 Realized Fixed-Point System Model
Figure 8.1 The Xilinx™ System Generator & MATLAB™ Interface
Figure 8.2 Xilinx™ Blockset-based System Model
Figure 8.3 Design Synthesis Using Synopsys™ FPGA Express Synthesis Environment
Figure B.1 Output after the Bernoulli Random Binary Generator Block
Figure B.2 Output after the Sum Block
Figure B.3 Output after the Relational Operator Block
Figure B.4 Output after the Zero Order-Hold1 Block
Figure B.5 Output after the Remez FIR Filter Design Block
Figure B.6 Output after the -0.5 Constant & Sum Blocks
Figure B.7 Output after the AWGN Channel Block
Figure B.8 Output after the Digital FIR Filter Design1 Block
Figure B.9 Output after the Abs1 Block
Figure B.10 Output after the Digital FIR Filter Design3 Block
Figure B.11 Output after the Bernoulli Random Binary Generator Block
Figure B.12 Output after the FixPt Sum Block
Figure B.13 Output after the FixPt Relational Operator Block
Figure B.14 Output after the FixPt Gateway Out Block
Figure B.15 Output after the Bernoulli Random Binary Generator Block
Figure B.16 Output after the FixPt Sum Block
Figure B.17 Output after the FixPt Relational Operator Block

- Figure B.18 Output after the FixPt Gateway Out Block
Figure B.19 Output after the Bernoulli Random Binary Generator Block
Figure B.20 Output after the Adder2 Block
Figure B.21 Output after the Relational Block
Figure B.22 Output after the Gateway Out Block
Figure E.1 Custom-Built Xilinx™ FPGA Test Board
Figure E.2 Xilinx™ FPGA Test Setup

1

INTRODUCTION

1.1 INTRODUCTION

This report is the result of my M.Sc. research project that is the final part of my studies in Computer Systems Engineering at the Institute of Informatics & Mathematical Modeling [IMM] at the Danish Technical University [DTU]. This project was carried out at the Centre for Integrated Electronics [CIE], Oersted Institute, DTU and Nokia, ASIC R&D, Copenhagen, Denmark during the Spring of 2001. The duration of the project was limited to six months as a full-time work.

1.2 ORGANIZATION OF THE REPORT

This report requires the reader to have familiarity with the fundamental concepts of Communication Systems, Digital Signal Processing, Digital Systems, and ASIC Design & Testing. Familiarity with the structure and operation of Electronic System Design Automation (ESDA) tools can augment the comprehension of the subject matter as well. However, some of the theoretical foundations for grasping the design details have been discussed in chapters 3 to 5 of this report. An extensive list of references has been provided as Appendix-G for thorough understanding of the ideas discussed in the report. Citations to some of the references might not be found in the report but they are listed because they were used to gain deeper insights into the respective subjects.

This report is organized as follows:

Chapter 1 gives an overview of the Bluetooth™ receiver system and briefly describes how this project work fits into the overall task of the receiver system design.

Chapter 2 details the design methodology adopted for this project and the selection of the design automation tools to implement the adopted design methodology.

Chapter 3 discusses A/D converters and the issues and considerations for selecting the appropriate sampling rate for the designed system.

Chapter 4 elaborates the theoretical details of the modulation formats employed by the Bluetooth™ specification and the algorithm-level design of the GFSK demodulator.

Chapter 5 explains the architecture-level implementation of the demodulator and the concepts of digital filtering employed to design the demodulator architecture.

Chapter 6 is the key chapter of this report that puts together all the details of the previous chapters into a set of hierarchical system models for validating the performance of the designed system architecture and extracting the design parameters for the system blocks.

Chapter 7 discusses the system realization steps and tradeoffs.

Chapter 8 is also an important chapter of this report as it explains the process of rapid system prototyping by translating the refined system model into VHDL code and subsequent synthesis, translation, mapping, placement, and routing of the FPGA.

Chapter 9 describes the issues and strategies of system testing considered for this project.

Chapter 10 summarizes and concludes the report by discussing the final design and gives suggestions for further improvement.

1.3 PROJECT BACKGROUND

This project work is a part of the Confront project at the Centre for Integrated Electronics (CIE), Oersted Institute, Technical University of Denmark (DTU), that involves the Design of an integrated receiver based on the Bluetooth™ specification. The proposed block diagram of the Bluetooth™ receiver is shown below:

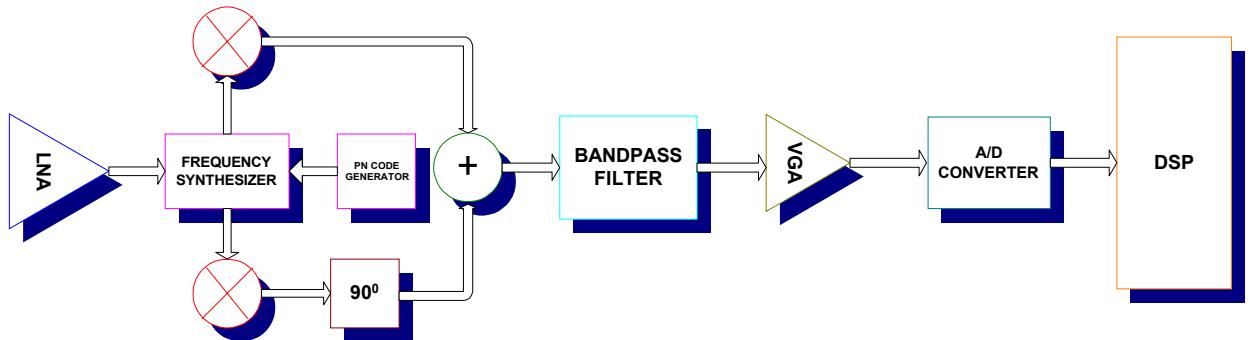


Figure 1.1 Bluetooth™ Receiver Block Diagram

The Bluetooth™ receiver mainly consists of the following circuit blocks:

RF Frontend — Low-IF [33]

- 1) Low Noise Amplifier (LNA)
- 2) Image-reject Mixers
- 3) PLL-based Frequency Synthesizer
- 4) 90° Phase Shifter
- 5) Band Pass Filter

- 6) Variable Gain Amplifier (VGA)

Baseband Backend

- 1) Analog-to Digital Converter (A/D)
- 2) Application-Specific Baseband Coprocessor / General-Purpose Control Processor

A brief functional description of each of the above circuit blocks is given below:

The RF signal received at the antenna is filtered through a SAW (Surface Acoustic Wave) filter and amplified by a broadband, low-noise amplifier (LNA). After the LNA, the amplified signal is split into in-phase (I-) and quadrature-phase (Q-) components and mixed with the frequencies generated by the PLL frequency synthesizer, acting as a *local oscillator*, in the *image-reject* mixers to down-convert it to an intermediate frequency (IF) of 1 MHz. The down-converted signal is passed through the VGA (Variable Gain Amplifier) to stabilize its (possibly) varying gain to a constant value so that it can be sampled by the A/D Converter(s) of relatively less *dynamic range*. The sampled signal is input to the Application-Specific Baseband Coprocessor where it is digitally demodulated and detected to recover the binary bit-valued data for further processing.

This project work involves the design of the Baseband Backend for the Bluetooth™ Receiver that includes the design specification of the A/D Converter and the Application-Specific Baseband Coprocessor and a detailed design investigation of the frequency hopping and GFSK Demodulation algorithms to be processed by the Application-Specific Baseband Co-processor.

SUMMARY

This chapter introduced the project and briefly described the receiver system based on the Bluetooth™ specification. To put the project work into perspective, the receiver system block diagram was briefly explained indicating the place and function of the GFSK demodulation block in the baseband backend.

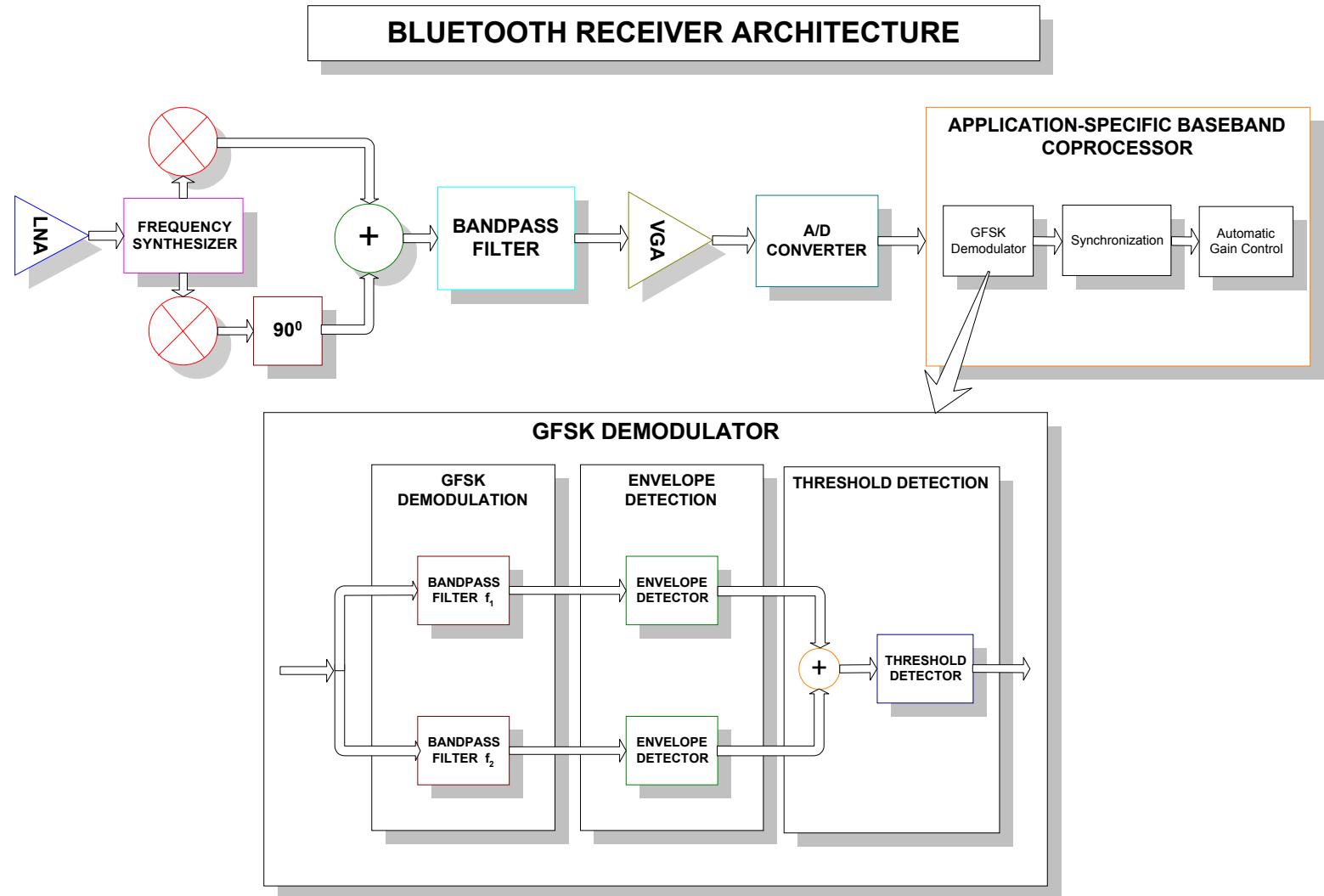


Figure 1.2 Bluetooth™ Receiver Architecture

2

SYSTEM DESIGN METHODOLOGY

2.1 INTEGRATED SYSTEM DESIGN

In order to design an integrated GFSK demodulator for a Bluetooth™ Receiver a structured design methodology was adopted.

A structured design methodology can be described as the overall system design strategy to organize and solve the system design issues at different steps of the system design process. Generally, the system design process is viewed as the development of a sequence of system models, where each subsequent version of the system model is more refined than the previous one. The refinement process continues until all the system design issues are resolved [17].

A structured design methodology was primarily employed to reduce the complexity of the design problem. Other important objectives of adopting such a design strategy were to:

- guarantee that the system performance goals are fulfilled. Therefore, the overall performance goals were expressed in terms of sub-goals such as acceptable physical size of the implemented chip, power consumption, speed, and the number of I/O pins, to mention a few.
- attain a shorter and predictable design time so that the project could be accomplished within the stipulated time frame. This implied that the risk of ending up with a non-working integrated circuit due to design errors, erroneous interfaces, unsatisfactory throughput, etc. must be minimized by using a good design method.

The degree of automation of the design process had a major impact on the overall design process .

The starting point for the system design process was the *System Specification*. Starting from the system specification, the system design process was mainly partitioned into two major phases:

- 1) *System Architecture Design*
- 2) *Integrated Circuit Design*

These two phases were followed by the *System Testing* phase. This sequence of steps is described in the figure below:

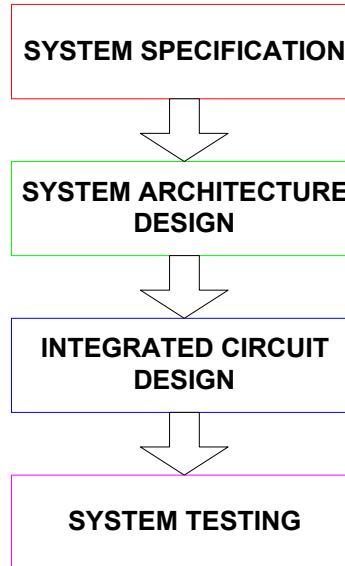


Figure 2.1 System Design Methodology

System Specification

The Bluetooth™ Specification [38], formulated by the Bluetooth Special Interest Group (SIG) was used extensively to extract the *system specification*. The system specification outlined the intended functions and performance criteria to be met by the designed system.

System Algorithm Design

The functions to be performed by the system being designed, as demanded by the system specification were refined and organized into a set of *system algorithms*.

System Partitioning

System Partitioning, generally, is a system optimization process that involves the segregation of the system algorithms for efficient mapping onto either hardware or software and is carried out taking into consideration various factors, most important of them being speed of execution, flexibility, communication overhead among various system blocks, etc. In the context of this project, system partitioning was actually performed at a level higher than the GFSK demodulator — at the level of baseband processing. The partitioning was very coarse that separated the data-intensive baseband processing algorithms for mapping onto an application-specific baseband coprocessor and code-intensive algorithms onto software to be executed on a general-purpose control processor.

System Architecture Design

A realization of the set of system algorithms in hardware (generally, either in hardware or software or both), yielded the *system architecture*.

System Validation

In order to check the validity of the system architecture against the system specification, *system modeling* was carried out and the resulting system model was subjected to *system simulation* to ascertain whether the system being designed meets the performance criteria as laid out in the system specification. *System Validation* was also used to extract the unknown design parameters for the blocks constituting the system architecture.

2.2 SELECTION OF DESIGN AUTOMATION TOOLS

The main requirement of *system-level codesign* tools arose from the need to concurrently model algorithmic, architectural, and hardware realized system blocks in a single environment and the need for architectural exploration. The main purpose of architectural exploration was to explore different hardware architectures to efficiently meet the performance targets with minimal re-work of the high-level specification. Architectural exploration was a multiple-choice exercise with each choice causing a ripple effect. The automated codesign tools could predict such effects.

As mentioned earlier, the degree of automation of the design process had a major impact on the overall design process. Considerable attention was paid in this project to the selection of appropriate design automation tools.

DSP designers build systems that are more complicated than ever, while market pressures force them to complete the designs in less time and at a lower cost. Most DSP systems are complex and involve a wide variety of design disciplines. Tasks to be completed range from algorithm, hardware, and software design to system simulation and integration/debugging. As a result, designers typically must use multiple tools, ranging from filter design packages to block diagram programming environments to hardware synthesis tools. DSP design tool vendors have been increasing the design tool support for the various stages of the design process by adding capabilities to the existing tools and by linking their tools to those supplied by other companies [59].

2.2.1 WORKSTATION-BASED TOOLS

Mentor Graphics™, **Synopsys™**, **Cadence™**, and **Hewlett-Packard™** have capable DSP design tool offerings.

Mentor Graphics™ has integrated the **DSP Architect** design entry tool, the **MISTRAL 2** hardware synthesis technology, a VHDL simulator, and the TI TMS320C52 processor simulation model providing a great deal of power within a single environment. However, the DSP Architect currently has no way to generate C52 assembly code from a high level specification, so the code running on the C52 has to be designed manually. Additionally, the C52 processor model does not feature an interactive user interface, and this lack stifles attempts to debug both the hardware and the software [62].

Synopsys™ has combined the **COSSAP** simulation environment and processor simulation models for AT&T's DSP1610 fixed-point DSP chip allowing algorithms expressed as block diagrams within COSSAP to be cosimulated with DSP1610 machine code executing on the DSP1610 processor simulation model. Additional processor models are likely to be integrated into COSSAP in the future. Additionally, COSSAP can probably provide

improved hardware synthesis capabilities from a combination of COSSAP's VHDL code generation capabilities and Synopsys™ Behavioral Compiler hardware synthesis tool [60].

Cadence™ has also integrated **Signal Processing Worksystem's (SPW) Hardware Design System (HDS)** and the AT&T DSP1610 processor. The fixed-point optimizer option for SPW aids design engineers in creating fixed-point implementations of floating-point algorithms. The fixed-point optimizer allows the designer to specify a block diagram design in floating-point, and then give the system target signal-to-quantization-noise performance specifications. SPW then simulates the system using user-provided input signals and calculates appropriate fixed-point parameters (signed/unsigned, number of integer bits, number of fractional bits) for the different stages in the signal flow diagram. This is an important capability for designers targeting reduced cost or high-speed fixed-point design [61].

Hewlett-Packard™'s Advanced Design System (ADS) is based on the **Ptolemy** design tool developed at the University of California, Berkeley. ADS has a flexible graphical tool for designing electronic systems, including DSP systems. The **ADS DSP Designer** provides block-diagram based design entry, several kinds of dataflow and discrete event simulation capabilities, C and assembly code generation, VHDL code generation, fixed-point simulation, and support for several computational models [63].

2.2.2 PC-BASED TOOLS

Hyperception™'s Hypersignal-Block Diagram package has C code generation capability, the ability to generate a stand-alone executable program from a block diagram without actual code generation. This is done by linking together pre-compiled object code to form a complete executable that can run outside of Hypersignal-Block Diagram. This approach works equally well for applications that execute on the PC as well as those that run on a DSP add-in card. Because these stand-alone executables can make use of the Microsoft™ Windows graphical user interface, rapid generation of impressive-looking applications is possible.

Additionally, Hyperception™ has a standardized interface to a wide variety of DSP plug-in boards. This standard device driver specification allows the user to make use of cards from different manufacturers, and simplifies retargeting Hypersignal-Block Diagram to new boards [64].

Signalogic™'s DSPower is a generic block-diagram front end that can generate C code as well as code for a variety of other tools, e.g., Hypersignal and MATLAB. This approach of integration through code generation allows users to take advantage of the best features of several different tools [65].

Elanix™ SystemView is another DSP system design package for communication system design that offers interface to **Xilinx™** for VHDL code generation [66].

MathWorks™ provides a complete system-level design environment based on **Simulink™**, a powerful block diagram-based simulation environment. Simulink™ is built on top of **MATLAB™**, the proven software for DSP algorithm development. Simulink™ streamlines communication system and DSP design by providing the fastest path from product concept to validated system model to a working system prototype. It maximizes scarce engineering resources by enabling to move a design effortlessly through algorithm development, behavioral simulation, model verification, and system prototyping without having to transfer data, rewrite code, or change software environments. With Simulink™, its possible to test design concepts and tradeoffs earlier in the development process. By verifying the design at

the system level, the risk of expensive errors in software and/or silicon is minimized. Eliminating these errors early cuts down the design time and development costs [67].

Design Automation Tool vendors have essentially two ways to increase DSP tool capabilities. One is to add new capabilities to their own tools, providing better design coverage within their tool suite. The other is to forge links to complementary tools. Both are important and useful approaches, since no single tool is ever appropriate for all designs. MathWorks™ has combined with Xilinx™ to provide access to the rapid prototyping phase through the **System Generator** Toolbox and **xPC** Toolbox for hardware-in-the-loop simulation [67] [68].

MATLAB™ and its associated tools were selected for this project, the main reason being their availability and prior familiarity with some of the tools. Details of the tools deployed for accomplishing specific design tasks are illustrated below:

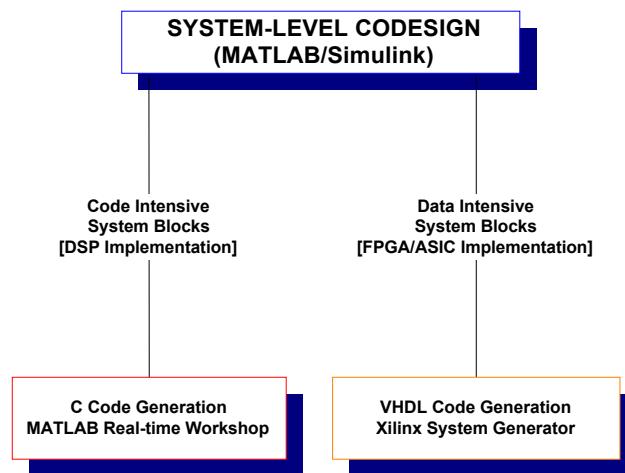


Figure 2.2 The MATLAB™ CoDesign Environment for Hardware & Software

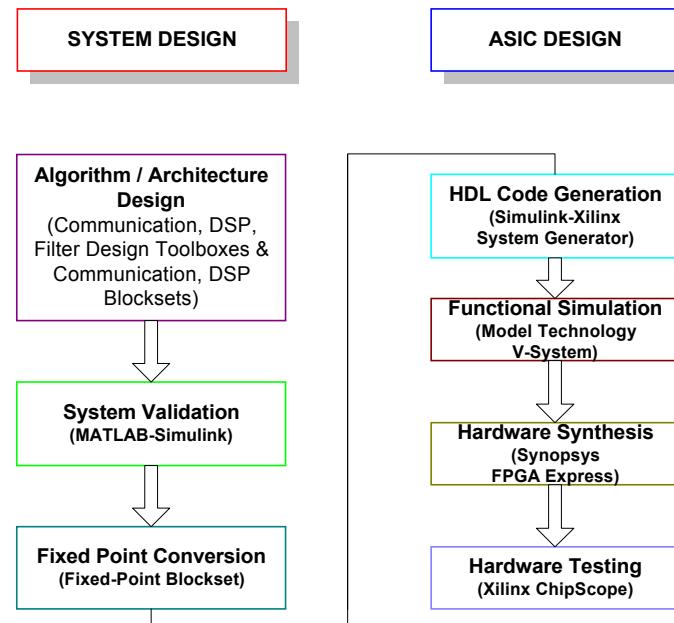


Figure 2.3 The MATLAB™ Design Environment for System & Hardware-level Design

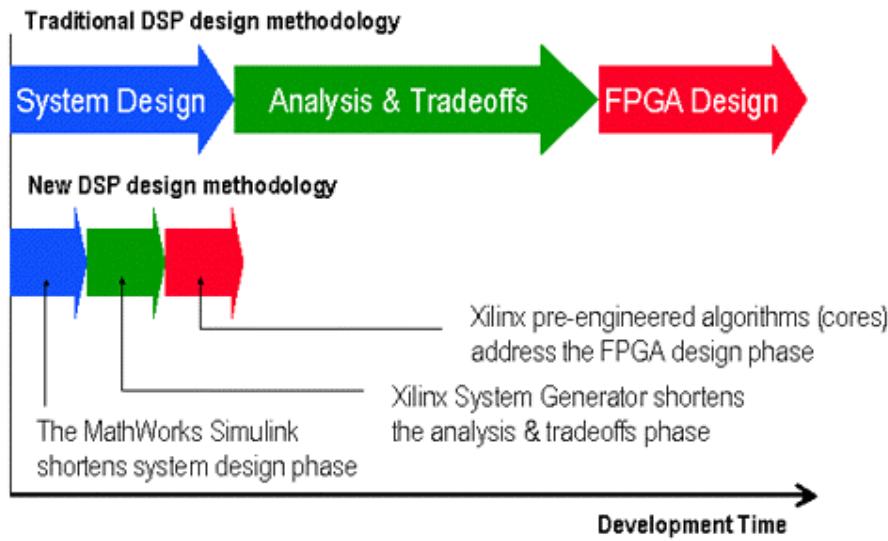


Figure 2.4 Impact of having a Common Design Environment on System Design Time [68]

SUMMARY

This chapter introduced the integrated system design issues and described the design flow for handling complex integrated system designs. It explained the approach followed by various design automation tools to have a common design environment for system-level design and hardware/software-level design and such a set of tools used for this project to implement the design.

3

A/D CONVERSION

The Analog-to-Digital Converters (A/D Converters or ADCs) are a key building block in digital communication receivers employing digital signal processing techniques. In bridging the gap between the analog and the digital domains, the performance of the ADCs often limits the achievable performance of the receiver. Many architectural choices in a receiver are affected by the A/D Converter architecture. The essential parameters that generally characterize A/D Converter architectures are speed, resolution, and power. In the context of CMOS mixed-signal solutions to digital communication receivers, other important A/D Converter parameters include input capacitance, settling time (time allowed for the sample-and-hold circuit to settle to its final value while driving the input capacitance), latency (through the A/D Converter), comparator metastability, and the output sparkle codes.

3.1 A/D CONVERTER ARCHITECTURES

The A/D Converter performs the following functions:

- 1) Signal Sampling
- 2) Signal Quantization
- 3) Signal Coding

Signal Sampling is essentially an operation of frequency translation or frequency mixing where the sampled signal frequency is mixed with the sampling frequency of the local oscillator that generates a series of impulses at the sampling frequency.

Signal Quantization involves rounding off the samples to the nearest quantization value. This process introduces *quantization noise* into the sampled signal.

Signal Coding involves representation of each quantized signal sample in the format of a unique b -bit binary sequence [14].

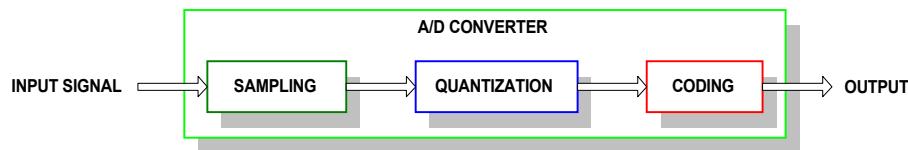


Figure 3.1 A/D Converter Functions

The following A/D Converter architectures were investigated to select the appropriate architecture for the Bluetooth™ receiver:

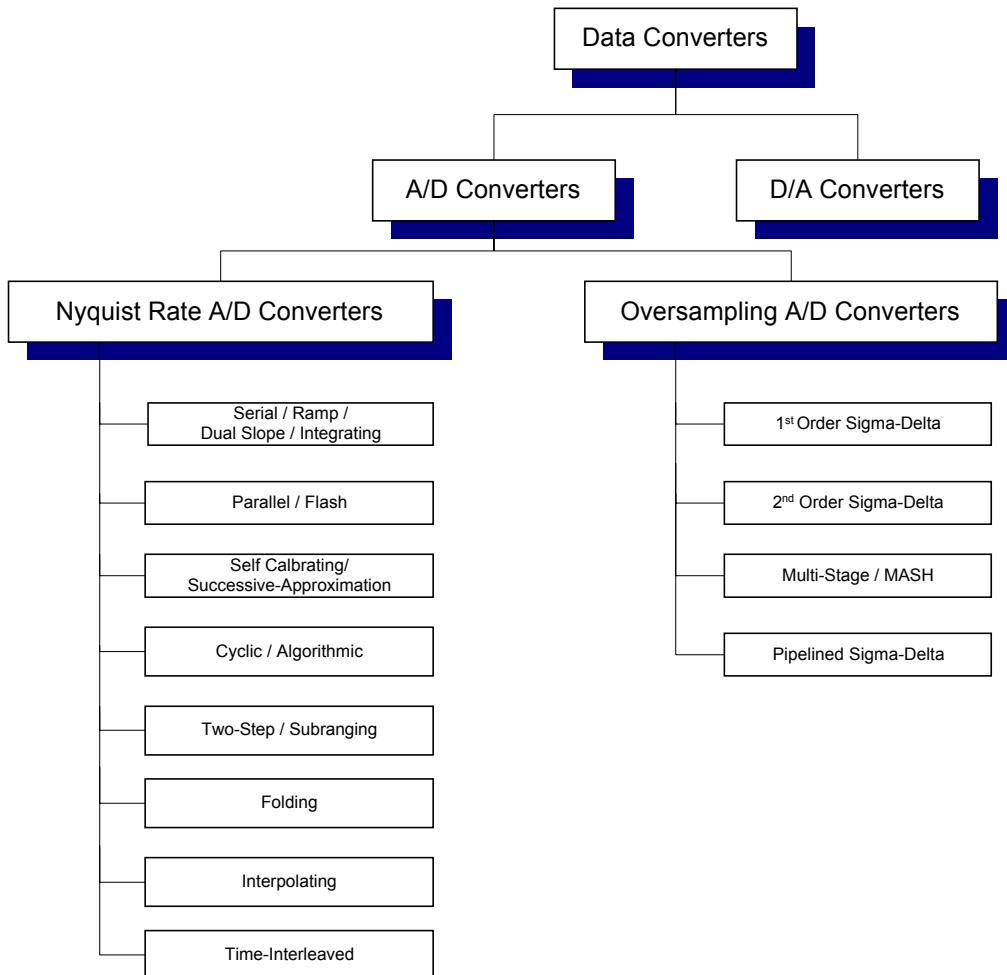


Figure 3.2 Common A/D Converter Architectures

The most suitable type of A/D converters for the Bluetooth™ receiver are the Flash Converters. The reason being that at an intermediate frequency (IF) of 1 MHz, the flash converters provide the best performance in terms of converted bandwidth and resolution.

3.2 FLASH A/D CONVERTERS

The principle of operation of Flash Converters is very simple. The input voltage is compared with all the possible thresholds that define the transition between two successive

codes. Since for N bits there are 2^N quantization steps, 2^{N-1} comparators are necessary. The comparison operations are performed simultaneously and only one clock cycle is required to perform the entire conversion. The necessary 2^{N-1} voltages are obtained with a resistive divider. The outputs of the comparators are the input of a logic circuit encoding the result into its digital code. The speed of a flash converter is determined by the speed of the comparators and by the encoding logic. In general, the encoding logic is very fast and so the comparator speed is the main concern. Moreover, even for medium resolution, the number of comparators is very large which rises exponentially with the linear increase in resolution. Thus the power dissipation and the silicon area rapidly reach unacceptable values. For 8 or more bits of resolution, a more convenient technique is the 2-step or sub-ranging flash converter.

As the flash converters are not efficient from the point of view of low-power operation. Therefore, oversampling converters can provide better performance if they can be designed to handle a converted bandwidth of 2 MHz [11].

3.3 OVERSAMPLING A/D CONVERTERS

Oversampling converters relax the requirements placed on the analog circuits at the expense of more complicated digital circuits. This tradeoff becomes more desirable for modern sub-micron process technologies operating at low-voltage power supplies where complicated high-speed digital circuits are more easily realized in less area, but the realization of high-resolution analog circuits is complicated by low-voltage power supplies and poor transistor output impedances caused by short channel effects. With oversampling data converters, the analog components have reduced the requirements on matching tolerances and amplifier gains. A second advantage of oversampling converters is that they simplify the requirements placed on the analog antialiasing filters for A/D converters. Furthermore, a sample-and-hold amplifier is not required at the input of an oversampling A/D converter. By sampling a signal at a rate much higher than the Nyquist rate, extra bits of resolution can be extracted from A/D converters but this extra resolution can be obtained with lower oversampling rates by spectrally shaping the quantization noise through the use of feedback. The use of shaped quantization noise applied to oversampling signals is commonly referred to as delta-sigma ($\Delta-\Sigma$) modulation [11] [12].

3.4 CONSIDERATIONS FOR THE SELECTION OF SAMPLING FREQUENCY

Oversampling

Oversampling occurs when the signals of interest are bandlimited to f_0 while the sampling rate is at f_s , where, $f_s > 2f_0$ ($2f_0$ being the Nyquist rate or, equivalently, the minimum sampling rate for signals bandlimited to f_0). Oversampling ratio, OSR, is defined as:

$$OSR = f_s/2f_0$$

$$SNR_{\max} = 6.02N + 1.76 + 10\log_{10} OSR$$

The first term is the SNR due to the N -bit quantizer while the OSR term is the SNR enhancement obtained from oversampling. Straight oversampling yields an SNR improvement of 3 dB/octave. The reason for this SNR improvement through the use of oversampling is that when quantized samples are averaged together, the signal portion adds

linearly, whereas, the noise portion adds as the square root of the sum of squares. While oversampling improves the Signal-to-Noise Ratio (SNR), it does not improve linearity [11].

Oversampling With Noise Shaping

For a 1st Order Noise Shaping Loop:

$$SNR_{max} = 6.02N + 1.76 - 5.17 + 30\log_{10} OSR$$

For a 2nd Order Noise Shaping Loop:

$$SNR_{max} = 6.02N + 1.76 - 12.9 + 50\log_{10} OSR$$

The design of GFSK demodulator does not incorporate oversampling A/D converters because designing oversampling converters has a separate set of design issues that have to be dealt with separately and could not be addressed within the stipulated time of this project. As a result, in the absence of quantization noise shaping, a high oversampling ratio has been used to offset the effects of the quantization noise on the acceptable bit-error rate (BER). The exact values of the A/D converter bits of resolution and oversampling ratio were extracted through the simulation of the system model and will be described in chapter 6. The resolution is 16-bits and the oversampling ratio is 20.

SUMMARY

This chapter introduced the A/D Converters that are an integral component of a DSP system and discussed the considerations for the selection of an appropriate sampling rate for the GFSK demodulator.

4

SYSTEM ALGORITHM DESIGN

In order to design a demodulation algorithm for a Bluetooth receiver it was important to precisely understand the modulation format used by a Bluetooth transmitter. A Bluetooth Transmitter uses Frequency-Hopping Spread-Spectrum (FH-SS) as *secondary modulation* preceded by Gaussian Frequency Shift Keying (GFSK) as *primary modulation*.

The secondary, spread spectrum modulation will be explained first.

4.1 SPREAD SPECTRUM MODULATION

Spread-Spectrum modulation, with its inherent interference attenuation capability, has over the years become an increasingly popular technique for use in many different systems. Applications range from anti-jam systems, to Code Division Multiple Access (CDMA) systems, to systems designed to combat Multipath distortion. Spread-Spectrum modulation-based communication systems have been developed since about 1950's. The initial applications have been to military anti-jamming tactical communications, to guidance systems for missiles and space rockets, to experimental anti-multipath systems, and other applications [23a].

A definition of Spread-Spectrum that adequately reflects the characteristics of this technique is as follows:

Spread-Spectrum is a means of information transmission in which the modulated signal occupies a bandwidth in excess of the minimum necessary to send the information; the signal band spread is accomplished by means of a code which is independent of the data, and a synchronized reception with the code at the receiver is used for despreading and subsequent data recovery [23a].

Under this definition, standard modulation schemes such as FM and PCM which also spread the spectrum of an information signal do not qualify as spread spectrum [23a].

Motivation For Spectrum Spreading

There are many reasons for spreading the spectrum, and if done properly, multiplicity of benefits can accrue simultaneously. Some of these are :

- 1) Anti-jamming
- 2) Anti-interference
- 3) Low Probability of Intercept
- 4) Multiple-user random-access communications with selective addressing capability

4.1.1 SPECTRUM SPREADING TECHNIQUES

The means by which the spectrum is spread is crucial. Several of the techniques are :

Direct-Sequence Technique

In Direct-Sequence technique, a fast pseudorandomly generated code sequence causes phase transitions in the carrier containing data.

Frequency Hopping Technique

In Frequency Hopping technique, the carrier is caused to shift frequency in a pseudorandom way.

Time Hopping Technique

In Time Hopping technique, bursts of signal are initiated at pseudorandom times [23a]

4.1.2 DIRECT-SEQUENCE SPREAD-SPECTRUM

Direct-Sequence Spread-Spectrum (DS-SS) results when a primary modulated signal is multiplied by a spreading signal in a mixer called the 'Spreading Correlator'. The spreading code rate is :

$$R_c = 1 / T_c$$

where T_c is the time duration of a single pulse, called a *chip* having 100-1000 times shorter duration than a data bit ($T_c \ll T_b$ — bit duration). Consequently, the transmitted spectrum will be 100-1000 times greater than the bandwidth of the primary-modulated signal, having been finely chopped-up by a wideband, unique spreading code. The resulting signal spectrum is highly correlated with the spectrum of the spreading code.

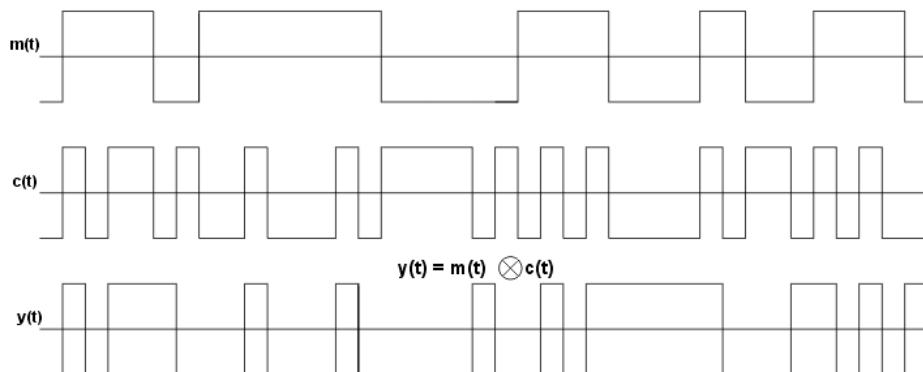


Figure 4.1 Direct Sequence Spread Spectrum Modulation [5]

4.1.3 FREQUENCY HOPPING SPREAD SPECTRUM

Frequency Hopping Spread Spectrum (FH-SS), spreads the primary-modulated signal energy over a wide frequency band. A FH-SS transmitter switches from one narrowband frequency to another at a specific rate and in accordance with a predefined frequency hopping code sequence, sending several data bits at each narrow frequency band. By limiting the time spent by a transmitter at each narrow frequency band, the probability of any two FH-SS transmitters using the same narrow frequency band at the same time is minimized.

The frequency hopping rate is usually selected to be either equal to the (coded or uncoded) symbol rate or faster than that rate. In a *fast-hopped signal* there are multiple hops per symbol. On the other hand, in a *slow-hopped signal* hopping is performed at the symbol rate [2].

Typically, the frequency spectrum is divided into 1-MHz channels, and frequency-hopped systems must not spend too much time on any one channel — no more than 400 ms out of any 20 seconds on a channel in the 900-MHz band, and no more than 400 ms out of 30 seconds at 2.4 GHz. They must also hop through at least 50 channels in the 900-MHz band or 75 channels in the 2.4 GHz band.

The primary, GFSK modulation is explained next.

4.2 GFSK MODULATION

The GFSK Modulation is a form of the Continuous-Phase FSK (CP-FSK) which, in turn, is a modification of the Discontinuous-Phase FSK modulation. Therefore, it is necessary to first describe the Discontinuous-Phase FSK modulation and the Continuous-Phase FSK Modulation schemes.

4.2.1 FREQUENCY SHIFT KEYING [FSK]

In the (Binary) FSK modulation, the 0's and 1's in the baseband digital signal are transmitted using two different frequencies, f_1 and $f_2 = f_1 + \Delta f$ (where $\Delta f = f_2 - f_1$) shifting from one frequency to the other according to the binary value of the data sequence.

The *Modulation Bandwidth* for an FSK signal is

$$B_{FSK} = 3 [maximum\ Bit\ Rate\ (f_b)] + [maximum\ Frequency\ Shift\ (\Delta f)]$$

The Modulation Index for an FSK signal is

$$I_m = \beta = \frac{\Delta f}{f_c}$$

where: $f_c = Modulation\ / Carrier\ Frequency$

FSK is much less susceptible to corruption by unwanted amplitude modulation - due to noise or transients. However, in general, there is no direct relationship between the (two) frequencies of the (Binary) FSK modulated signal and the Bit Rate. So, in principle, discontinuities in the transmitted waveform can occur. In order to avoid this problem, Continuous Wave signals are used, which incorporate smooth transitions between the (two) frequencies [4] [6].

4.2.2 GAUSSIAN FREQUENCY SHIFT KEYING [GFSK]

GFSK can be viewed as a form of the Continuous-Phase Frequency Shift Keying (CPFSK). In CPFSK modulation, the high-frequency components in the output spectrum of the modulated signal are reduced because of the continuous phase variation of the CPFSK-modulated signal. In GFSK, the baseband signal is passed through a Pulse-Shaping Gaussian Low Pass Filter before modulation in order to shape the pulses to give them half-sinusoidal shape so that the phase trajectory of the FSK signal becomes smooth and the instantaneous frequency variations over time are stabilized. This has the following advantages:

- 1) The envelope of the modulated signal is constant. This allows the GFSK modulated signal to be operated with a Class-C Power Amplifier without introducing Spectrum Regeneration. Therefore, lower power consumption and higher power efficiency can be achieved.
- 2) The output spectrum has a narrow Main Lobe and a lower level of spectral Side Lobes than in Discontinuous-Phase FSK. This keeps the adjacent channel interference to low levels achieving high spectral efficiency. This is important for a bandlimited channel, and particularly important when the channel is nonlinear.
- 3) The GFSK modulated signal can be demodulated by Non-Coherent Demodulation schemes leading to low-cost GFSK receivers [9] [4].

The sole purpose of pre-modulation lowpass filtering is to narrow the transmitted spectrum of the FSK modulated transmitted signal. It is important, however, that the lowpass filter have a well-behaved time-domain response. A class of filters that has a well-behaved time domain response is Gaussian Filters. The frequency response of a Gaussian Lowpass Filter is 'Gaussian' in nature, following the following relation:

$$H(\omega) = \tau \sqrt{2\pi} e^{-\frac{1}{2}(\tau\omega)^2}$$

where: ω = frequency (in radians/sec)

τ = constant

This is the shape of the Gaussian or Normal Probability Density Function.

A peculiar property of a filter with a Gaussian frequency response is that its time-domain or impulse response is Gaussian as well. This can be seen by taking the inverse Fourier transform of its frequency response

$$h(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} H(\omega) e^{-j\omega t} d\omega = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \left[\tau \sqrt{2\pi} e^{-\frac{1}{2}(\tau\omega)^2} \right] e^{-j\omega t} d\omega$$

which yields

$$h(t) = e^{-\frac{1}{2}\left(\frac{t}{\tau}\right)^2}$$

The impulse response of the Gaussian Filter is well-behaved, exhibiting no ringing or overshoot. However, the frequency response of the filter tends to fall off rather slowly. In general, the more gradual the frequency-domain response of a filter, the better its time-domain response (exceptions are the digital FIR filters). Gaussian filters have one of the most gradual frequency-domain responses of any analog filter type [9] [4].

The frequency-domain response of the Gaussian Filter is linear. This implies that the phase of the Gaussian filter is linear.

The bandwidth of the Gaussian Filter is often given in terms of its relation to the Bit Rate

$$T = T_b = 1/f_b$$

where: T or T_b = Bit Period or Duration of the filter

f_b = Bit Rate

If B (or W) is the 3-dB bandwidth of the filter, then the filter response can be specified in terms of its *Relative Bandwidth*, or *BT Product* (also expressed as WT_b)

$$BT = \text{Filter Bandwidth} \cdot \text{Bit Period} = \frac{\text{Filter Bandwidth}}{\text{Bitrate}}$$

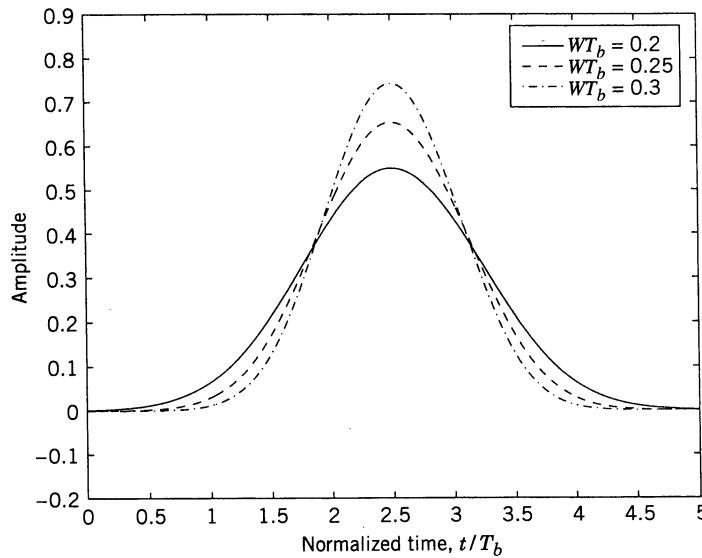


Figure 4.2 The effect of Gaussian filter bandwidth on the signal frequency spectrum

Gaussian Filters with smaller relative bandwidths cause faster spectral roll-offs. These faster spectral roll-offs have a price, however. As the relative filter bandwidth is lowered, more and more ISI (Inter-Symbol Interference) is imparted to the waveform. ISI is caused by data bits in the periods preceding the present data period not fully settling out. Eye Diagram gives a qualitative indication of the ISI [9] [4].

If the Gaussian Filter is tuned into an infinite bandwidth ($BT = \infty$, i.e., no filter is used) a GFSK signal with a Modulation Index of I_m can be expressed as a modulation by $I_m(D[k]/(2T_b))$ around a centre carrier frequency f_c . Therefore, such a GFSK modulated signal can be expressed as:

$$S[t] = \cos[2\pi(f_c + (I_m D[k] P[t-nT]/2T_b)t]$$

where: $D(t) = \sum_{n=0}^N D[k] P[t-nT_b]$ = Binary Data represented as a Rectangular Pulse Stream.

$1/T_b$ = Bit Duration of the Baseband Modulating Signal

f_c = Centre / Carrier Frequency

I_m = Modulation Index

A Gaussian Filter is represented by the following Transfer Function:

$$G[\omega] = A_0 e^{-\alpha\omega^2} e^{-j\omega\tau_0}$$

where: $\alpha = (\ln 2) / (2B_{3dB}^2)$

The Impulse Response of a Gaussian Filter is:

$$H(t) = (A_0 / \sqrt{\pi}) \beta e^{-[(t-t_0)\beta]^2}$$

$$\text{where: } \beta = \sqrt{\frac{2}{\ln 2}} \pi f_c B T_b$$

So the GFSK modulated signal is:

$$S[t] = A_0 \cos \left[2\pi f_c t + \left(2\pi I_m / 2T_b \right) \int_0^t ((D(\tau) \otimes H(\tau)) / 2T_b) d\tau \right] \quad [4]$$

The reception of a GFSK modulated signal, generally, involves two steps:

- 1) Demodulation
- 2) Decoding / Detection

4.3 DEMODULATION ALGORITHMS FOR GFSK

As the GFSK modulation is a modified form of the FSK modulation, therefore, the demodulation algorithms for the FSK modulated signals are applicable to the GFSK modulated signals as well.

The concept of a *matched filter* is central to any optimal demodulation algorithm.

A *matched filter* is a filter whose frequency response is designed to exactly match the frequency spectrum of the input signal. The operation of a matched filter is the same as correlating a signal with a delayed copy of itself.

There are two types of algorithms for the FSK Demodulation:

- 1) Coherent / Synchronous Demodulation
- 2) Non-Coherent / Asynchronous Demodulation

4.3.1 COHERENT DEMODULATION

In phase-coherent or synchronous FSK demodulation, both the magnitude and the phase response of the matched filters is required for demodulating the received signal, therefore, exact knowledge of the phase of the incoming signal is required. In this demodulation scheme, the phase of the received signal is estimated by correlating it with each of the possible received signals [2] [3] [7].

4.3.2 NONCOHERENT DEMODULATION

In phase-noncoherent or asynchronous FSK demodulation, only the magnitude response of the matched filters is required for demodulating the received signal, therefore, exact knowledge of the phase of the incoming signal is not required. However, to prevent significant overlap of the passbands of the two filters [which causes *intersymbol interference* (ISI)], the frequency spacing must be at least $\Delta f T >> 1$, for *orthogonal signaling* [2] [3] [7].

Coherent demodulation is superior in performance to noncoherent demodulation but the requirement of estimating the carrier phases makes coherent FSK demodulation quite

complex. Therefore, noncoherent demodulation is the most commonly used demodulation scheme for most receivers and has been selected for the GFSK demodulator design.

In this design, noncoherent demodulation algorithm was adopted. Apart from its simplicity, the main reason for the selection of noncoherent demodulation algorithm in the GFSK demodulator design was that due to the use of FH-SS as secondary modulation, coherent demodulation is not feasible because it is difficult to maintain phase coherence in the synthesis of the frequencies used in the frequency hopping sequence and, also, in the propagation of the signal over the channel as the signal is hopped from one frequency to another over a wide frequency band [7].

The following most common algorithms for noncoherent FSK demodulation were considered:

- 1) Matched filter-based demodulation
- 2) Frequency Discriminator-based demodulation

4.3.3 MATCHED FILTER BASED DEMODULATION

In this FSK demodulation algorithm, an FSK modulated signal is decomposed into two ASK modulated signals at two different carrier frequencies. Therefore, two *matched filters*, one for each carrier frequency, are used to demodulate the received FSK modulated signal. If one carrier frequency is present in the absence of noise, it is assumed that the output of one matched filter is zero and the output of the other matched filter is maximum and vice versa [5].

4.3.4 FREQUENCY DISCRIMINATOR BASED DEMODULATION

In this algorithm, the frequency variations in the FSK modulated signal are converted into amplitude variations [5].

The selection of either of the two above-mentioned demodulation algorithms was made on the basis of their architectural details and will be discussed in the following chapter.

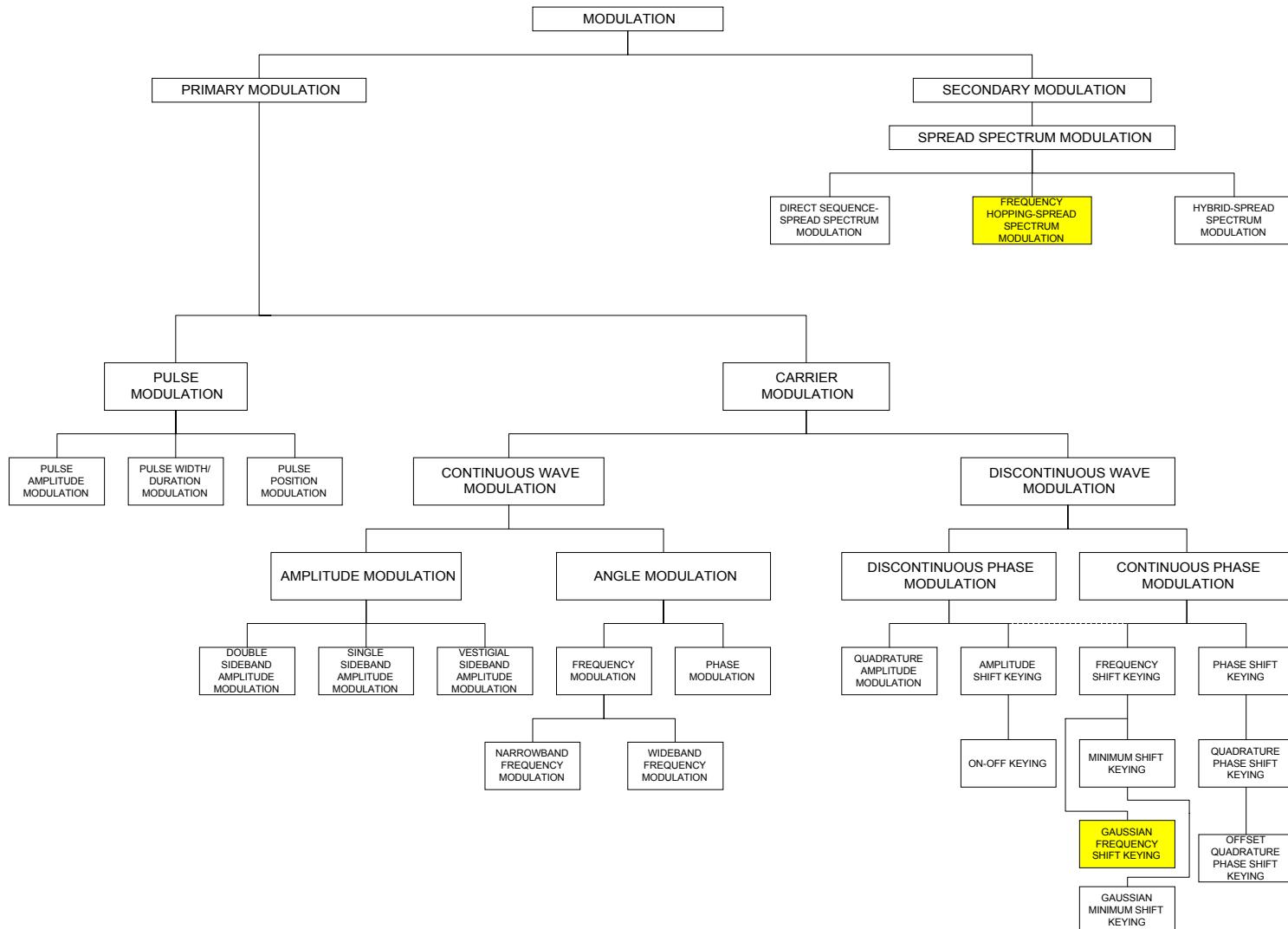


Figure 4.3 Classification of Modulation Formats

4.4 DETECTION ALGORITHMS FOR GFSK

In Digital Communications, the terms Demodulation and Detection are used somewhat interchangeably, although Demodulation emphasizes removal of the carrier, and Detection includes the process of symbol decision [3].

There are two types of detection algorithms based on the *optimum detection* criterion:

- 1) Envelope Detection
- 2) Square Law Detection

Both types of algorithms are exactly equivalent in performance but their architectural implementation details vary.

4.5 BLUETOOTH MODULATION SPECIFICATION

The Receiver based on the Bluetooth™ specification is a Frequency-Hopped Spread Spectrum (FHSS) System, with a hop rate of 1600 hops/sec (slow hopping), operating in the Industrial, Scientific, and Medical (ISM) Band (2.402 - 2.480 GHz in Europe & North America). The number of frequency hop channels is 79. The length of the Pseudo-random Frequency Hop Sequence is 2^{24} with the largest possible hop of 78 MHz. The Symbol Rate is 1Mb/s and the modulation scheme is 2-Level Gaussian-Filtered FSK (GFSK), baseband filtered with a Gaussian Filter having a 3-dB Bandwidth of 500 KHz (Bandwidth-Symbol Interval product, BT= 0.5). The Modulation Index can vary from 0.28 to 0.35, i.e., a Frequency Deviation, Δf , of ± 140 KHz to ± 175 KHz [1] [38].

SUMMARY

This chapter discussed the algorithmic design of the GFSK demodulator. The type of demodulation algorithm selected was noncoherent demodulation. The selection of demodulation and detection algorithms was postponed till the architecture design phase due to their strong coupling to the architectural details and will be discussed in the following chapter.

5

SYSTEM ARCHITECTURE DESIGN

5.1 DEMODULATOR ARCHITECTURES

Two realizations of a matched filter are available, therefore, two matched filter-based demodulator architectures are possible:

- 1) Correlator-based Demodulator
- 2) Convolver-based Demodulator

5.1.1 CORRELATOR-BASED DEMODULATOR

In this case, for noncoherent demodulation, there are two correlators per signal waveform. The received signal is correlated with the basis functions (quadrature carriers). The outputs of the correlators are sampled at the end of the signal interval and are passed to the Detector.

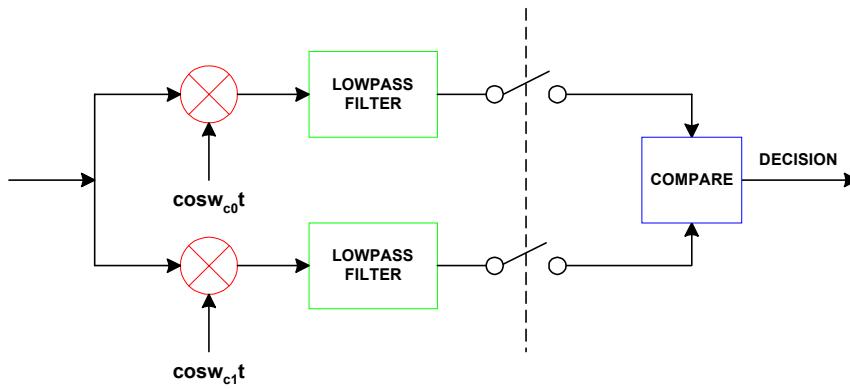


Figure 5.1 Correlator-Based Demodulator

5.1.2 CONVOLVER-BASED DEMODULATOR

The convolver-based demodulator is equivalent to the correlator-based demodulator where the correlators are replaced by convolvers [2].

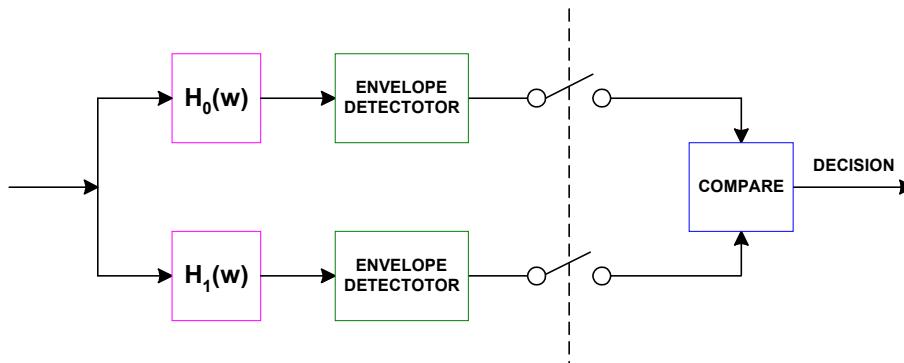


Figure 5.2 Convolver-based Demodulator

5.2 DETECTOR ARCHITECTURES

5.2.1 ENVELOPE DETECTOR

An envelope detector consists of a full-wave rectifier and a lowpass filter having a cut-off frequency equal to the bandwidth of the signal. The envelope detector is matched to the signal envelope and not to the signal itself. Ideally, the output of the envelope detector is of the form:

$$d(t) = g_1 + g_2 m(t)$$

where, g_1 represents a dc component and g_2 is a gain factor due to the signal demodulator. The dc component can be eliminated by passing $d(t)$ through a transformer, whose output is $g_2 m(t)$ [3].

A half-wave rectifier can also be used in an envelope detector but this results in reduced detected signal energy and can potentially cause a higher *bit error rate* (BER) particularly if the transmission channel is noisy.

5.2.2 SQUARE-LAW DETECTOR

A square-law detector consists of a pair of squaring multipliers followed by a summer [3].

As described earlier, an FSK signal can be considered as a superposition of two signal waveforms. One of these has a transform centered at f_1 with sidebands following a $(\sin f)/f$ envelope. The second component is centered at f_2 with a similar envelope. Therefore, in the Convolver-based Demodulation, two Bandpass Filters having center frequencies of f_1 and f_2 having bandwidth $W_f = 1/T$ (where T = Symbol Interval) act as Signal Correlators [6].

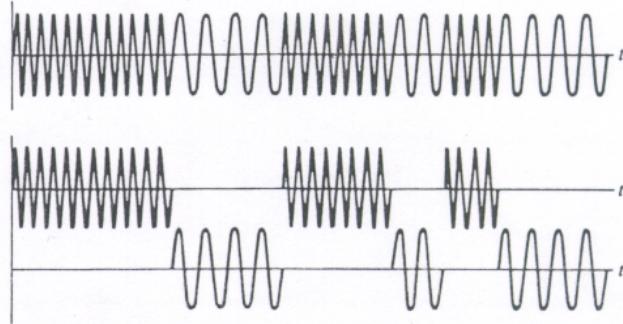


Figure 5.3 Decomposition of an FSK Signal into two ASK Signals [6]

The frequency discriminator-based demodulator uses two resonant circuits one tuned above and the other tuned below the carrier frequency. The inputs to the two circuits are equal but of opposite sign. The outputs of the resonant circuits are envelope detected and subtracted to give the demodulated signal. This scheme yields a slightly poor performance than the matched filter demodulation [5].

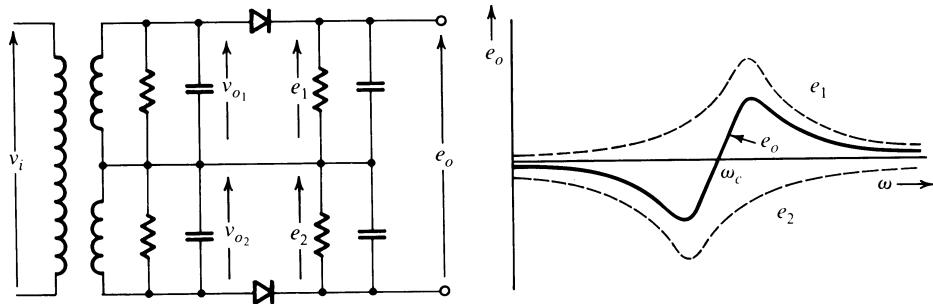


Figure 5.4 Frequency Discriminator-based FSK Demodulator [5]

The matched filter-based architecture was selected for this project because of its claimed superior performance, fully digital implementation and the possibility of reusing most of its constituent blocks in the PN Code acquisition and tracking designs as illustrated below:

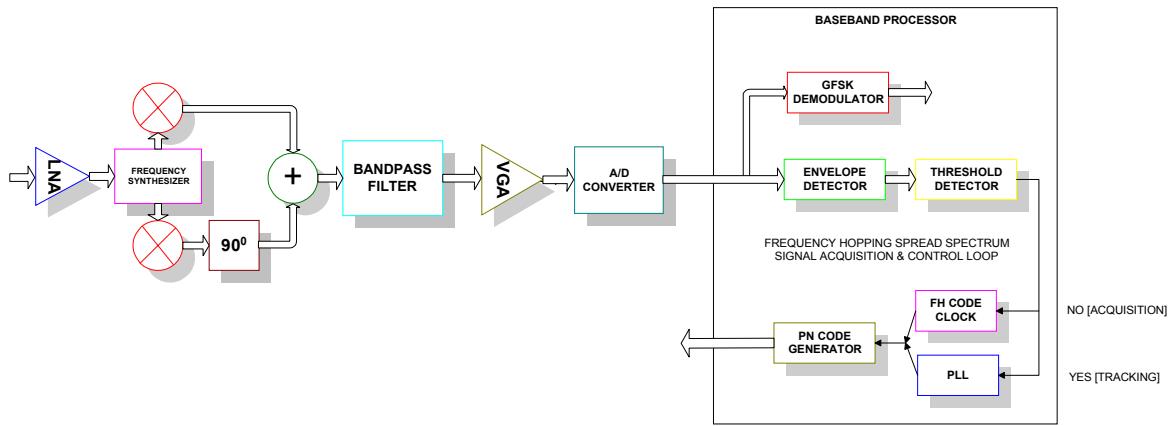


Figure 5.5 Block-level details of the Baseband Coprocessor

5.3 SYSTEM ARCHITECTURE

The convolver-based demodulator was implemented practically by approximating the convolvers with bandpass filters which were followed by the envelope detectors comprising full-wave rectifiers and low-pass filters.

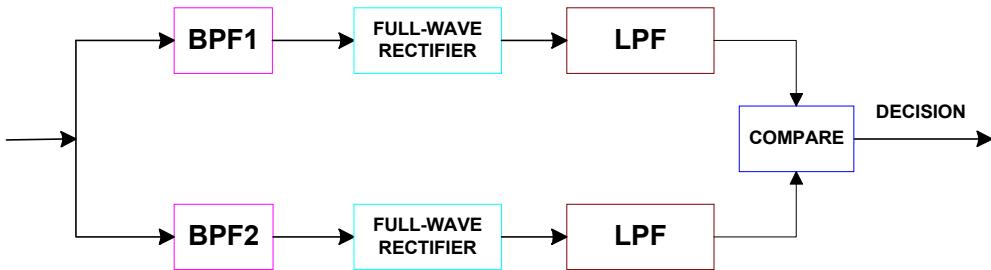


Figure 5.6 Convolver-based Demodulator Architecture

The Bandpass Filters can be realized as either analog or digital. Analog filters can be cheaper, faster, and have greater dynamic range; digital filters outstrip their analog counterparts in flexibility. The ability to create filters that have arbitrary frequency response curve shapes, and filters that meet the performance constraints, such as passband width and transition region width, is well beyond that of analog filters.

Quantization is a natural outgrowth of digital filtering and digital signal processing development. Also, there is a growing need for fixed-point filters that meet power, cost, and size restrictions. Quantization is performed to convert a floating-point filter to a fixed-point filter.

5.4 DIGITAL FILTERS

A *digital filter* is a digital system that filters a digital input signal according to some pre-designated criteria (e.g., selectively discriminate signals in different frequency bands and/or modify the phase of the signals) and produces a digital output signal [16].

A digital filter can be represented by a block diagram:



Figure 5.7 Digital Filter Block Diagram

Input $x(nT)$ and output $y(nT)$ are the excitation and response of the digital filter, respectively. The response is related to the excitation by some rule of correspondence.

$$y(nT) = \mathfrak{R}x(nT)$$

where \mathfrak{R} is an operator .

Unlike a *transmission digital filter* where the continuous data stream entering the filter is modified to form a new continuous data stream, a *digital matched filter* output is a flow of individual results which are analyzed to identify the individual point where the match occurred.

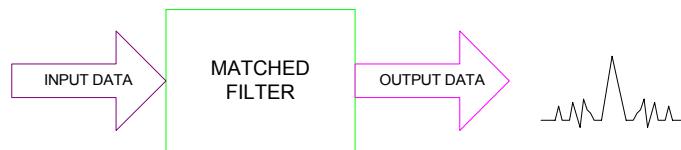


Figure 5.8 Digital Matched Filter Block Diagram

Digital Filters, like other systems, can be classified as linear or non-linear, causal or noncausal, time-invariant or time-variant, and spectral or spatial.

Just as analog filters are characterized in terms of *differential equations*, digital filters are characterized in terms of *difference equations*. Two types of digital filters can be identified, *non-recursive filters* and *recursive filters*. If a digital filter is non-recursive, its *impulse response* is of finite duration. On the other hand, in recursive digital filters, the impulse response is usually, but not always, of infinite duration, but if the impulse response is of infinite duration, then the digital filter is always recursive. Alternatively, digital filters can be classified as:

- 1) Infinite Impulse Response Filters (IIR Filters)
- 2) Finite Impulse Response Filters (FIR Filters)

5.4.1 IIR FILTERS

The phase response if IIR filters is non-linear and their stability cannot always be guaranteed. The effects of using a limited number of bits to implement IIR Filters, such as roundoff noise and coefficient quantization errors are normally severe. IIR filters, however, require relatively less coefficients for sharp cut-offs. Moreover, equivalent IIR filters can be easily realized by transforming Analog Filters with similar specifications [16].

5.4.2 FIR FILTERS

FIR Filters can have exactly linear phase response. The implication of this is that no phase distortion is introduced into the signal by the filter. Non-recursive FIR filters are always stable and the effects of roundoff noise and coefficient quantization errors are normally less severe [16].

5.5 DIGITAL FILTER DESIGN FLOW

The design of digital filters comprises five general steps as follows:

- 1) Tolerance Specification
- 2) Coefficient Approximation
- 3) Filter Realization/Synthesis
- 4) Analysis of Finite Word Length Effects
- 5) Filter Implementation & Testing

The characteristics of digital filters are often specified in the frequency domain. For frequency selective filters, such as bandpass or lowpass filters, the specifications are often in the form of tolerance schemes. An example tolerance diagram is as follows:

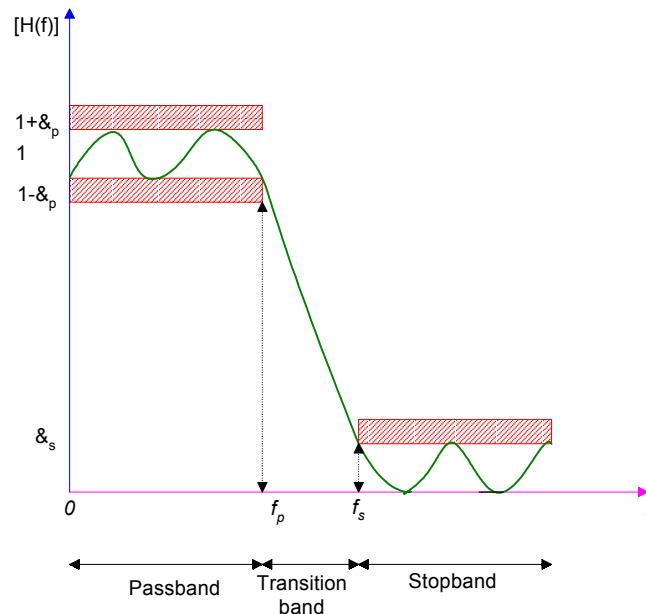


Figure 5.9 Digital Filter Tolerance Scheme [16]

The shaded horizontal lines indicate the tolerance limits. In the passband, the magnitude response has a peak deviation of δ_p and, in the stopband, it has a maximum deviation of δ_s . The width of the transition band determines the sharpness of the filter. The magnitude response decreases monotonically from the passband to the stopband in this region. The following are the key parameters of interest:

- δ_p Passband deviation
- δ_s Stopband deviation
- f_p Passband edge frequency
- f_s Stopband edge frequency

The edge frequencies are often expressed in normalized form, i.e., as a fraction of the sampling frequency (f/F_s). Passband and stopband deviations may be expressed as ordinary numbers or in decibels when they specify the passband ripple and minimum stopband attenuation respectively. Thus the minimum stopband attenuation, A_s , and the peak passband ripple, A_p , in decibels are given as:

$$A_s \text{ (Stopband attenuation)} = -20\log_{10}\delta_p$$

$$A_p \text{ (Passband ripple)} = 20\log_{10}(1+\delta_p) \quad [16]$$

The coefficient approximation step is the process of generating a transfer function that satisfies the desired specifications, which may concern the amplitude, phase, and possibly the time domain response of the filter. The available methods for the solution of the approximation problem can be classified as *direct* or *indirect*. In direct methods, the problem is solved directly in the z domain. In indirect methods, a continuous-time transfer function is first obtained and then converted into a corresponding discrete-time transfer function. Non-recursive digital filters are always designed through direct methods whereas recursive filters can be designed either through direct or indirect methods. Coefficient methods can also be classified as *closed-form* or *iterative*. In closed-form methods, the problem is solved through a small number of design steps using a set of closed-form formulas. In iterative methods, an initial solution is assumed and through the application of optimization methods, a series of progressively improved solutions are obtained until some design criterion is satisfied.

The synthesis of a digital filter is the process of converting the transfer function or some other characterization of the digital filter into a network structure. This process is also referred to as the realization step and the network structure obtained is said to be the realization of the transfer function. As for coefficient approximation methods, realization methods can be classified as direct or indirect. In direct methods, the transfer function is expressed or transformed in some form that allows the identification of an interconnection of elemental digital filter subnetworks. In indirect methods, an analog filter network is converted into a topologically related digital filter network. The best realizations are those that require the minimum number of unit delays, adders, and multipliers, and which are not seriously affected by the use of finite precision arithmetic in the implementation.

During the coefficient approximation step the coefficients of the filter transfer function are determined to a high degree of precision. In practice, however, digital hardware have a finite precision which depends upon: the length of the registers to store binary numbers, the type of the number system used (e.g., signed-magnitude, two's complement), the type of arithmetic used (e.g., fixed-point or floating-point), etc. Consequently, the filter coefficients must be quantized (e.g., rounded or truncated) before they can be stored in registers. When the transfer function coefficients are quantized, errors are introduced in the amplitude and phase responses of the filter. In extreme cases, the required specifications can actually be violated. Similarly, signals to be processed, as well as the internal signals of a digital filter (e.g., products generated by multipliers), must be quantized. Since errors introduced by the quantization of signals are actually sources of noise, they can have a dramatic effect on the performance of a digital filter.

The implementation of a digital filter can assume two forms: *software* or *hardware*. In the first case, implementation involves the simulation of the filter network on a general-purpose digital computer, or a DSP chip. In the second case, it involves the conversion of the filter network into a dedicated piece of hardware. The choice of implementation is usually critically dependent on the application at hand. In *non-real-time* applications where a record of the data to be processed is available, a software implementation may be entirely satisfactory. In *real-time* applications, however, where data must be processed at a very high rate (e.g., in communication systems) a hardware implementation is mandatory. Often the best engineering solution might be partially in terms of software and partially in terms of hardware, since software and hardware are highly exchangeable [15].

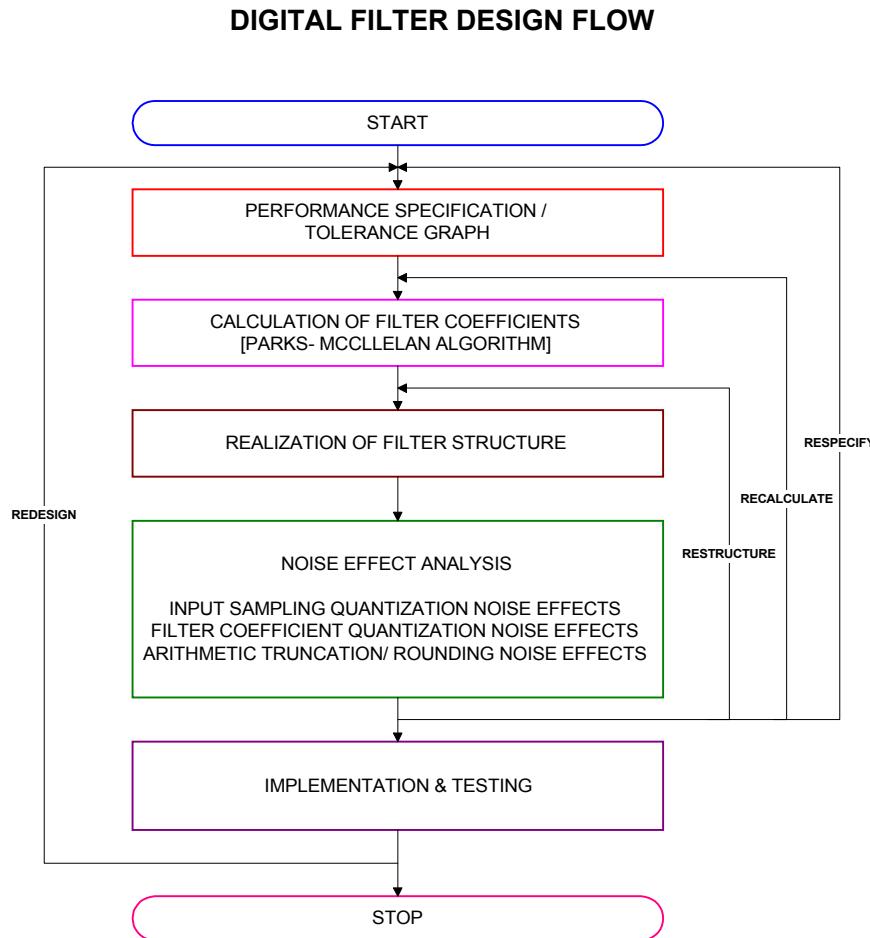


Figure 5.10 Digital Filter Design Flow

5.6 DIGITAL FILTER DESIGN ALGORITHMS

Commonly used techniques for the design of IIR filters are based on transformations of continuous-time IIR systems into discrete-time IIR systems. This is partly because continuous-time filter design was a highly advanced field before discrete-time filters were of interest and partly because of the difficulty of implementing a non-iterative direct design method for IIR filters [12].

5.6.1 BILINEAR TRANSFORMATION METHOD

This is by far the most important method of obtaining IIR filter coefficients. In this method, *Bilinear Transformation* is applied to convert an analog filter transfer function, $H(s)$, into an equivalent digital filter transfer function, $H(z)$ as shown below:

$$s = k \frac{z - 1}{z + 1}, k = 1 \quad \text{or} \quad \frac{2}{T}$$

The above transformation maps the analog transfer function, $H(s)$, from the s-plane into the discrete transfer function, $H(z)$, in the z-plane [16].

5.6.2 IMPULSE INVARIANT METHOD

In this method, starting with a suitable analog transfer function, $H(s)$, the filter impulse response $h(t)$, is obtained using the Laplace Transform. The $h(t)$ so obtained is suitably sampled to produce $h(kT)$, where T is the sampling interval.

5.6.3 POLE-ZERO PLACEMENT METHOD

When a zero is placed at a given point on the z -plane, the frequency response will be zero at the corresponding point. A pole, on the other hand, produces a peak at the corresponding frequency point. Poles that are close to the unit circle give rise to large peaks, whereas zeros close to or on the unit circle produce troughs or minima. Thus, by strategically placing the poles and the zeros on the z -plane, simple lowpass, or other frequency-selective filters can be obtained. For the coefficients of the filter to be real, the poles and the zeros must either be real (i.e., lie on the positive or the negative real axis) or occur in complex conjugate pairs.

In contrast to IIR filters, FIR filters are almost entirely restricted to discrete-time implementations. Consequently, the design techniques for FIR filters are based on directly approximating the specified frequency response of the discrete-time system. Furthermore, most techniques for approximating the magnitude response of an FIR system assume linear phase constraint avoiding the complications involved in the direct implementation of IIR systems [8].

An FIR filter is characterized by the following equations:

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$

$$H(z) = \sum_{k=0}^{N-1} h(k)z^{-k}$$

where: $h(k)$, $k=0, 1, 2, \dots, N-1$ are the impulse response coefficients of the FIR filter,

$H(z)$ is the transfer function of the FIR filter,

N is the FIR filter length.

The first equation is a time-domain, non-recursive difference equation of an FIR filter. The second equation is the transfer function of an FIR filter.

The sole objective of most FIR filter coefficient calculation/approximation methods is to obtain values of $h(k)$ such that the resulting filter meets the design specifications, such as amplitude-frequency response and throughput requirements. Several methods are available for obtaining $h(k)$. The following three methods are the most common, however, as they all can lead to linear phase FIR filters.

5.6.4 WINDOW METHOD

The simplest method of FIR filter design is called the window method. The ideal transfer function of a non-recursive digital filter is:

$$H_{ideal}(z) = \sum_{k=-\infty}^{+\infty} h_{ideal}(k)z^{-k}$$

The corresponding ideal frequency response is:

$$H_{ideal}(z) = \sum_{k=-\infty}^{+\infty} h_{ideal}(k)z^{-k} \Big|_{z=e^{j\omega T}}$$

$$H_{ideal}(e^{j\omega T}) = \sum_{k=-\infty}^{+\infty} h_{ideal}(k) e^{-j\omega kT}$$

The ideal frequency response $|H_{ideal}(e^{j\omega T})| = H_{ideal}(\omega)$ and the corresponding ideal impulse response $h_{ideal}(k)$ are related by the inverse Fourier transform:

$$h_{ideal}(k) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_{ideal}(\omega) e^{j\omega k} d\omega$$

For an FIR filter, the ideal impulse response has to be truncated by setting $h_{ideal}(k)=0$ for $k>N$. However, this introduces undesirable ripples and overshoots called the *Gibb's phenomenon*. Direct truncation of $h_{ideal}(k)$ is equivalent to multiplying $h_{ideal}(k)$ by a rectangular window of the form:

$$w(k) = \begin{cases} 1 & k = 0, 1, \dots, N-1 \\ 0 & \text{elsewhere} \end{cases}$$

In the frequency domain this is equivalent to convolving $H_{ideal}(\omega)$ and $W(\omega)$, where $W(\omega)$ is the Fourier transform of $w(k)$. As $W(\omega)$ has the $(\sin x)/x$ shape, truncation of $h_{ideal}(k)$ leads to overshoots and ripples in the frequency response.

A practical approach is to multiply $h_{ideal}(k)$, by a suitable window function, $w(k)$, whose duration is finite. This yields an impulse response that decays smoothly towards zero, however, the transition width is wider than for the rectangular window. The transition width of the filter is determined by the width of the *main lobe* of the window. The *side lobes* produce ripples in both the passband and the stopband.

Several window functions have been proposed. Some of the most common window functions are:

Hamming

Hanning

Blackman

Bartlett/Triangular

Kaiser

Chebychev

Lanczos

Tukey [14]

The Kaiser window was used to design most of the filters in the system.

5.6.5 OPTIMAL METHOD

Inherent in the process of calculating suitable filter coefficients in the window method is the problem of finding a suitable approximation to a specified or ideal frequency response. The peak ripple of filters designed by the window method occurs near the band edges, and decreases away from the band edges. If the ripples were distributed more evenly over the passband and the stopband, a better approximation of the desired frequency response can be achieved. The optimal method is based on the concept of equiripple passband and stopband.

The difference between the ideal and the practical filter response can be represented by an error function:

$$E(\omega) = W(\omega)[H_{ideal}(\omega) - H(\omega)]$$

where: $H_{ideal}(\omega)$ is the ideal filter response and $W(\omega)$ is a weighting function that allows the relative error of approximation between different frequency bands (passbands, stopbands, transitionbands) to be defined.

In the optimal method, the objective is to determine the filter coefficients, $h(k)$, such that the value of the maximum weighted error, $|E(\omega)|$, is minimized in the passband and the stopband. Mathematically, this may be expressed as:

$$\min[\max|E(\omega)|]$$

over the passbands and the stopbands. It can be shown that when $\max|E(\omega)|$ is minimized the resulting filter response will have equiripple passbands and stopbands, with the ripple alternating in sign between two equal-amplitude levels. For a given set of filter specifications, the location of the extremal (the maxima and the minima are known as the extrema) frequencies, apart from those at the band edges are not known a priori. Thus the main problem in the optimal method is to find the locations of the extremal frequencies. A technique employing the *Remez Exchange Algorithm* to find the extremal frequencies is used. Knowing the locations if the extremal frequencies, it is a simple matter to work out the actual frequency response and hence the impulse response of an FIR filter. For a given set of specifications (i.e., passband edge frequencies, N , and the ratio between the passband and the stopband ripples) the optimal method involves the following key steps:

- 1) use the *Remez Exchange Algorithm* to find the optimum set of extremal frequencies.
- 2) determine the frequency response using the extremal frequencies
- 3) obtain the impulse response coefficients

The heart of the optimal method is the first step where an iterative process is used to determine the extremal frequencies of the filter whose amplitude-frequency response satisfies the optimality condition. This step relies on the alternation theorem which specifies the number of extremal frequencies that can exist for a given value of N [16].

The Remez Exchange Algorithm was used to model the pre-modulation Gaussian Lowpass filter in the modulator section of the system model.

5.6.6 FREQUENCY SAMPLING METHOD

The frequency sampling method yields non-recursive FIR filters for both standard frequency-selective filters (lowpass, bandpass, highpass) as well as filters with arbitrary frequency response. A unique attraction of the frequency sampling method is that it also allows recursive implementation of FIR filters, leading to computationally efficient filters. With some restrictions, recursive FIR filters with integer coefficients may also be designed which is convenient when implementing only primitive arithmetic operations as in systems with standard microprocessors [16].

SUMMARY

This chapter outlined the architecture-level design of the GFSK demodulator. The convolver-based architecture was selected for the demodulator where the convolvers were implemented as bandpass filters using FIR filter banks. The envelope detector was implemented with a full-wave rectifier followed by a low-pass filter.

6

SYSTEM VALIDATION

This chapter discusses the validation of the designed architecture through various modeling and simulation phases and extraction of the design parameters for the system blocks through architectural exploration and iterative performance optimization.

As a first validation step, the designed architecture was validated by constructing a floating-point system model.

6.1 FLOATING POINT MODEL VALIDATION

6.1.1 FLOATING POINT ARITHMETIC

In floating-point arithmetic, a number N is expressed as:

$$N = M \times 2^e$$

where e = an integer and

$$\frac{1}{2} \leq M < 1$$

M and e are referred to as the *mantissa* and the *exponent*, respectively. Negative numbers are handled in the same way as the fixed-point arithmetic.

Floating-point addition is carried out by shifting the mantissa of the smaller number to the right and increasing the exponent until the exponents of the two numbers are equal. The mantissas are then added to form the sum, which is subsequently put back into the normalized representation. Multiplication is accomplished by multiplying mantissas, adding exponents, and then readjusting the product.

Floating-point arithmetic leads to increased cost of hardware and to reduced processing speed. The reason is that hardware is in a sense duplicated since both the mantissa and the exponent have to be manipulated. For software, non-real-time implementations on general purpose digital computers, floating-point arithmetic is always preferred since neither the cost of hardware nor the processing speed is a significant factor [15].

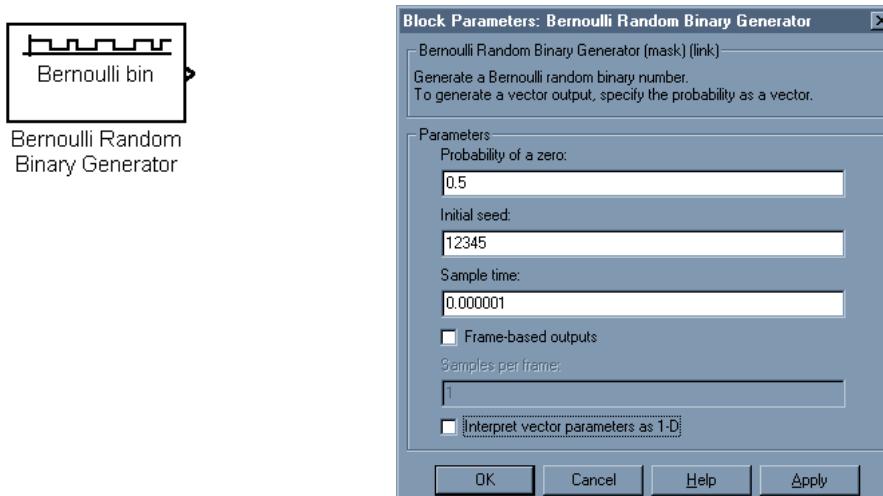
6.1.2 FLOATING-POINT SYSTEM MODELING

MODULATOR MODELING

As an initial step, the GFSK modulator had to be modeled to generate the necessary modulation signal for the demodulator.

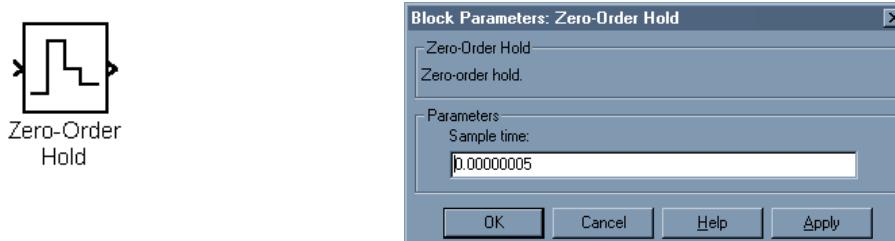
Transmit Bit Stream Generator

The data to be transmitted was modeled as a stochastic Bernoulli process with equal probability of occurrence of a ‘1’ and a ‘0’. For the generation of the sequence of binary numbers in the range [0,1], an initial *seed number* was provided to the random number generation algorithm. The transmit data rate specified by the Bluetooth specification is 1 Mbit/sec that corresponds to a sample time of $1/10^6$.



Oversampler

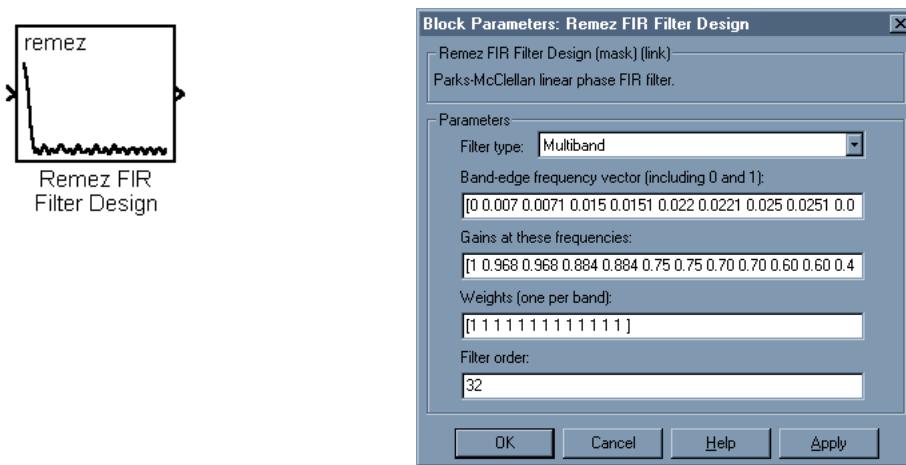
The binary data signal generated by the Bernoulli Random Binary Generator block was sampled at an arbitrarily high rate (to minimize the quantization noise to a reasonable value) of 20Msamples/sec for subsequent Gaussian filtering.



Gaussian Filter

The Gaussian filtering was performed to smooth out the sharp signal transitions by giving them approximately half sinusoidal shape for the reasons explained in chapter 4. This introduces an undesirable DC offset in the Gaussian filtered signal in case of a long chain of '1's in the input signal as well as increased Inter-Symbol Interference (ISI). The Bluetooth specification limits the occurrence of a long chain of '1's through data whitening or scrambling that is performed prior to Gaussian filtering.

The Gaussian Filter was implemented using the multiband filter design technique exploiting the Parks-McClellan algorithm with the Remez Exchange Algorithm explained in chapter 5. The data for the filter band edge frequencies was extracted from slicing a Gaussian curve, plotted in MATLAB™, into narrow bands. For more details please refer to Appendix-B & C.

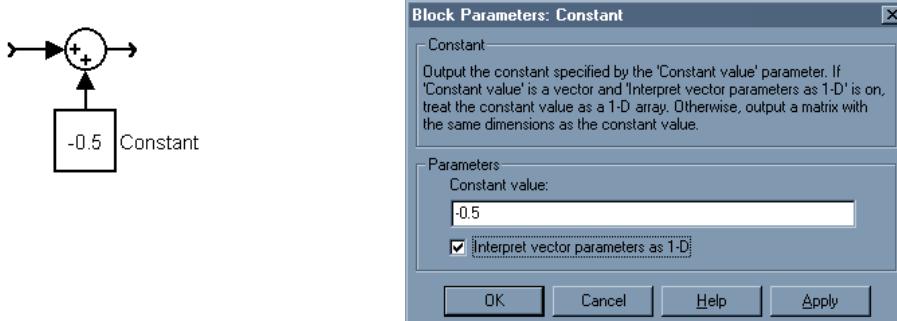


Polar Signaling

Polar signaling was used before FM modulation because the data signal is analog in nature after Gaussian filtering, therefore, FSK modulation cannot be applied. Moreover, Polar signaling removes the condition of :

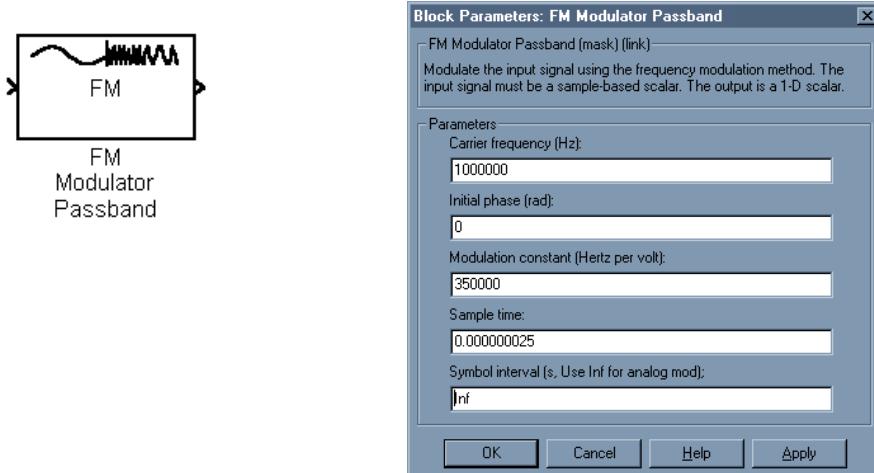
$$\Delta f T_b > \text{Symbol Rate}$$

which is required in case of FSK employing Orthogonal signaling. Polar signal was achieved by balancing the Gaussian filtered signal across the horizontal axis by the addition of a constant DC level of -0.5.



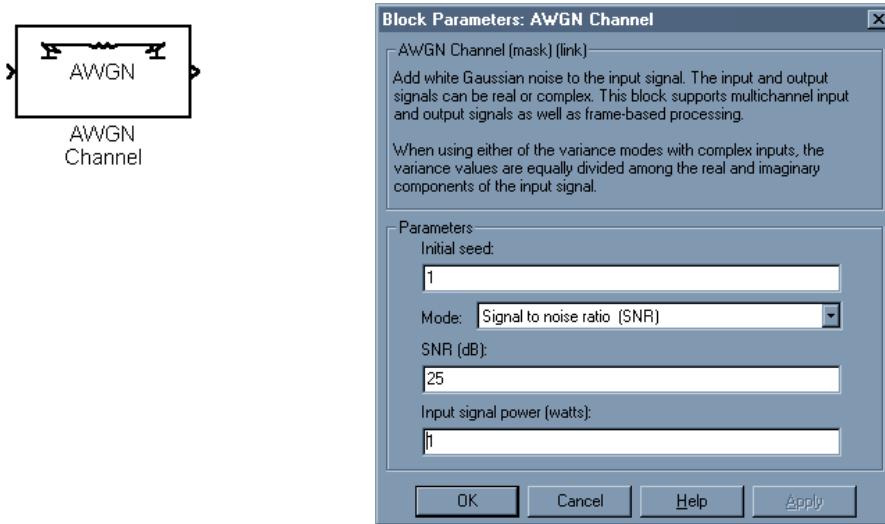
FM Modulation

The FM modulation uses a carrier frequency of 1MHz (the IF on the receiver side), a frequency separation of 350 KHz (Bluetooth specification), and a sampling rate of 40 Msamples/sec to limit the quantization noise for achieving the Bit Error Rate (BER) specified in the Bluetooth™ specification.



Channel Model

The Bluetooth™ specification does not specify a channel model. The Additive White Gaussian Noise (AWGN) channel model was used with statistically independent Gaussian noise samples corrupting data samples free of Inter-Symbol Interference (ISI). An SNR of 25 dB was used based upon calculations performed by the Analog Front-end designers. Other channel impairments like channel fading, Doppler shift, etc., are not modeled.



DEMODULATOR MODELING

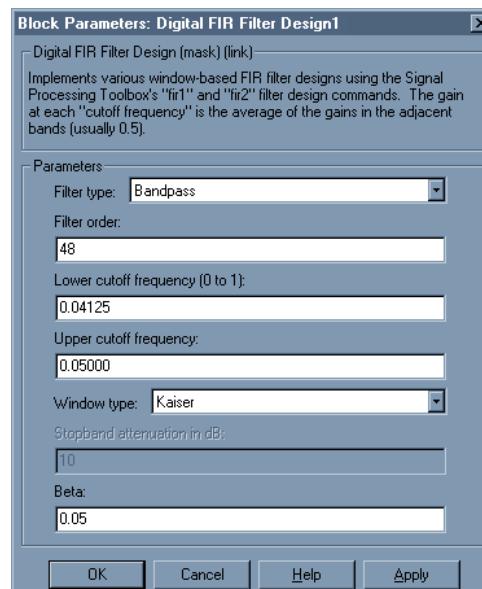
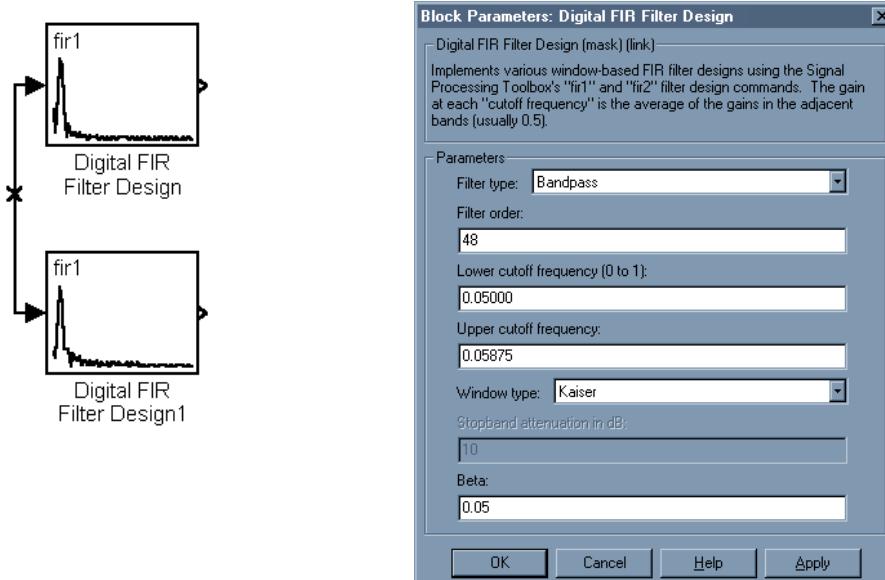
The demodulator model accepts the GFSK modulated signal at an IF of 1MHz, sampled at 40 Msamples/sec, and corrupted by AWGN.

Demodulator Bandpass Filters

The two digital FIR bandpass filters in the demodulator stage, as explained in chapters 4 and 5, split the GFSK signal into two ASK modulated components. The upper bandpass filter has an upper cutoff frequency of $1\text{MHz}+350\text{KHz}/2 = 1175\text{KHz}$ normalized to $1175\text{KHz}/20\text{MHz} = 0.05875$ (where 20MHz is half the sampling frequency) and a lower cutoff frequency of 1MHz normalized to $1\text{MHz}/20\text{MHz} = 0.05000$.

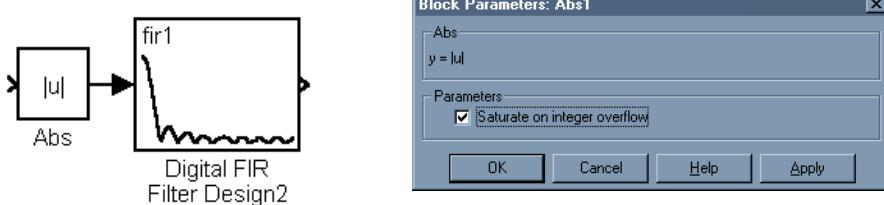
Similarly the lower bandpass filter has an upper cutoff frequency of 1MHz normalized to $1\text{MHz}/20\text{MHz} = 0.05000$ and a lower cutoff frequency of $1\text{MHz}-175\text{KHz} = 825\text{KHz}$ normalized to $825\text{KHz}/20\text{MHz} = 0.04125$.

Both bandpass filters yield optimum performance when designed using the Kaiser window algorithm with $\beta=0.05$ and a filter order of 48.



Envelope Detection

As described in chapter 4, envelope detection involves full-wave rectification followed by lowpass filtering. The full-wave rectifier is realized as an absolute-value function clipping the signal if the magnitude of the input signal exceeds the maximum value representable within 16-bits. The lowpass filter has a cutoff frequency equal to the lowest cutoff frequency of the two bandpass filters, i.e., 825KHz normalized to $825\text{KHz}/20\text{MHz} = 0.04125$ selected through trial & error. The lowpass filter also uses the Kaiser window algorithm with $\beta = 0.05$ and has an order of 16. The envelope detectors in both the branches of the demodulator are exactly symmetric.



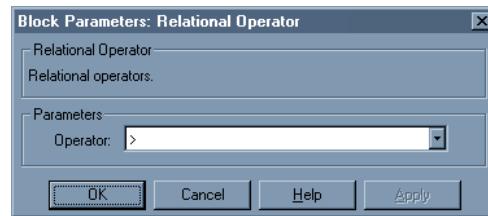
Combiner/Adder

The combiner/adder simply combines or adds together the two signal components in the two branches of the demodulator after envelope detection and inverting one of the signal components before addition/combination.

Threshold Detection

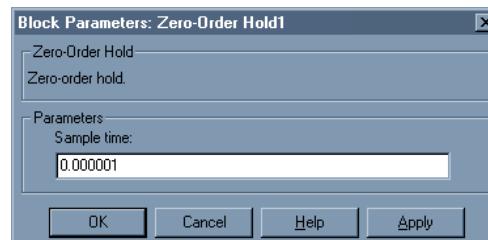
The threshold detector or the magnitude comparator rehabilitates the combined signal. It just acts like a Schmitt Trigger to recondition the received signal.





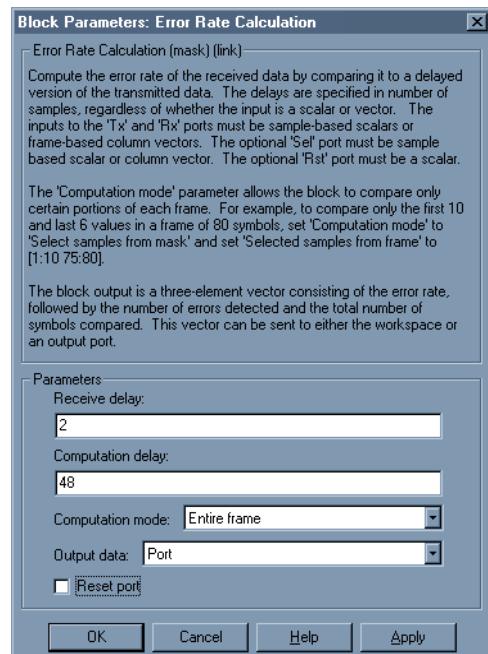
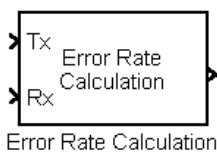
Decimator

The decimator brings the sampling rate of the demodulated signal down to the data transmit rate, i.e., 1Msample/sec.



Bit Error Rate Calculator

The bit error rate calculator calculates the ratio of the number of received bits in error to the total number of bits transmitted by comparing the received data with the transmitted data padded with necessary delays.



The complete floating-point model is illustrated below and the simulation waveforms at different points are attached as Appendix-B. The Bit Error Rate (BER) is zero.

FLOATING-POINT SIMULATION MODEL OF THE GFSK MODULATOR/DEMODULATOR

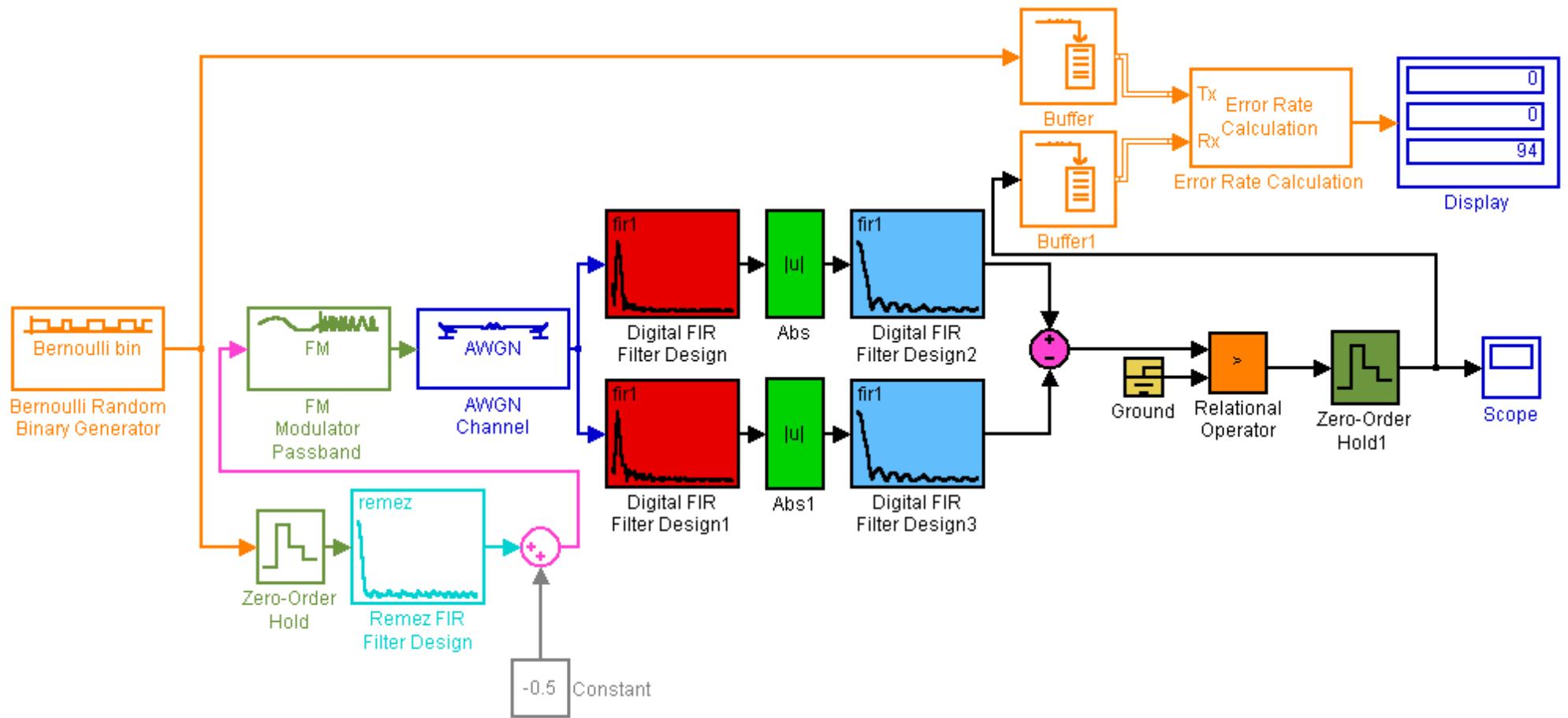


Figure 6.1 Floating Point System Model

To model the effects of quantization or finite word length as described in chapter 5, and to ascertain the effects of limited precision supported by fixed-length arithmetic, the floating-point model was transformed into a fixed-point model.

6.2 FIXED-POINT MODEL VALIDATION

6.2.1 FIXED POINT ARITHMETIC

In fixed-point arithmetic, the numbers are usually assumed to be proper fractions. Integers and mixed numbers are avoided because:

- 1) the number of bits representing an integer cannot be reduced by rounding or truncation without compromising the accuracy of the number value.
- 2) mixed-numbers are more difficult to multiply.

For these reasons, the binary point is usually set between the first and the second bit positions in the register. The first position is reserved for the sign of the number.

Depending upon the representation of negative numbers, fixed-point arithmetic can assume three forms:

- 1) Signed Magnitude
- 2) 1's Complement
- 3) 2's Complement

In the *signed magnitude* arithmetic a fractional number:

$$N = \pm 0.b_{-1}b_{-2}\dots b_{-m}$$

is represented as:

$$N_{sm} = \begin{cases} 0.b_{-1}b_{-2}\dots b_{-m} & \text{for } N \geq 0 \\ 1.b_{-1}b_{-2}\dots b_{-m} & \text{for } N \leq 0 \end{cases}$$

The most significant bit is the *sign* bit.

The *1's complement* representation of a number N is defined as:

$$N_{sm} = \begin{cases} N & \text{for } N \geq 0 \\ 2 - 2^L - |N| & \text{for } N \leq 0 \end{cases}$$

where L , referred to as the *wordlength*, is the number of bit locations in the register to the right of the binary point. The binary form of $2 - 2^L$ is a string of 1's filling the $L+1$ locations of the register. Thus the 1's complement of a negative number can be deduced by representing the number by $L+1$ bits, including 0's if necessary, and then complementing (changing 0's into 1's and 1's into 0's) all bits.

The *2's complement* of a negative number can be formed by adding 1 at the least significant position of the 1's complement. Similarly, a negative number can be recovered from its 2's complement by complementing and then adding 1 at the least significant position.

The merits and demerits of the three types of arithmetic can be envisaged by examining how arithmetic operations are performed in each case.

1's complement addition of any two numbers is carried out by simply adding their 1's complements bit by bit. A carry bit at the most significant position, if generated, is added at the least significant position (*end around carry*).

2's complement addition is exactly the same except that a carry bit at the most significant position is ignored.

Signed magnitude addition, on the other hand, is much more complicated as it involves sign checks as well as complementing and end-around carry.

In the 1's or the 2's complement arithmetic, direct multiplication of the complements does not always yield the product, and as a consequence special algorithms must be employed. By contrast, signed-magnitude multiplication is accomplished by simply multiplying the magnitudes of the two numbers bit by bit and then adjusting the sign bit of the product.

An important feature of the 1's and the 2's complement addition is that a sum $S=n_1+n_2+\dots+n_i+\dots$ will always be evaluated correctly, even if overflow does occur in the evaluation of the partial sums.

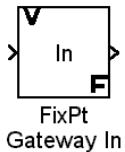
There are two basic disadvantages in a fixed-point arithmetic:

- 1) The range of numbers that can be handled is small
- 2) The percentage error produced by truncation or rounding tends to increase as the magnitude of the number is decreased [17] [13].

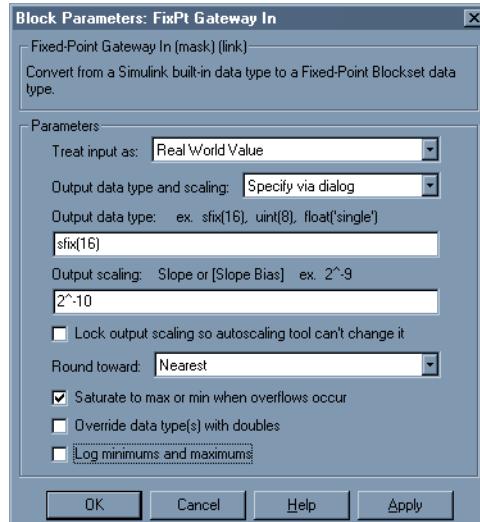
An advantage of fractional fixed-point arithmetic is that parasitic oscillations are more easily suppressed than they are in the floating-point arithmetic. Fixed-point arithmetic also requires less chip area and is much faster than the floating-point arithmetic [15].

6.2.2 FIXED-POINT SYSTEM MODELING

Fixed-Point Gateway In



The Fixed-Point Gateway In block converts the Simulink data type to a Fixed-Point Blockset data type.



To convert the filters in the system from floating-point designs to fixed-point designs, the Filter Design Toolbox was used.

6.2.2.1 FILTER DESIGN TOOLBOX

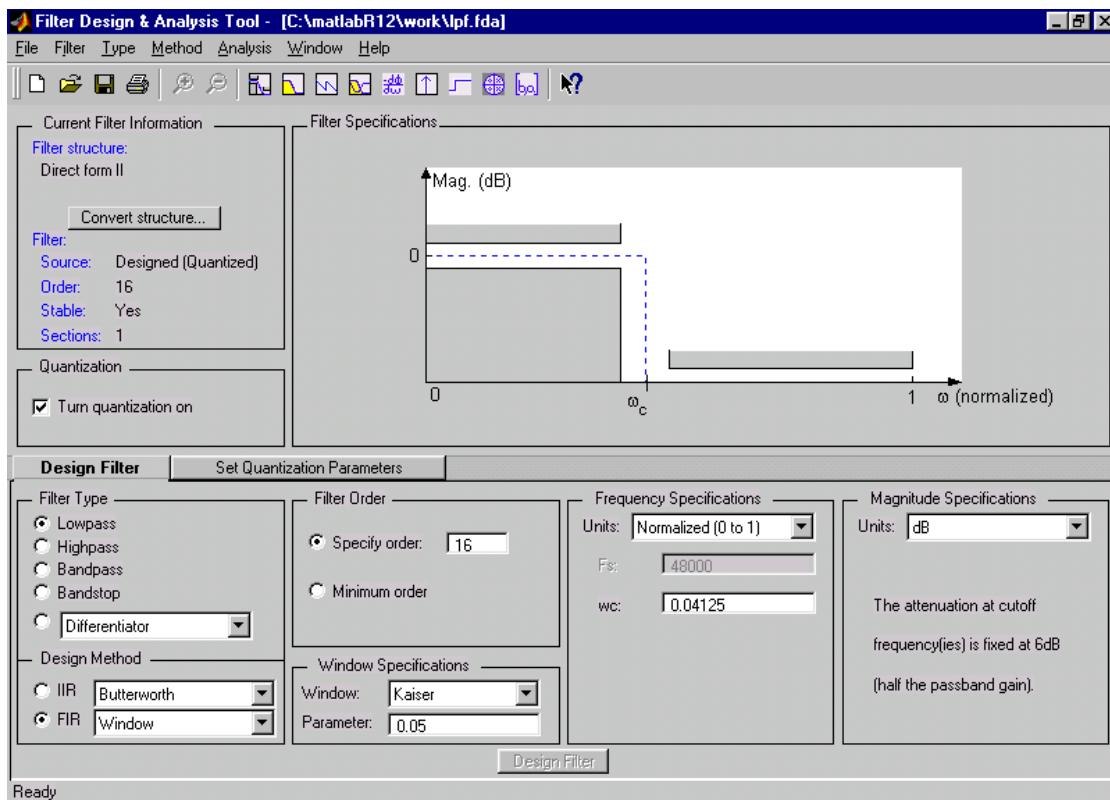
The **Filter Design Toolbox** or the **Quantized Filtering Toolbox** is a collection of tools built on top of the MATLAB™ computing environment and the **Signal Processing Toolbox**. It provides extensive simulation analysis and tools for the fixed-point and custom precision floating-point filters. The Quantized Filtering Toolbox supports the simulation and bit-true analysis of these filters in a wide range of precision and it mainly consists of the Filter Design & Analysis Tool [67].

FILTER DESIGN & ANALYSIS TOOL

The Filter Design & Analysis Tool is a tool in the MATLAB™ Filter Design Toolbox that provides the functions needed to develop filters that meet the needs of fixed-point algorithms and electronic systems. In addition to offering tools for analyzing the effects of quantization on filter performance and signal processing performance, the FDA Tool offers filter structures to develop prototype filter designs. With structures ranging from Finite Impulse Response (FIR) filters to Infinite Impulse Response (IIR) filters, one can investigate alternative fixed-point realizations of filters that might meet the performance goals.

Algorithmic filters are often designed using mathematical methods based on infinite precision and range assuming that the filter coefficients and the filter states are stored with infinite precision and range. However, real-time filters for embedded real-time DSP applications are designed to represent numbers using a finite format. The Quantized Filtering Toolbox can help design these filters by offering analysis tools for fixed-point and custom precision floating-point filters [67].

The Graphical User Interface of the FDA Tool is quite intuitive to use and is illustrated below:



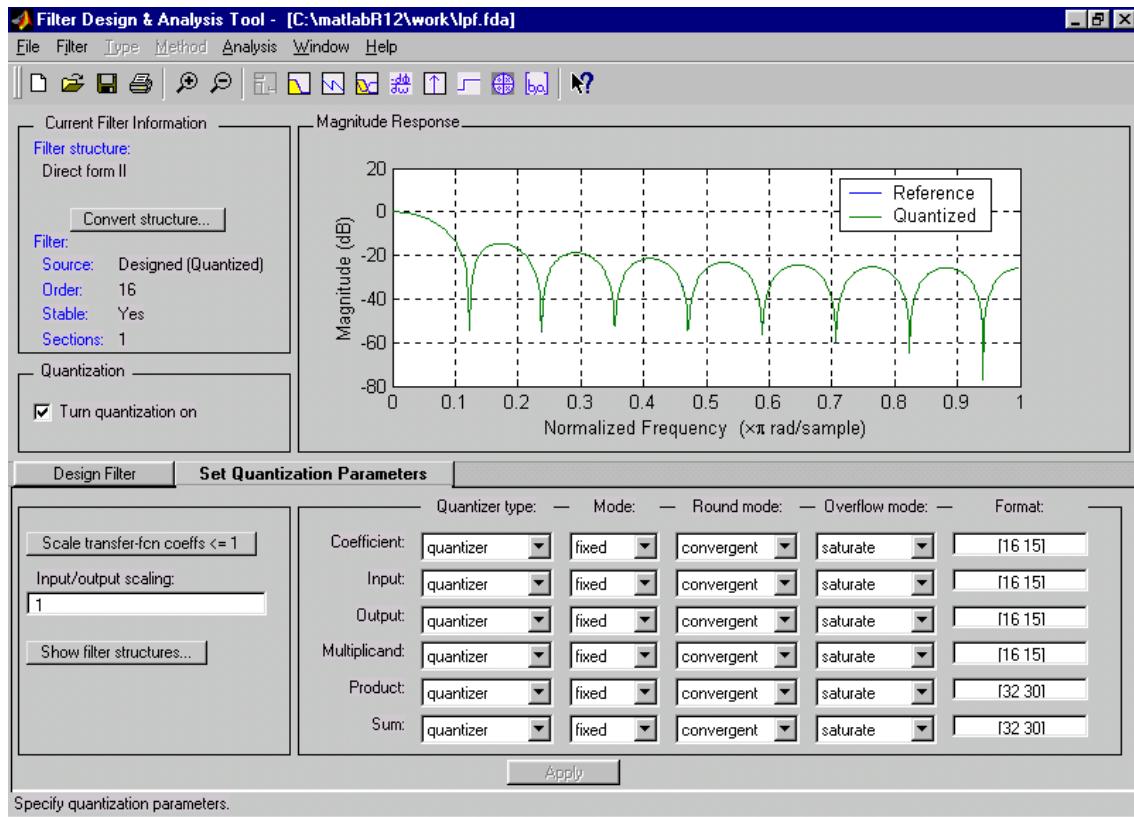
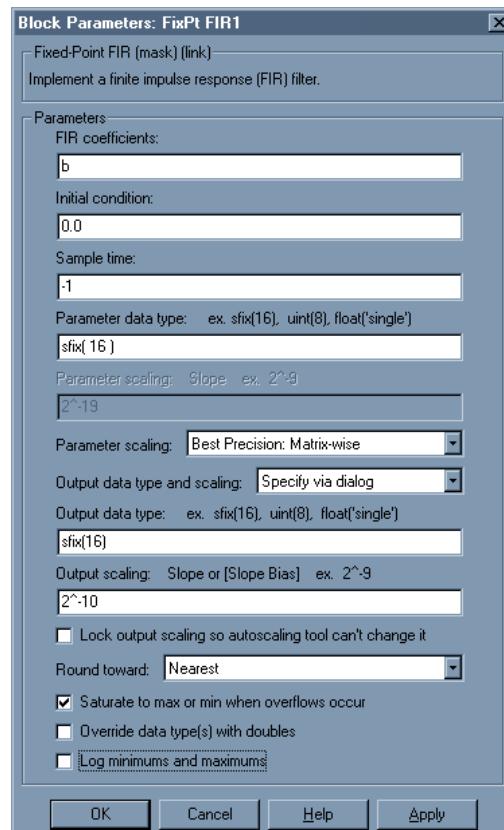
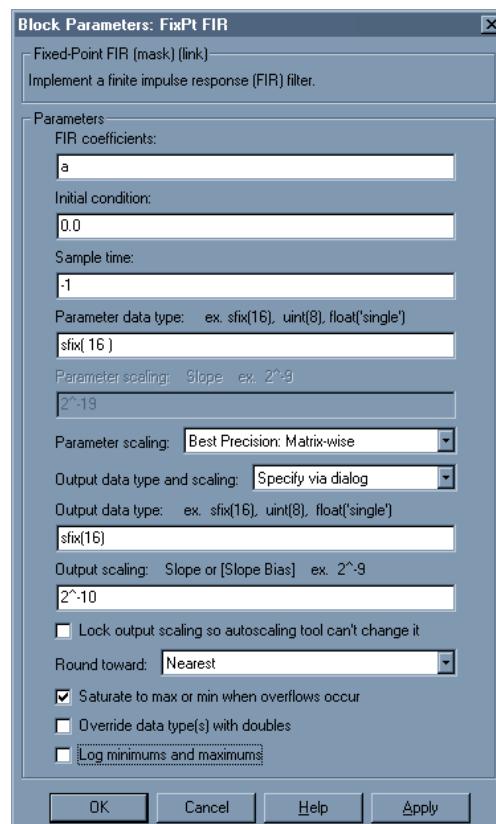
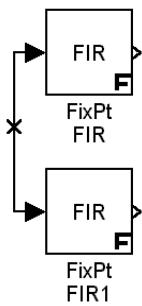


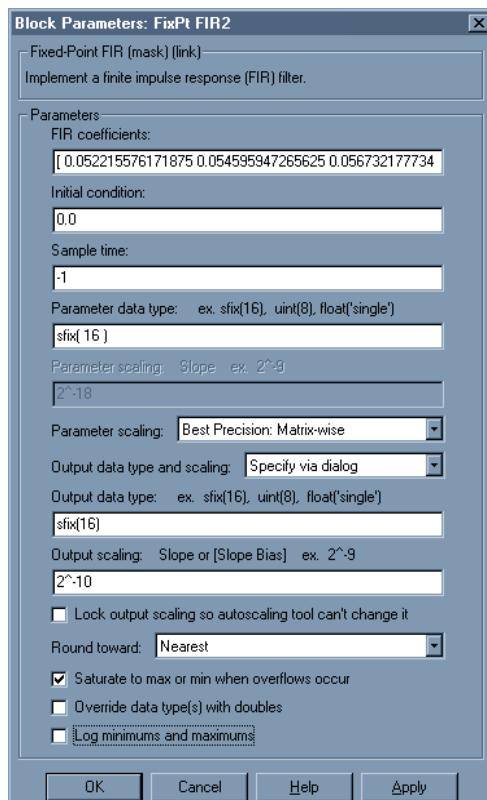
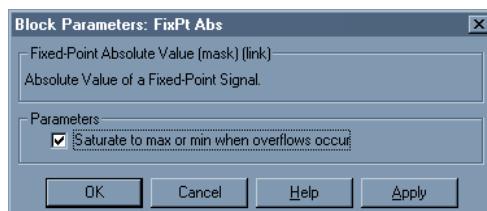
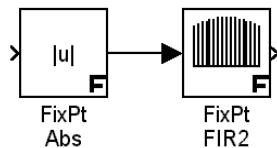
Figure 6.2 Settings of the quantization parameters for the Filter Design & Analysis Tool

The details of the filters designed using the FDA Tool are attached as Appendix-C of this report. The reader is referred to [4] for more detailed discussion about the selection of various options in the FDA design environment. The quantized coefficients for the filters generated by the FDA Tool were plugged into the corresponding fixed-point filter block GUI's in the fixed-point model. The **Direct Form II** coefficient structure was selected for all the filters in the design because of its low sensitivity to coefficient quantization effects.

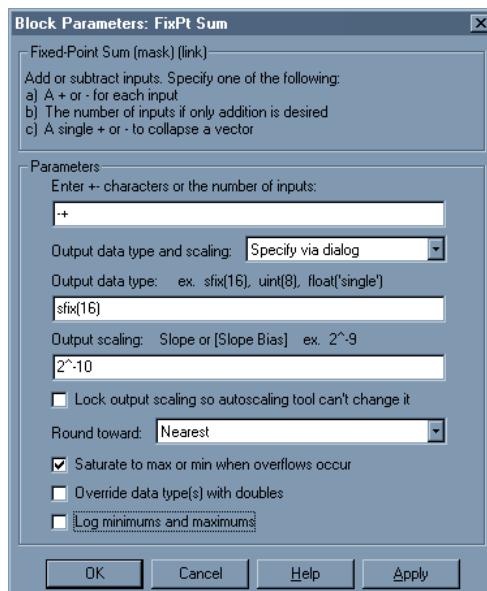
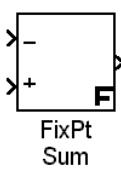
Fixed-Point Bandpass Filters



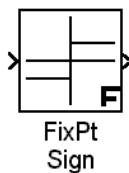
Fixed-Point Envelope Detector



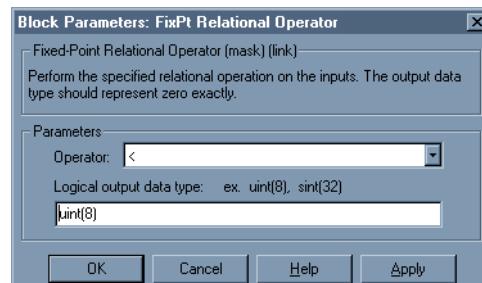
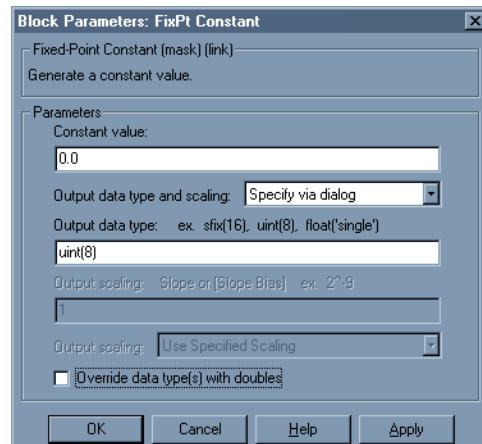
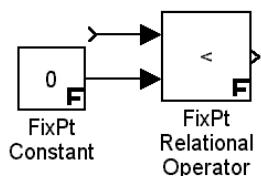
Fixed-Point Summer



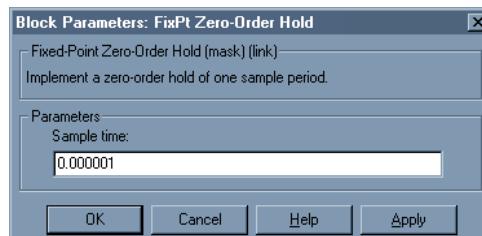
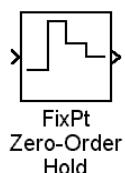
Fixed-Point One-Bit Quantizer



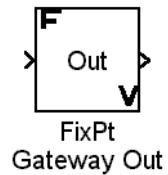
Fixed-Point Threshold Detector



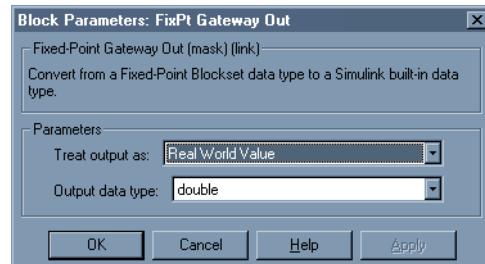
Fixed-Point Decimator



Fixed-Point Gateway Out



FixPt
Gateway Out



The complete fixed-point model is illustrated below and the simulation waveforms at different points are attached as Appendix-B. The Bit Error Rate (BER) is zero.

FIXED-POINT SIMULATION MODEL OF THE GFSK MODULATOR/DEMODULATOR

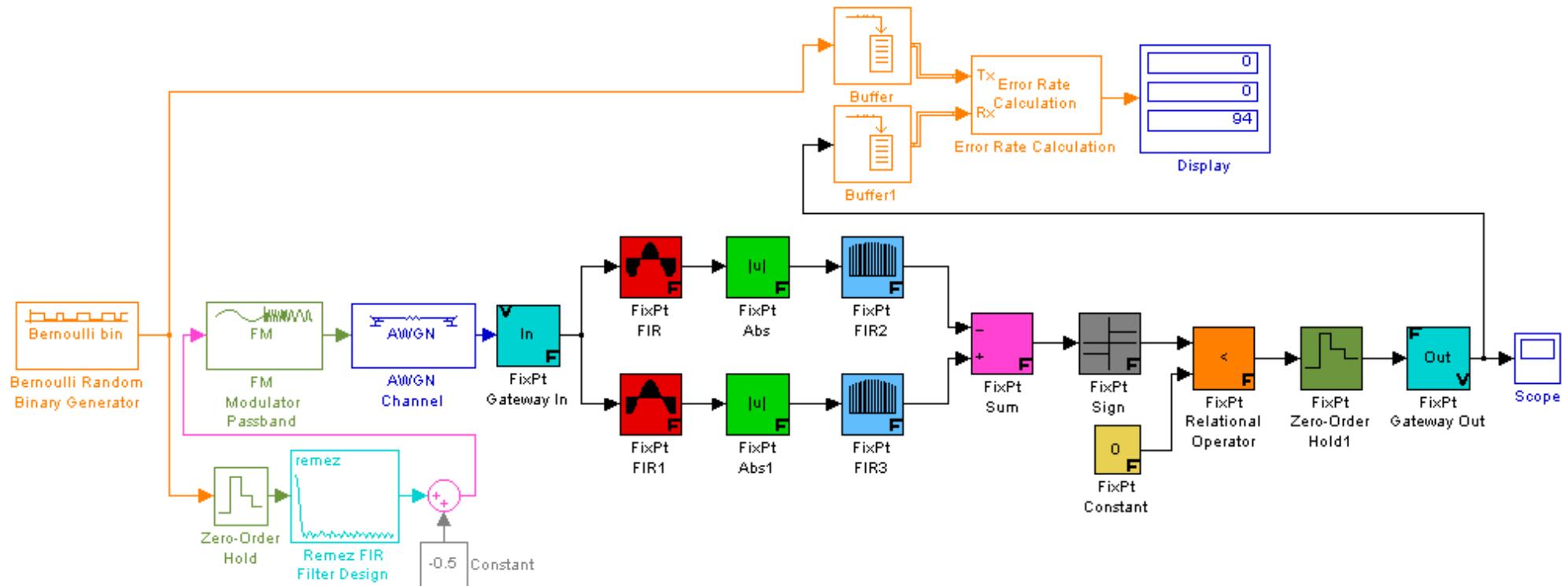


Figure 6.3 Fixed-Point System Model

SUMMARY

This chapter described the system validation phase of the project which consisted of building a set of hierarchical system models which were refined and transformed to bring them closer to system realization and at the same time each model was validated to ascertain its performance versus the system specification and to extract the design parameters for the system blocks.

7

SYSTEM REALIZATION

This chapter discusses the steps and issues of system realization which involved synthesizing the filters used in the design, inserting the synthesized filter structures into the fixed-point system model and validating the system model with realized filter structures.

7.1 DIGITAL FILTER REALIZATION STRUCTURES

Realization structures are essentially the block or signal flow diagram representations of the different theoretically equivalent ways a digital filter transfer function can be arranged. In most of the cases, they consist of an interconnection of unit delay elements, multipliers, and adders.

Two types of realization methods have been proposed, namely, direct and indirect. In direct methods, the transfer function is put in some form that allows the identification of an interconnection of elemental digital filter subnetworks of low order. The most commonly used realization methods of this class are [13-17]:

- 1) Direct / Direct Form I
- 2) Transposed Direct Form I
- 3) Direct Canonic / Direct Form II
- 4) Transposed Direct Form II
- 5) Linear Phase
- 6) Transposed Linear Phase
- 7) Frequency Sampling
- 8) Fast Convolution

- 9) Parallel
- 10) Series / Cascade
- 11) Lattice
- 12) Transposed Lattice
- 13) Ladder
- 14) Systolic
- 15) State-Space

In indirect methods, an analog filter network is converted into a topologically related digital filter network through the application of network theoretic concepts in conjunction with some simple transformations, e.g., wave structures.

Digital filter structures obtained by different methods can differ quite significantly with respect to complexity, number of elements, and their properties. One structure might require a large number of multipliers and yet be relatively insensitive to coefficient quantization errors, and a second structure might be economical in terms of elements but generate parasitic oscillations when signals are quantized, and so on.

Transversal / Direct Form I

The transversal / direct / tapped delay line structure is the most popular FIR structure. The input, $x(k)$, and output of the FIR filter for the direct form structure are related simply by:

$$y(k) = \sum_{k=0}^{N-1} h(k)x(n-k)$$

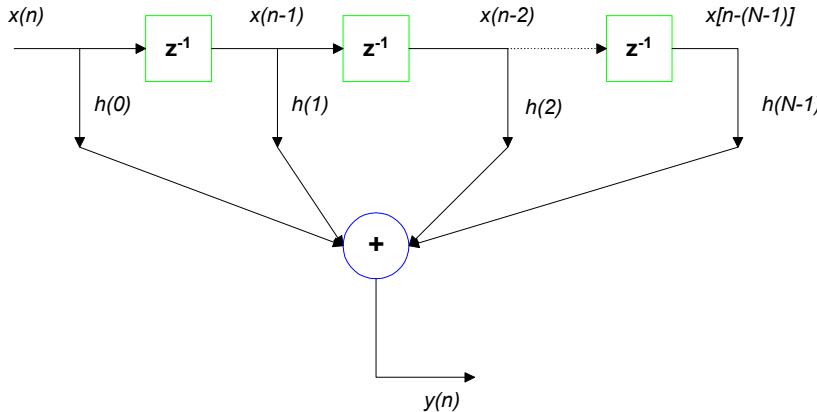


Figure 7.1 Direct Form I Filter Structure

The symbol z^{-1} represents a delay of one sample or unit of time. Thus $x(n-1)$ is $x(n)$ delayed by one sample. In digital implementations, the boxes labeled z^{-1} could represent shift registers. The output sample, $y(n)$, is a weighted sum of the present input, $x(n)$, and $N-1$ previous samples of the input, i.e., $x(n-1)$ to $x(n-N)$. For the transversal structure, the computation of each output sample, $y(n)$, requires:

- $N-1$ memory locations to store the $N-1$ input samples,

- N memory locations to store the N coefficients,
- N multiplications
- $N-1$ additions

Alternative filter structures can be obtained by using the transposition theorem. In essence, a new filter structure is obtained by reversing the direction of all branches in a signal flow graph and changing inputs to outputs and vice versa. This new filter structure has the same transfer function as the original filter but the numerical properties are generally different.

Transposed Direct Form I

The transpose structure is similar to the direct structure, except that the partial sums feed into succeeding stages. This method is more susceptible to roundoff noise than the direct method.

Direct Canonic / Direct Form II

A digital network is said to be canonic if the number of unit delay elements employed is equal to the order of the transfer function.

Linear Phase Structure

A variation of the transversal structure is the linear phase structure which takes advantage of the symmetry or anti-symmetry in the impulse response coefficients for linear phase FIR filters to reduce the computational complexity of the filter implementation.

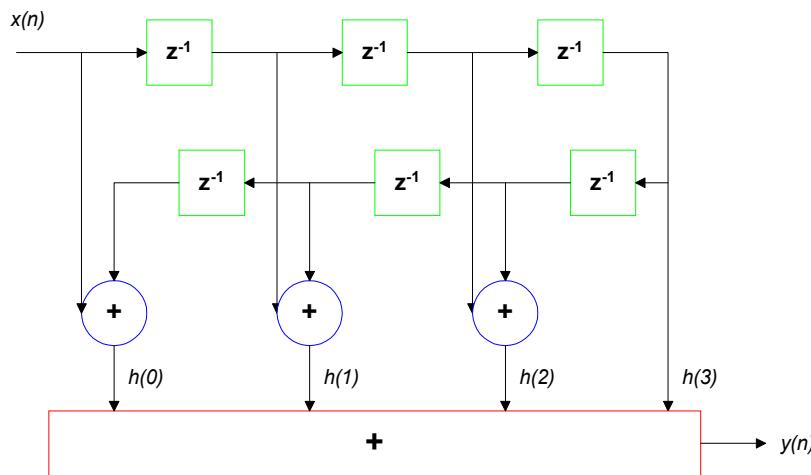


Figure 7.2 A Linear Phase Structure for an FIR Filter with 7 Coefficients

In a linear-phase filter, the coefficients are symmetrical, i.e., $h(n)=\pm h(N-n-1)$. Thus the filter equation can be rewritten to take account of this symmetry with a consequent reduction in both the number of multiplications and additions. The number of multiplications is reduced from N to $N/2$ for N even and to $(N-1)/2$ for N odd. A major drawback is that the group delay for linear phase FIR filters is often too large to be useful in many applications.

Frequency Sampling

In the frequency sampling structure, the filters are characterized by the samples of the desired frequency response, $H(k)$, instead of its impulse response coefficients. For

narrowband filters, most of the frequency samples will be zero, and so the resulting frequency sampling filter will require a smaller number of coefficients and hence multiplications and additions than an equivalent transversal structure. However, the frequency sampling structures suffer from high coefficient sensitivity, low dynamic range, and severe stability problems.

Fast Convolution

The fast convolution method involves performing the convolution operation in the frequency domain. Convolution in the time domain is equivalent to multiplication in the frequency domain. Filtering is performed by first computing the DFTs of $x(n)$ and $h(n)$ (suitably zero padded), multiplying these together and then obtaining their inverse. In practice, techniques known as overlap-add and overlap-save are used in real-time filtering.

Lattice Form

FIR filters that are embedded in adaptive FIR filters are often realized by a lattice structure. A drawback of such a structure is that the number of operations is high since there are two multiplications and two additions for each filter coefficient.

Series / Cascade

High-order IIR filters are often realized as a cascade of several low-order filters in order to reduce the sensitivity to coefficient errors. This approach, in principle, can also be used for FIR filters, but the benefits are offset by a decrease in dynamic signal range. In the cascade realization, the transfer function, $H(z)$, is expressed as the product of second-order and first-order sections.

Parallel

Parallel filter structures, comprising first- and second-order filter sections, can be obtained by expanding the filter transfer function into partial fractions.

7.2 FINITE WORD LENGTH / QUANTIZATION EFFECTS

The effect of finite word length depends on the filter structure, pole-zero configuration, representation of negative numbers, rounding or truncation of products, overflow characteristics, and the input signal. Finite word length gives rise to a large number of phenomena caused by different types of nonlinearities. Some of those phenomena include [17]:

- 1) Aliasing Errors
- 2) Round-off / Product Quantization Errors
- 3) Coefficient Quantization Errors

ALIASING ERRORS

Aliasing errors occur in A/D and D/A converters.

ROUND-OFF/PRODUCT QUANTIZATION

Rounding or truncation of products must be done so that the word length does not increase above the limit. Product Quantization errors can be regarded as white noise sources which give rise to output roundoff noise. However, both rounding and truncation are nonlinear operations that may cause parasitic oscillations.

COEFFICIENT QUANTIZATION ERRORS

The filter coefficients can only be represented with finite precision. Coefficient Quantization errors introduce perturbations in the zeros and poles of the transfer function, which in turn manifest themselves as a static deviation of the filter's frequency response from the ideal frequency response.

It is frequently advantageous to use different word lengths for the filter coefficients and the input samples. The coefficient word length can be chosen to satisfy prescribed frequency-response specifications, whereas the input signal word length can be chosen to satisfy a signal-to-noise ratio specification.

7.3 FILTER REALIZATION/SYNTHESIS

7.3.1 FILTER REALIZATION WIZARD

The **Filter Realization Wizard** is a tool in the MATLAB™ **DSP Blockset** for automatically generating floating-point, block diagram models of digital filters from the filter specifications. The FRW automatically synthesizes digital filter structures with specific architectures. The FRW graphical user interface enables the choice of the filter structure and filter coefficients, the type of data to be filtered (fixed-point or floating-point), and optimization criteria for the filter design. The FRW then builds the specified filter structure composed of Sum, Gain, and Delay blocks that can be saved under the specified name as a MATLAB model [67].

The **Architecture** panel in the FRW GUI allows the option of the following realizations:

Architecture	Parameters
Direct-Form I	Numerator, Denominator
Direct-Form II	Numerator, Denominator
Lattice (AR)	Lattice Coefficients
Lattice (MA)	Lattice Coefficients
Lattice (ARMA)	Lattice Coefficients, Ladder Coefficients
Symmetric FIR	Coefficients

The **Optimization** panel in the FRW GUI allows the choice of optimizing for zero gains and unity gains. A zero-gain optimization removes zero-gain paths from the filter structure, and a unity-gain optimization substitutes a wire (short circuit) for unity gains [67].

The filter coefficients generated by the MATLAB™ **Filter Design & Analysis Tool** were stored in a vector variable, e.g., for the bpf1, by the following command:

```
a = [-0.020996093750000 -0.025817871093750 -0.029907226562500
      -0.033111572265625 -0.035369873046875 -0.036590576171875
      -0.036773681640625 -0.035858154296875 -0.033874511718750
      -0.030944824218750 -0.027069091796875 -0.022399902343750
      -0.017089843750000 -0.011260986328125 -0.005096435546875
      0.001251220703125 0.007537841796875 0.013610839843750]
```

```

0.019317626953125 0.024444580078125 0.028869628906250
0.032440185546875 0.035064697265625 0.036682128906250
0.037231445312500 0.036682128906250 0.035064697265625
0.032440185546875 0.028869628906250 0.024444580078125
0.019317626953125 0.013610839843750 0.007537841796875
0.001251220703125 -0.005096435546875 -0.011260986328125
-0.017089843750000 -0.022399902343750 -0.027069091796875
-0.030944824218750 -0.033874511718750 -0.035858154296875
-0.036773681640625 -0.036590576171875 -0.035369873046875
-0.033111572265625 -0.029907226562500 -0.025817871093750
-0.020996093750000 ]

```

The variable ‘*a*’ was inserted as ‘Numerator’ in the Filter Realization Wizard to synthesize the filter structure.

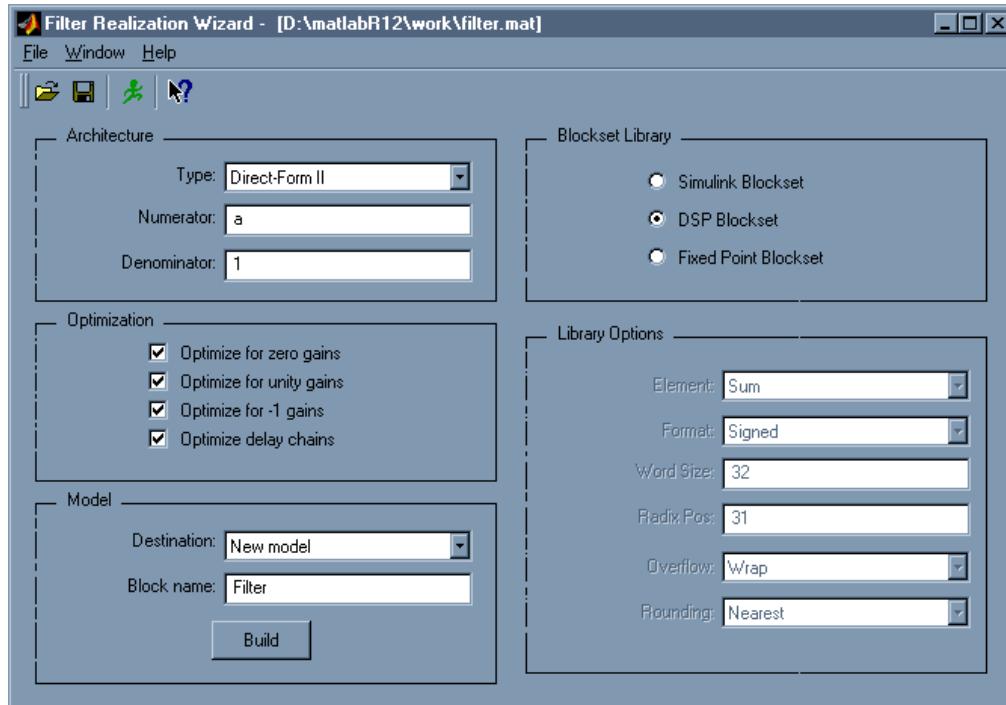


Figure 7.3 Graphical User Interface of the Filter Realization Wizard

The Blockset Library option refers to the selection of the adder, multiplier/gain, and delay components comprising the filter structure from the library components of the Simulink Blockset, DSP Blockset, or the Fixed-Point Blockset. DSP Blockset Library was selected to keep compatibility with the floating-point filter models from the DSP Blockset Library.

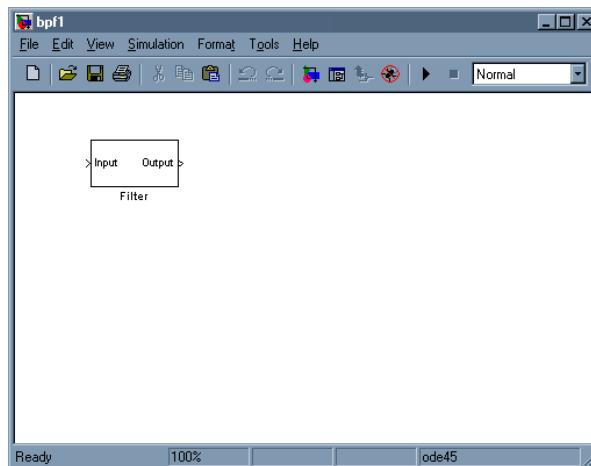


Figure 7.4 The Floating-Point Filter Block Synthesized by the Filter Realization Wizard

The filter structure synthesized by the Filter Realization Wizard is in the form of a floating-point realized filter model that was saved in the work directory and plugged into the floating-point system model to analyze the performance of the system model for different realization structures for the component filter blocks. To ascertain the performance of the fixed-point system model, the floating-point realized filter model was converted to its fixed-point equivalent, e.g., for the `bpf1` by the following command:

```
res = fixpt_convert('bpf1') [ MATLAB™ Fixed-Point Blockset User Manual, page 8-10]
```

the resulting fixed-point realized filter model was saved as `bpf1_fixpt` in the work directory and plugged into the fixed-point system model.

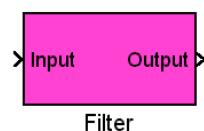
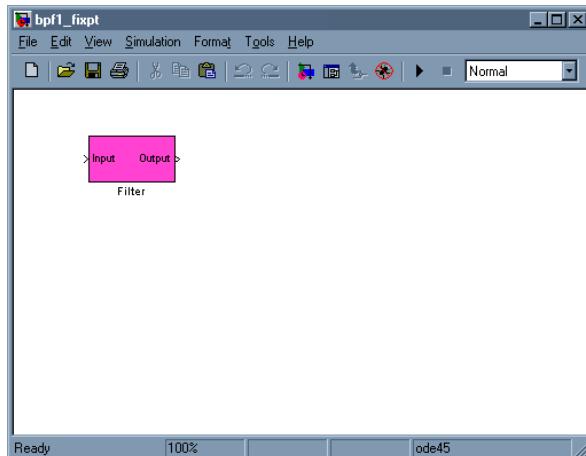


Figure 7.5 The Fixed-Point Filter Block converted from the Floating-Point Filter Block

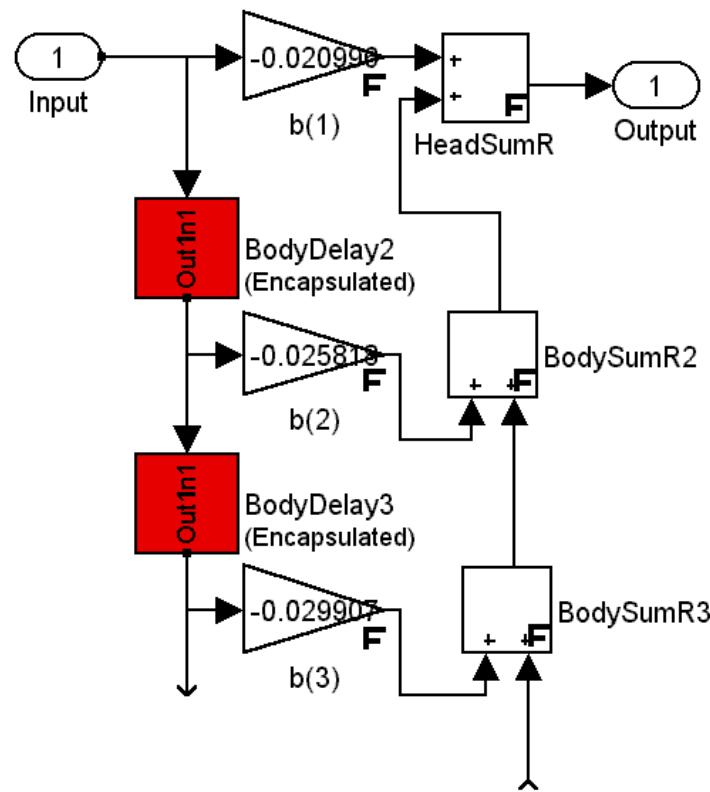


Figure 7.6 Section of the Synthesized Fixed-Point Direct Form II Realization Structure

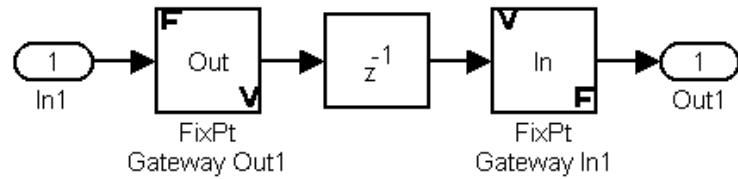


Figure 7.7 The Encapsulated Fixed-Point Delay Block (shown in red in Figure 7.6)

SUMMARY

This chapter described the process of system realization and its validation and optimization through the realized system model.

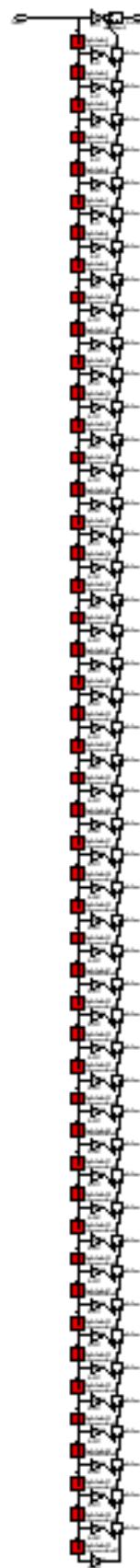


Figure 7.8 The Synthesized 49-Tap Direct Form II Bandpass FIR Filter Structure

REALIZED FIXED-POINT SIMULATION MODEL OF THE GFSK MODULATOR/DEMODULATOR

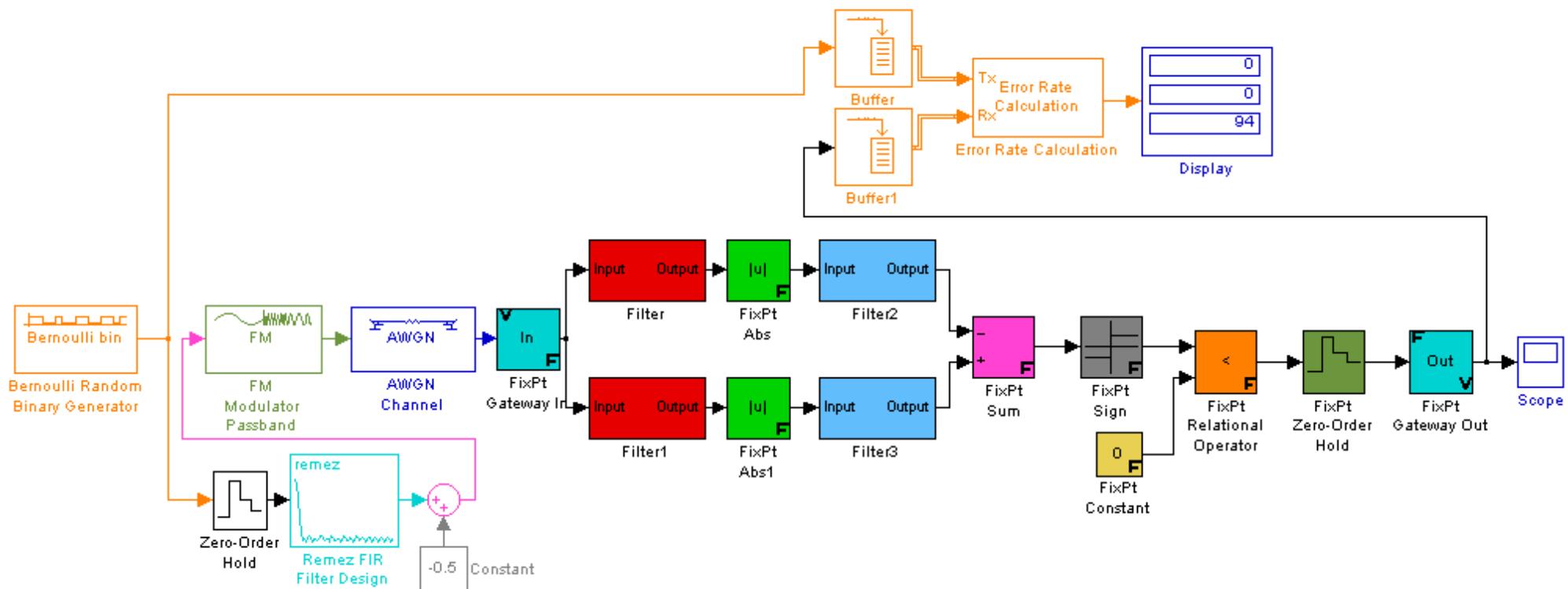


Figure 7.9 Realized Fixed-Point System Model

8

FPGA IMPLEMENTATION

8.1 FPGA-BASED RAPID PROTOTYPING

Approximately, 80% of the project development time is spent at the verification step. Moving towards the System-on-a-Chip makes the verification gap even wider than the design gap. Among the currently available design verification possibilities are [31]:

- 1) Formal Verification
- 2) Simulation
- 3) Accelerated Simulation
- 4) Emulation
- 5) Rapid Prototyping

Each design verification method plays its own role in the design process.

Formal Verification allows design verification with high level of confidence and is good for maintaining functional equivalence at different stages of the design flow and eliminating regression simulations.

Simulation is not expensive and good for initial block-by-block verification of the design, and is widely supported by different abstraction-level commercial simulators.

When the performance of the simulation becomes insufficient to insure the desired test coverage, the accelerated simulation takes place, which is implemented by reinforcing the workstation performance by connecting it to the hardware emulation boards. Working upto 10,000 times faster than simulation, the FPGA-based emulation allows significant extension of the test coverage.

Logic emulation fills a wide verification gap between simulation and actual silicon steps.

When the FPGA implementation achieves the speed of 10-20 MHz, one can speak about rapid prototyping. Working close to the real speed, the rapid prototyping provides an

extensive, ‘live’ test coverage and can be extremely useful for subjective (eye, ear, etc.) evaluation of an electronic product.

Since their introduction by Xilinx Inc., in 1985, the FPGAs became a key technology enabling a rapid hardware prototyping. Their reprogrammability is the basic feature that allows to build FPGA-based rapid prototyping systems.

FPGA Device Families

The FPGA capacity growth beyond 1 million gates in the last 2½ years, made possible by switching to the $0.18\text{ }\mu$ technology, allows to speak about the System-on-a-Programmable-Chip (SOPC). The latest FPGAs incorporate numerous ‘system’ features like:

- 1) embedded memories capable to implement RAM, ROM, FIFO, CAM, etc.
- 2) clock-handling circuitry— DLL, PLL, etc.
- 3) embedded arithmetic resources (carry chains, dedicated multiplier and counter support, etc.)
- 4) hierarchical architectures
- 5) combining LUT and product term logic
- 6) hierarchical routing resources including tri-state buses.

In parallel, large activity is observed regarding the FPGA-related design reuse:

- 1) FPGA vendor-provided macro block generators (COREGenerator from Xilinx Inc., MegaWizard IP Generator from Altera Corporation Inc.)
- 2) FPGA vendor-organized core, alliance, and partnership programs (Xilinx LogiCORE and AllianceCORE, Altera AMPP).
- 3) FPGA design reuse methodology (Xilinx XRMM)
- 4) Related expert programs (Xilinx XPERTs program)

All this approaches the FPGA-based design and prototyping to a system design process.

The reprogrammable FPGA families which make interest for the rapid prototyping are:

- 1) Xilinx
- 2) Altera
- 3) Lucent ORCA
- 4) Actel ProASIC

The largest devices are offered by Xilinx (VirtexE) and Altera (Apex20KE) that are fabricated with $0.18\text{ }\mu$ process.

The majority of the reprogrammable FPGAs are based on the SRAM technology. An exception is the Actel ProASIC family that is based on the Flash technology. The Flash FPGAs do not require the startup bit-stream and are live at power-up. This makes the read back of the configuration bit-stream impossible which attracts interest of the designers delivering IPs because it guarantees security.

FPGA Design Tools

The FPGA design flow starts from the RTL synthesis step, which is usually performed by one of the three market leader tools (Synopsys, Synplicity, and Exemplar). After synthesis, the FPGA vendor-supplied placement and routing tools are used to produce the final configuration bit-stream.

FPGA-based Prototyping Platforms

According to their role and their characteristics, the FPGA-based prototyping platforms can be classified into three major categories:

- 1) high-capacity commercial emulators represented by the three market leaders: Quickturn, IKOS, and Mentor Graphics.
- 2) semi-custom prototyping platforms like Aptix and Simutech
- 3) custom platforms like the Xilinx Prototyping Platforms and platforms developed through industry/university cooperation.

The first category covers the logic emulation field, and the last two categories are basically related to the rapid prototyping field [31].

8.2 VHDL CODE GENERATION

VHDL Code generation was carried out to cut down the design time by reducing the number of design iterations and saving the code debugging time and effort. For code generation, the **Xilinx™ Blockset** for MATLAB™/Simulink™ was used.

XILINX BLOCKSET

The Xilinx Blockset, like other Simulink™ Blocksets, contains elements that can be used to build simulation models. In addition, models built from Xilinx™ Blockset can be translated using the Xilinx System Generator into synthesizable VHDL code.

The Xilinx Blockset elements include VHDL models and association with **Xilinx™ LogiCORE™**'s. These models enable VHDL code to be generated for Simulink™ designs made up of Xilinx™ blocks. The Xilinx™ Blockset uses the **Xilinx™ System Generator** tool for VHDL code generation [68].

XILINX SYSTEM GENERATOR

The Xilinx™ System Generator enables the design of high-performance DSP systems for Xilinx™ FPGAs using the MATLAB™ tools from MathWorks. This software tool automatically generates VHDL code from the system model in Simulink™. The generated VHDL code is optimized for synthesis and implementation in Xilinx™ Virtex FPGAs. To maximize predictability, density, and performance the tool automatically maps the system design to the optimized Xilinx™ LogiCore™ modules. LogiCore™ is a library of IP cores from Xilinx. Because the VHDL code is generated automatically with little or no manual intervention, only the system representation of the design needs to be verified. With only one design representation, the risk of errors is minimized.

When System Generator is invoked, VHDL code, customized cores, VHDL netlist, and test vectors are generated according to the system parameters defined within the system model. The cores are generated using the **Xilinx™ CORE Generator** [68].

XILINX CORE GENERATOR

The Xilinx™ CORE Generator system generates and delivers parameterizable cores for Xilinx™ FPGAs. It facilitates design reuse by integrating user-defined IP cores using the Xilinx™ IP Capture and thus reduces design time.

The CORE Generator contains a library of LogiCORE™ parameterizable cores and Alliance COREs alongwith datasheet of each core. LogiCORE™'s are designed and supported by

Xilinx™, while AllianceCOREs are designed and supported by Xilinx™ AllianceCORE partners.

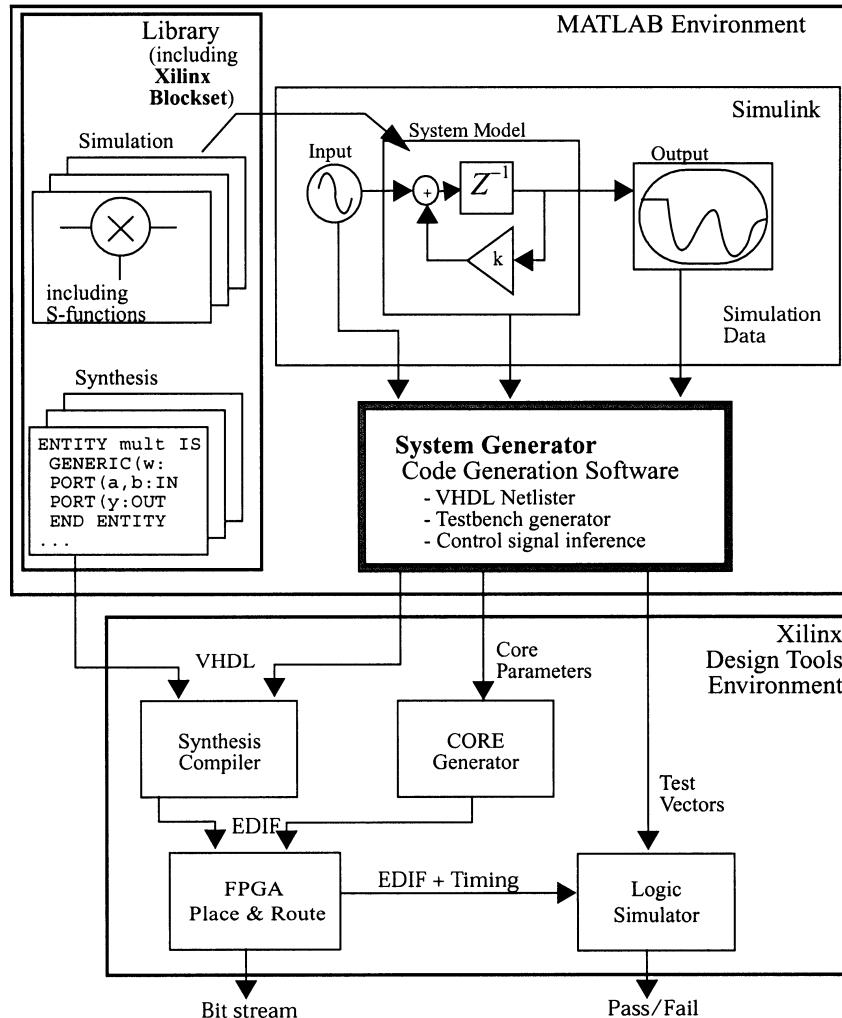


Figure 8.1 The Xilinx™ System Generator & MATLAB™ Interface [68]

8.2.1 XILINX BLOCKSET-BASED SYSTEM MODEL

Appropriate elements in the Xilinx™ Blockset were used to convert the floating-point representation of the system model in Simulink™ to the bit-true representation used in the hardware implementation. The model was then re-simulated to verify its performance with quantified coefficient values and limited data bit-widths, which could lead to overflow, saturation and scaling problems. When the model was converted to a form realizable in an FPGA, and its performance met the specifications, the Xilinx™ System Generator was invoked to generate the VHDL code and the test bench.

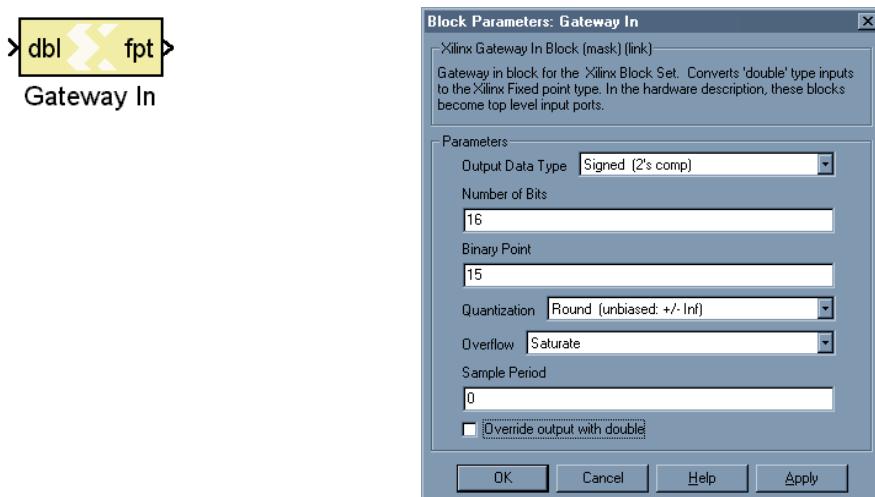
The Xilinx™ System Generator consists of a netlister, a mapper and a test-bench generator. The netlister extracts a hierarchical representation of the model structure annotated with all the element parameters and signal data types. The mapper then analyzes the elements in the hierarchy and creates a VHDL description of the design. The mapper, where possible, uses the Xilinx™ CORE Generator to make hardware macros for specific design elements. When an element or its parameter values imply functionality unavailable in the CORE Generator,

the mapper instantiates a reference to a parameterized, synthesizable entity in a synthesis library or the user-supplied model.

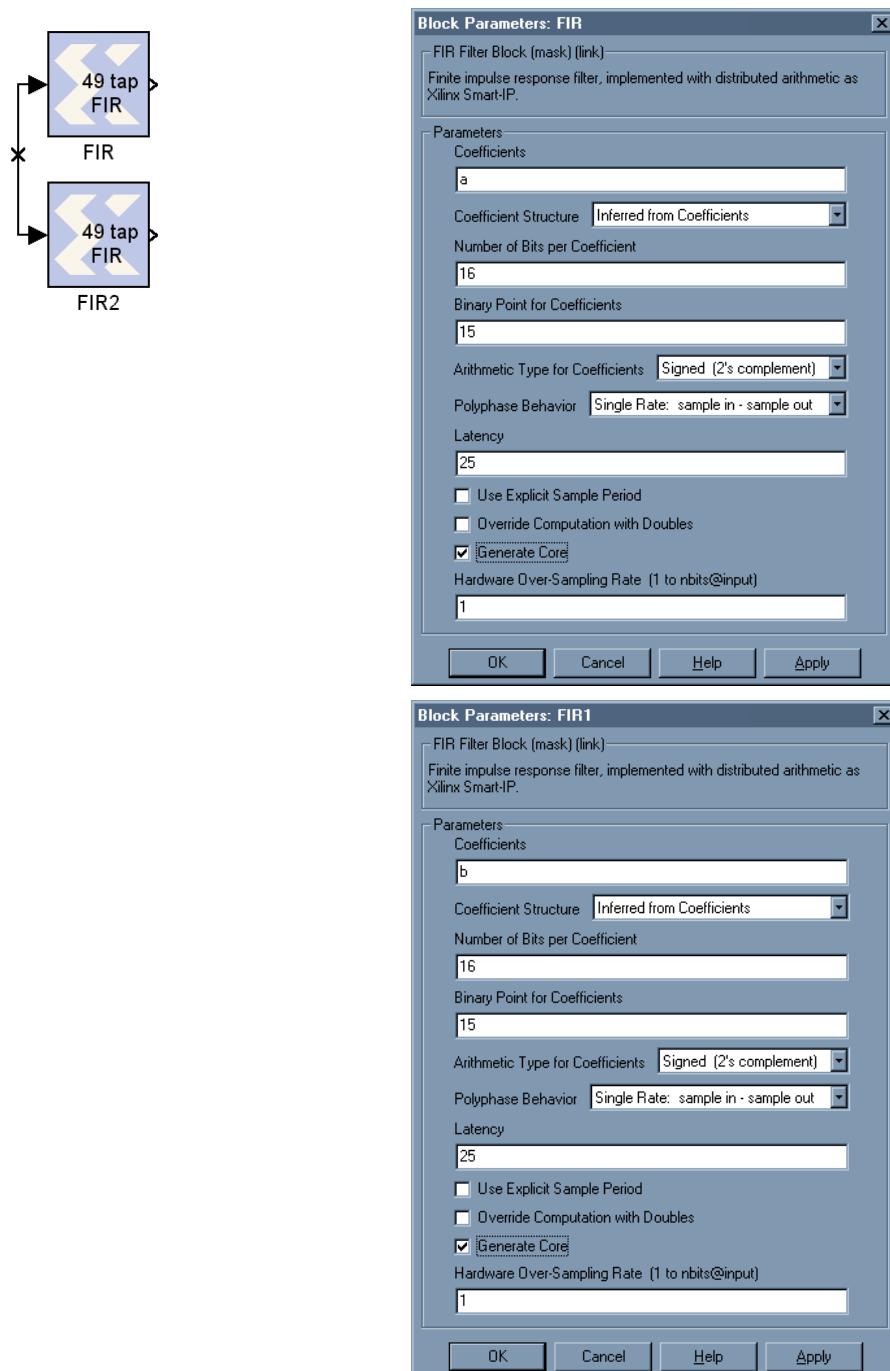
The actual hardware entities used have additional inputs and outputs for control signals that are not evident at the level of abstraction used in Simulink™. The mapper inserts the necessary control ports and connects them up to control logic blocks.

The test-bench generator is an interactive tool that runs in the MATLAB™ environment, in which the designer captures the input stimuli and system outputs of selected simulation runs for conversion to test vectors. The test-bench generator converts the captured simulation data into VHDL code that will exercise the implemented model and test its outputs against the expected results [68].

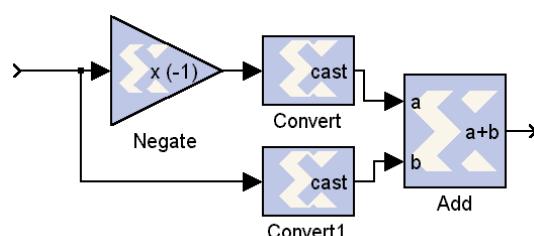
Xilinx™ System Generator Gateway In

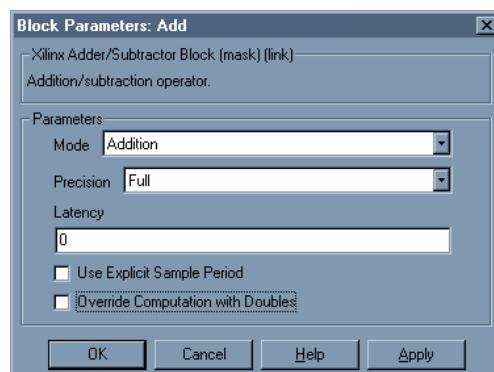
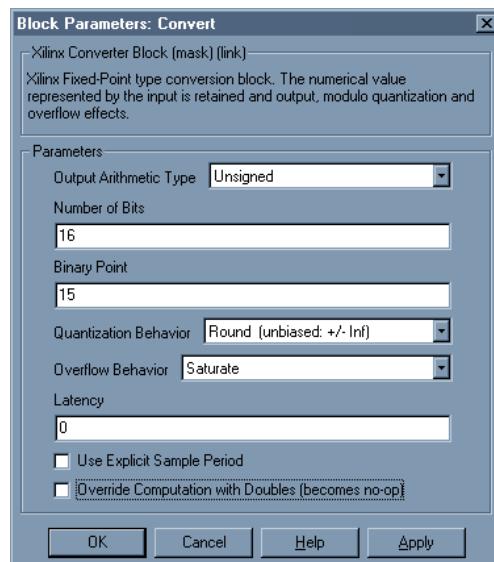
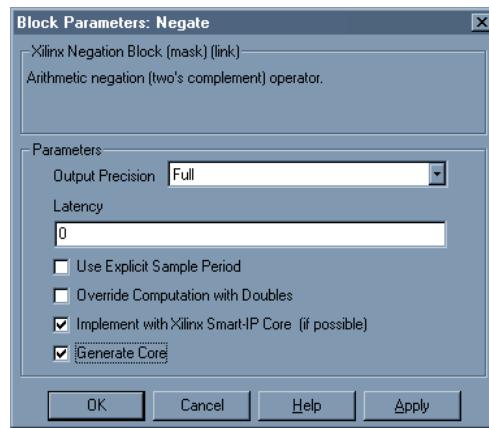


Xilinx™ System Generator Bandpass Filters

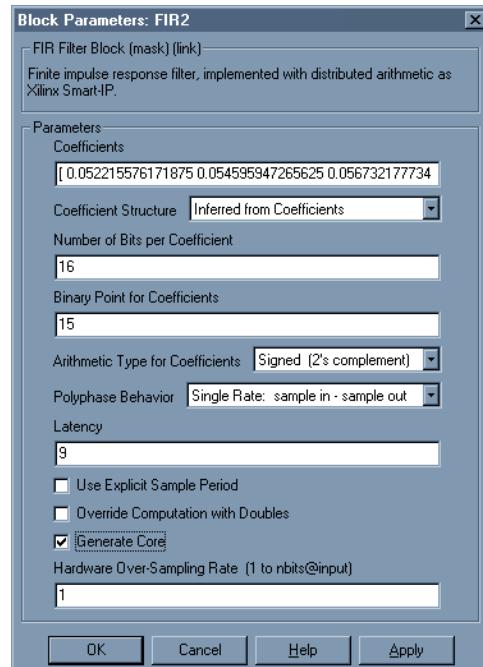


Xilinx™ System Generator Full-Wave Rectifier

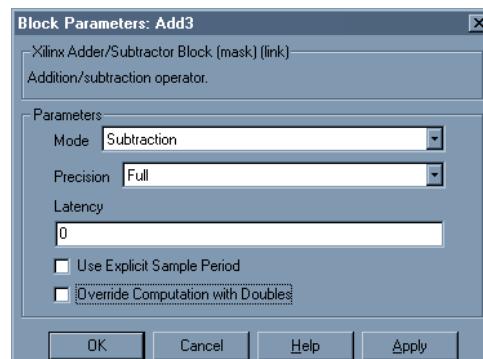
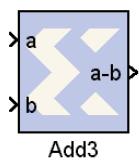




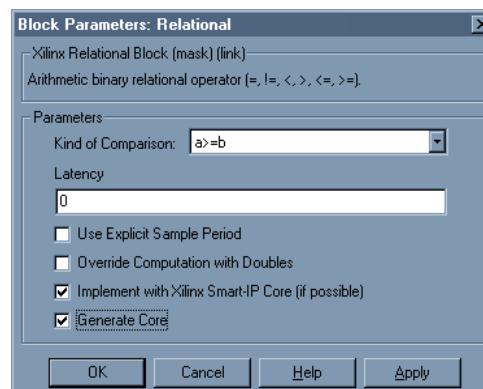
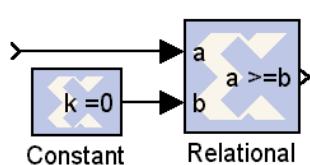
Xilinx™ System Generator Lowpass Filter

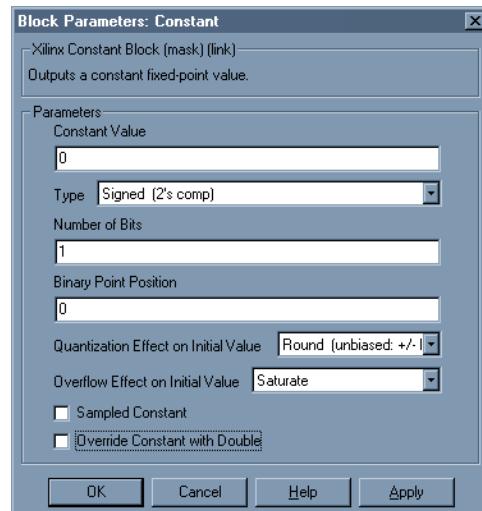


Xilinx™ System Generator Summer

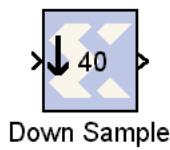


Xilinx™ System Threshold Detector

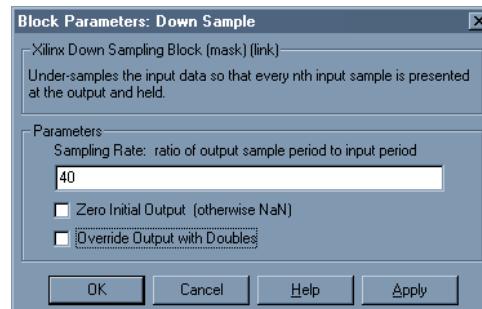




Xilinx™ System Generator Decimator



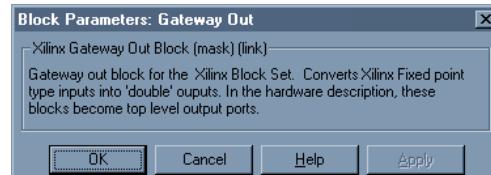
Down Sample



Xilinx™ System Generator Gateway Out



Gateway Out



Xilinx™ System Generator Token

The System Generator Token invokes the VHDL code generation and core generation for all the Xilinx Blockset elements. It enables mixed-mode system simulation.

XILINX SYSTEM GENERATOR TRANSFORMED SIMULATION MODEL OF THE GFSK MODULATOR/DEMODULATOR

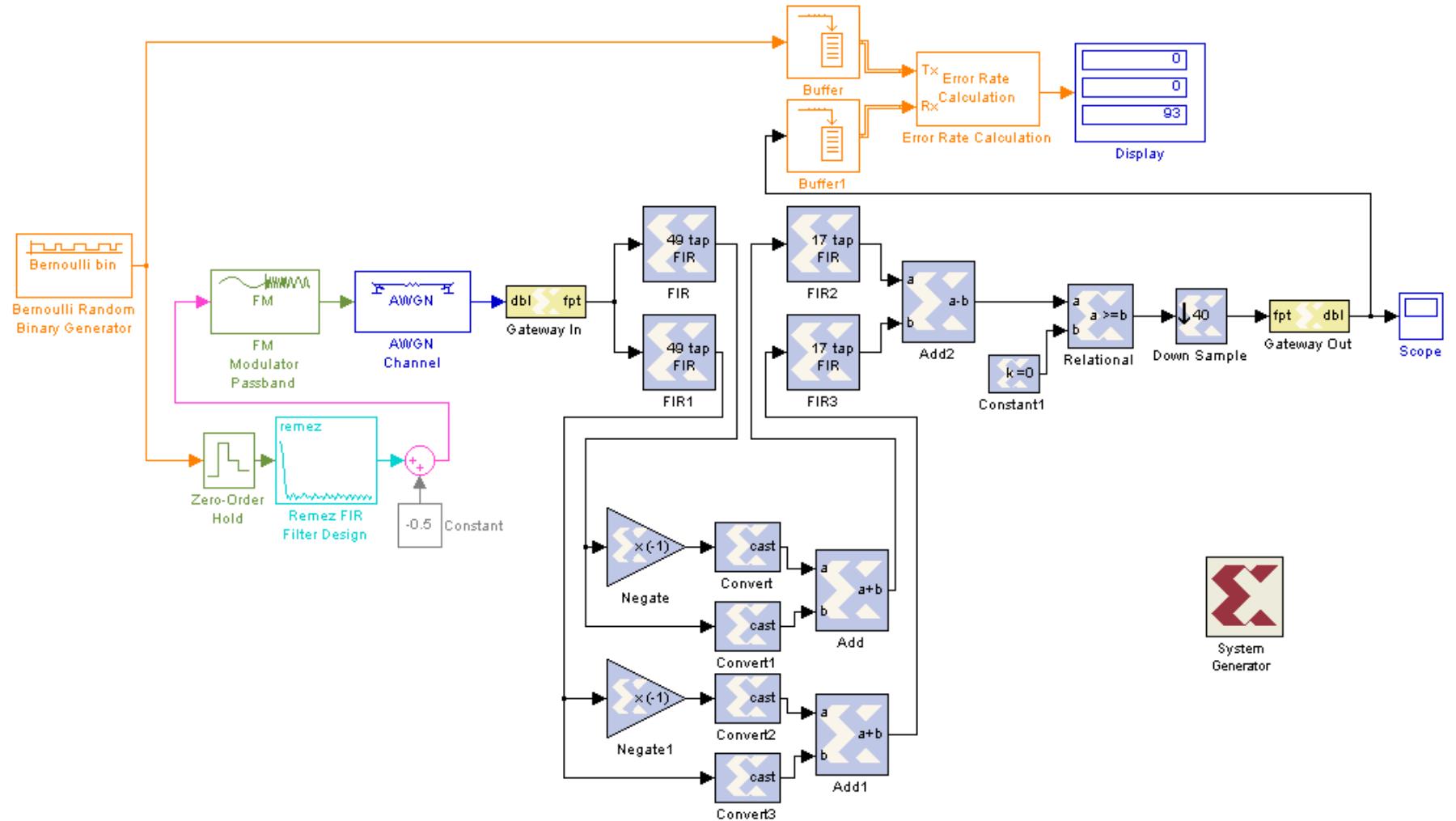


Figure 8.2 XilinxTM Blockset-Based System Model

The generated VHDL code was compiled and synthesized for FPGA implementation using the Synopsys™ FPGA Express. After synthesis, the design was run through the Xilinx implementation tool — Xilinx™ Design Manager (translate, map, place and route, generate configuration bitstream) to produce a bitstream for downloading to an FPGA.

8.3 DESIGN SYNTHESIS

SYNOPSYS FPGA EXPRESS

Synopsys™ FPGA Express enables rapid design of FPGAs and CPLDs on Windows-based PCs. It supports mixed circuit descriptions in any combination of VHDL, Verilog, schematic, and netlist sources. An easy-to-use GUI speeds the design cycle from circuit description to optimized design. Design requirements are entered easily with a familiar spreadsheet interface. Advanced architecture-specific optimization technology, guided by user entered design requirements result in synthesized designs that take full advantage of high-density and high-performance FPGA and CPLD devices. FPGA Express supports leading programmable architectures from Actel™, Altera™, Lucent™ Technologies, and Xilinx™ [60].

8.4 DESIGN TRANSLATION, MAPPING, PLACEMENT & ROUTING

XILINX DESIGN MANAGER / FLOW ENGINE

The Xilinx™ Design Manager manages the Xilinx™ FPGA designs while the Xilinx™ Flow Engine implements the Xilinx™ FPGA designs. The Flow Engine is closely integrated with the Design Manager sharing many of the same menus and dialog boxes. The Design Manager and the Flow Engine were used together to implement the FPGA design.

XILINX CONSTRAINTS EDITOR

The Xilinx™ Constraints Editor was used to lock the inputs and outputs in the design to the specific pins on the FPGA package so that they are easily accessible while testing the design on the test board.

XILINX FPGA EDITOR

The FPGA Editor is a graphical application for displaying and configuring FPGAs. This application was used to display the placed & routed FPGA design.

SUMMARY

This chapter detailed the process of rapid design prototyping using the FPGA. The objectives of rapid design prototyping were explained alongwith the sequence of steps and software tools employed for converting the Simulink™-based system model to fit into an FPGA.

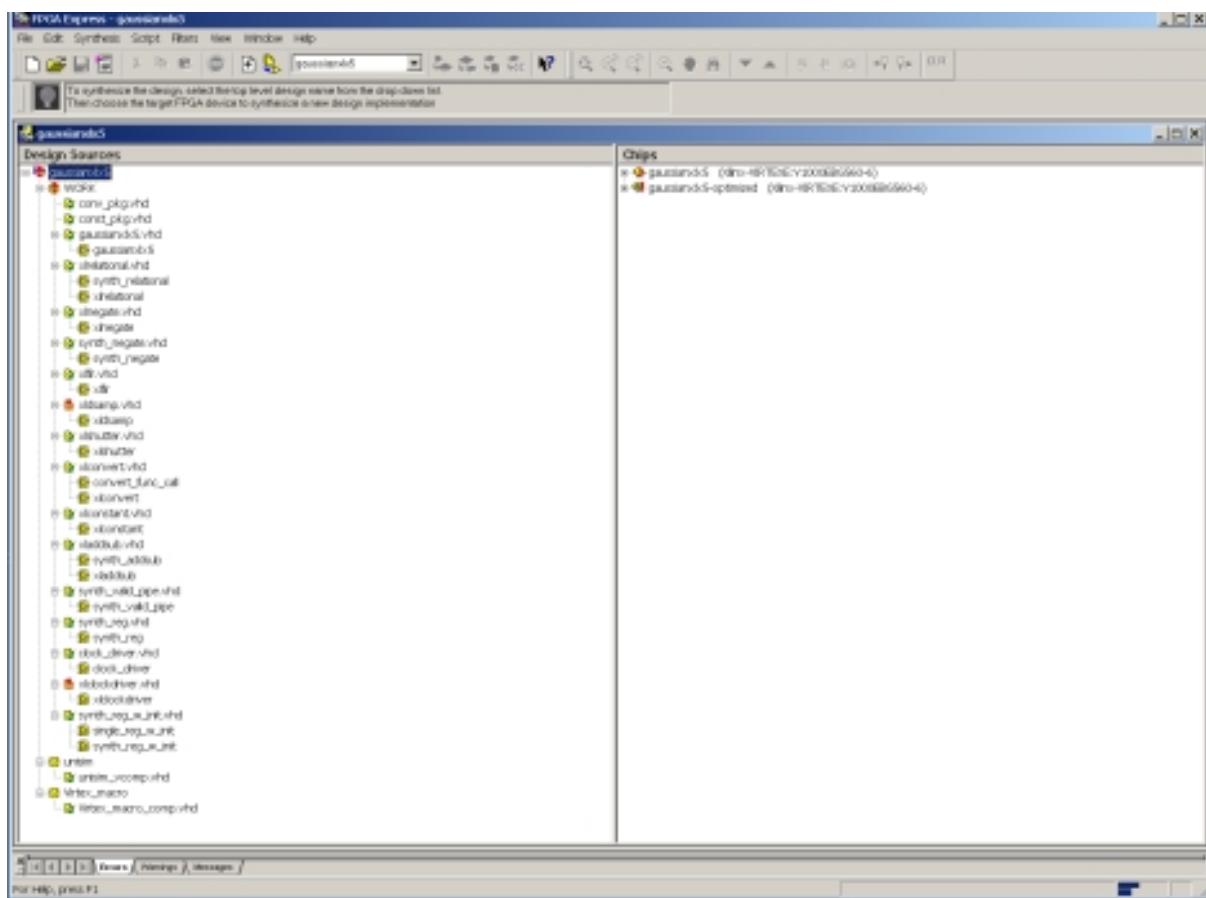


Figure 8.3 Design Synthesis Using The SynopsysTM FPGA Express Synthesis Environment

9

SYSTEM TESTING

9.1 FPGA PROGRAMMING

The configuration bt-stream file generated by the Flow Engine in the Xilinx™ Design Manager was loaded into the **Xilinx™ Hardware Debugger** tool with the serial port of the host computer linked to the **Xilinx™ Mulilinx Program Cable** whose other end was connected to the custom-built FPGA prototyping board. After performing checks on the integrity of the connection between the PC and the FPGA board through the program cable, the configuration bit stream was successfully downloaded to the Virtex™ 1000E FPGA on the FPGA board.

9.2 FPGA TEST STRATEGIES

Testing the design was (and is) a significant challenge because of a number of reasons:

The amount of test vectors generated by the Xilinx™ System Generator was huge (the test vector file was several mega bytes large). As the required input into the FPGA is at 40 MSamples/s with each sample 16 bit wide, handling such a large amount of data was not possible with the Pattern Generator available for test vector generation. Therefore, two strategies were followed for testing the design:

1) SIGNATURE ANALYSIS

It was decided to adopt a sort of Signature Analysis approach for testing. The XBS-based system model was simulated with pseudorandom noise as its input and the resulting system response was recorded.

The pattern generator was then programmed with a similar pseudorandom test vector sequence which was then applied to the FPGA input on the test board. However, it was found that the custom-built prototyping board was not designed to handle signals at 40 MHz as it severely distorted both the input clock as well as the input test vectors. Moreover, the number of FPGA pins available at the connectors on the FPGA board was limited. As the Xilinx™ VirtexE device used on the board is in Ball Grid Array (BGA) package, therefore, direct probing was not possible.

To get access to the available pins at the connectors, the floorplan of the design was modified using the Xilinx™ Constraints Editor to lock the input and the output pads to FPGA pins available at the connectors on the board. This step resulted in problems during the placement and routing of the FPGA as the router program had to operate with the stringent timing requirements as well as the pin-locking constraints. Several attempts were made by increasing the number of the placement and routing passes to fit the design within the device but apart from taking a lot of time for completion, it did not succeed in meeting the timing constraints. This testing approach is still being followed by improving the design of the connectors carrying input signals from the pattern generator to the FPGA board and modifying the design to run at 20 or 10 Msamples/sec.

2) ON-CHIP PROBING

As the density of the FPGA design was causing the impracticality of attaching test equipment probes to the device. The Xilinx™ ChipScope tools were used to integrate key logic analyzer hardware components with the target design inside the Virtex™ 1000E FPGA. The ChipScope tools communicate with these logic analyzer hardware components and provide a complete logic analyzer without the need for the cumbersome test probes or the expensive test equipment.

XILINX CHIPSCOPE

The ChipScope tools include

- the ChipScope Logic Analyzer Core Generator
- the ChipScope Logic Analyzer Core Inserter, and
- the ChipScope Logic Analyzer

The ChipScope Logic Analyzer Core Generator provides netlists and instantiation templates for the Integrated Controller (ICON) core and the Integrated Logic Analyzer (ILA) core.

The ChipScope Logic Analyzer Core Inserter automatically inserts these two cores into the synthesized design.

The ChipScope Logic Analyzer allows setup and trace display for the ILA core. The ILA core provides the trigger and trace capture capability. The ICON core communicates to the dedicated Boundary Scan pins.

The Logic Analyzer supports the Xilinx Multilinx program cable for communication between the host computer and theFPGA. The ILA and the ICON cores can be placed in the design either by [53]:

- generating the cores with the ChipScope Core Generator and then instantiating them into the source HDL code, or
- inserting the cores into the post-synthesis EDIF (*.edf) netlist using the ChipScope Core Inserter.

The later approach was adopted for ILA and ICON core insertion into the design. The design was then placed and routed using the Flow Engine in the Xilinx™ Design Manager. However, this design again ran into the problem of not meeting the timing constraints. This approach is still being followed by optimizing the design to reduce on-chip logic consumption and create more room for placement and routing on the chip.

The ChipScope Logic Analyzer supports user-selectable data channels range from 1 to 256 with the number of sample sizes range from 256 to 4096. The triggers can be changed in real-time without affecting the design logic.

SUMMARY

This chapter explained the test strategies adopted for testing the design after configuring the FPGA from the host computer. Two techniques were employed for testing and their relative merits and demerits were elaborated alongwith the problems faced during the testing of such a large prototype.

10

SUMMARY & CONCLUSIONS

10.1 SUMMARY & CONCLUSIONS

The increased size and complexity of designing highly integrated integrated circuits, coupled with compressed design times, has led many ASIC designers to rethink their traditional design processes. Compressed design times demand increasing the chance of success on the first attempt. Therefore, ASIC designers have to develop a well-thought-out methodology that includes all aspects of the design from concept to silicon.

A successful approach is to have a common design environment for designing all the system components. This approach reduces design errors by reducing the level of manual intervention in the design flow. It also facilitates the incorporation of optimization at every design stage and to have early considerations of design verification. With only one design representation, designers can significantly reduce, especially, the hardware development time by quickly iterating between system-oriented and hardware-oriented approaches to their designs. This is especially more important for DSP applications as many system-level design tradeoffs are based upon the outcome of the hardware implementation.

The main emphasis of this project was to gain hands-on experience with such a common design environment and to evaluate its performance using Bluetooth as a design example. Although the project work was concentrated on the hardware design part of the design flow, however, some insight was gained into the issues of software design, hardware/software integration and system-level design as a whole. Familiarity was also gained with the limitations of the design environment used and techniques were explored to devise workaround methods to circumvent those limitations.

The common design environment used for this project was MATLABTM alongwith its suite of design tools. A GFSK Demodulator block from a BluetoothTM receiver was selected as a design example.

Using SimulinkTM, and its Communications and DSP Blockset libraries, floating-point algorithms were developed that defined the system functions. Within this design environment, a floating-point model of the system was created and simulated to verify the performance of the designed algorithms and to extract the unknown design parameters. Working in this fashion, all the algorithms could be functionally verified while they were still in a floating-point format. It was extremely important to know that the designed algorithms are functionally correct before moving to subsequent stages in the design process.

After the floating-point system model was running, a Bit Error Rate (BER) check was performed on the system to get a theoretical best case while meeting the system specifications. This best case system model served as a starting point to optimize the system design by taking out taps from the filter blocks to minimize the final system implementation cost.

To determine the minimum number of filter taps that could be used, the number of taps was decreased and the system model was resimulated to check the BER. After iteratively simulating a number of times the filter orders were fixed to numbers that yielded the BER within system specifications while keeping tolerance for slight performance degradation after fixed-point conversion.

As floating-point algorithms cannot be used efficiently in low-power, small area hardware implementations, therefore, the algorithms had to be converted into fixed-point data and arithmetic representations.

After ensuring that the designed algorithms meet the system specifications, fixed-point format transformation of the design was carried out which involved, among other tasks, converting the signal nodes and design parameters to a fractional fixed-point format without changing the underlying functional behavior. This step was performed to study the finite word-length effects encountered when the algorithms are processed on silicon by the computation elements having storage registers with finite bit-widths.

A fixed-point model of the transformed system algorithms was constructed using the Fixed-Point Blockset library and the Quantized Filtering Toolbox to ensure that the functionality of the adapted algorithms corresponds exactly to the floating-point system algorithms. The MATLAB design environment provided a uniform test suite that could be used at every step in the design process eliminating one potential source of uncertainty should problems arise.

As a second system optimization step, data bit-widths were minimized to lower the cost and power consumption of the final system design. Simulations of the fixed-point system model were used to optimize the data bit-widths and to check for their effects on the BER achieving a compromise between the BER and system cost.

To explore the sensitivity of different hardware realization structures to quantization, execution speed, silicon area, and system power requirements, the filter blocks were synthesized into different realization structures using the Filter Realization Wizard. The synthesized filter structures were plugged into the fixed-point system model and simulated again to validate the system functionality.

Working in SimulinkTM, the functionally-validated floating-point system model was then transformed into a bit-true and cycle-accurate system hardware model using the Xilinx Blockset library. The model was re-simulated to verify its performance with quantified filter coefficient values and limited data bit-widths, which could lead to overflow, saturation and

scaling problems. Necessary delays were also padded into different system blocks for faster hardware execution through pipelined implementation. When the model was converted to a form realizable in an FPGA, and its performance met the specifications, the VHDL code and the testbench were generated using the Xilinx™ System Generator tool alongwith the Xilinx™ CORE Generator tool.

The Synopsys™ FPGA Express was used to synthesize the generated VHDL code. The Xilinx™ ChipScope tool was used for test-point insertion so that various signals can be monitored inside the FPGA-implemented design during hardware testing. The Xilinx™ Design Manager and Flow Engine were then used for mapping, translation, placement, routing, and configuration bit stream generation.

The Xilinx™ Hardware Debugger was used to download the configuration bitstream to a Xilinx™ Virtex series FPGA using the Xilinx™ Multilinx programming cable on a custom-built FPGA-test PCB (Printed Circuit Board). A Hewlett-Packard™ Pattern Generator was used to generate the input test vectors and a Hewlett-Packard™ Logic Analyzer and a Hewlett-Packard™ Digital Storage Oscilloscope were used to monitor the input and the output signals.

10.1.1 SEMI-CUSTOM ASIC DESIGN

After having prototyped the system on an FPGA and having evaluated the system performance, the prototyped design can be retargeted to a semi-custom ASIC using the standard-cell-based design approach.

The anticipated issues in this process can be the custom Xilinx™ IP cores for the system blocks that cannot be retargeted to a standard-cell library. One possible solution can be to generate the VHDL code for the system by turning off the core generation option in the System Generator tool. This will not yield a fully-optimized VHDL code but it can be later manually optimized and synthesized and mapped on a standard-cell library, possibly using the Synopsys™ synthesis tools interfaced with the Cadence™ layout generation tools.

10.1.2 SYSTEM MODEL REFINEMENT

Inclusion of RF Distortions

The system model can be modified to include the RF frontend and RF propagation channel distortion effects to make it reflect the real-world environment. While there are a large number of RF frontend model parameters that one can play with in simulations, its important to focus on the parameters that have the greatest effect on the performance of the system being designed. Only those distortions should be included that are most important as each distortion type requires a considerable amount of modeling effort and simulation time.

The most important RF propagation channel distortion is multipath distortion (present in real-world digital communication channels).

Taking into account the accurate RF frontend and RF propagation channel models in simulations can give an additional tool to improve performance and eliminate problems before more costly hardware prototyping and testing ensues.

RF Frontend Distortions

The most important RF frontend distortion effects include power amplifier non-linearities, and phase noise of the PLL frequency synthesizer.

There are power amplifiers in any communication system that transmits over an RF channel. Unfortunately, power amplifiers do not have an unlimited amplifying capability. Every power amplifier, at some point, starts clipping/compressing and distorting the signal being

amplified. Compression and distortion are modeled using the 1-dB compression point and the 3rd-order intercept point. The 3rd-order intercept point specifies a theoretical point where the power amplifier's fundamental output intercepts the 3rd-order distortion products. The 1-dB compression point and the 3rd-order intercept point should be included when designing the baseband DSP algorithms to avoid unexpected performance problems in the real-world system.

All oscillators in RF communication systems have phase noise. Essentially, phase noise is very small amounts of phase modulation on the oscillator that creates a modulation envelope rather than the theoretical single frequency that it is expected to generate. Excessive phase noise can raise havoc with any digital communication system using a local oscillator in the RF frontend. Low phase noise oscillators can be used to avoid this problem, but they tend to be much more expensive than their higher phase noise counterparts. System-level simulation can do a system-level trade-off study of the lower cost of using a higher phase-noise oscillator versus the increased signal processing that may be necessary to meet the system BER requirements.

RF Propagation Channel Distortion

Multipath distortion causes Inter-Symbol Interference (ISI), and, therefore, can cause the system to fail BER requirements. An accurate channel propagation model (e.g., Rician or Rayleigh) should be added to decide whether (adaptive) equalization is required in the baseband processor to correct for time-dispersion problems, thus canceling ISI.

Including these three key RF frontend distortions in the system analysis will yield better results.

Inclusion of A/D Converter Model

An implicit model for the A/D converter was used in the system model that assumed flash A/D conversion with excessive oversampling to compensate for the quantization noise effects. To accurately model the system, however, an oversampling Σ-Δ A/D converter model should be used with the oversampling ratio and the order of the noise-shaping loop selected to achieve a converted SNR that fits well within the system noise budget for achieving the specified BER.

Optimization of System Model

The present system model can be optimized in many ways. The most important optimization can be reducing the filter order by using the Optimal or Parks-McClellan algorithm for coefficient calculation. Wavelet transform-based techniques have also been tried for designing the bandpass filters for FSK demodulation [29]. Another optimization can be to reduce the input SNR, oversampling ratio and the data bit-width. Yet another optimization option worth exploring can be to convert the RZ signal to a bipolar signal before Gaussian filtering which might reduce the DC level introduced after Gaussian filtering and thus reduce the ISI resulting in improved BER performance of the system making allowance for further optimization in other system blocks.

As another option, the MATLABTM/SimulinkTM-based system models can be transported either to COSSAPTM or SPWTM and can be further refined utilizing the more powerful features available in those design environments that can also provide a more uniform interface to semi-custom ASIC implementation.

10.1.3 LOW POWER DESIGN

A major factor in the size and weight of portable devices, possibly having BluetoothTM capability in the future, is the size and weight of batteries which is directly impacted by the

power dissipated by the electronic circuits. Moreover, the significant cost of providing power and associated cooling demands reduction in the power consumption of a portable system.

Components Of Power Consumption

Dynamic Components:

- 1) Switching Component
- 2) Short Circuit Component

Static Component:

- 1) Leakage Component

Approaches To Low-Power Design

The approaches to power consumption reduction encompass all possible aspects of a system design ranging from the technology being used for the system implementation, circuit and logic topologies, the system architectures and even the system algorithms. Only algorithmic and architectural transformations will be discussed here as these are the most relevant in the context of the design flow used for the project.

Algorithmic Transformations

The choice of algorithms is the most highly leveraged decision in meeting the power constraints. The ability for an algorithm to be parallelized is critical and the basic complexity of the computation must be highly optimized. The most important algorithmic tradeoffs for low-power design include scaling to lower voltage through exploitation of concurrency and reduction in switching activity by minimizing the number of operations.

Voltage Reduction using Algorithmic Transformations

This includes removal of feedback paths as in IIR filters by algorithmic transformations to achieve parallelism resulting in voltage scaling. The most common method to remove feedback paths is loop unrolling.

Minimizing Number of Operations

The most effective approach to minimizing the number of operations is to convert multiplications with constants (as in digital filter structures) into shift-add operations. The application scope of this transformation is large. The basic idea is that multiplication with a 0 is a NOP (No OPeration) and, therefore, a multiplication with a constant degenerates to shift-add operations corresponding to the 1's in the filter coefficient. Techniques and tools exist to scale the filter coefficients so as to minimize the number of shift-add operations. In addition, if an input is being multiplied with multiple coefficients, some of the shift-add terms can be shared and the number of operations can be further reduced.

Architectural Transformations

Maintaining a given level of computation or throughput is a common concept in signal processing where there is no advantage in performing the computation faster than some given rate, since the processing element will simply have to wait until further processing is required. One of the most important ramifications of only maintaining throughput is that it enables an architecture-driven voltage scaling strategy, in which aggressive voltage reduction is used to reduce power, and the resulting reduction in logic speed is compensated through parallel architectures to maintain throughput [30].

Finally, the choice of selecting either a convolver-based demodulator architecture alongwith envelope detection or a frequency discriminator-based demodulator for the GFSK demodulation for commercial Bluetooth designs seems quite even-handed. The recently announced BlueCore™ 01 single-chip Bluetooth solution by Cambridge Silicon Radio Ltd., U.K. [34] claims to use a digital demodulator that, probably, is a convolver-based digital demodulator. On the other hand, the single-chip Bluetooth solution by the Broadcom Corporation, U.S.A. [36] uses the frequency discriminator-based demodulator. Cadence Design Systems Inc., has included both demodulators in its Bluetooth library for the Signal Processing Worksystem (SPW) system-level design tool. Probably, the choice depends upon the level of expertise and confidence of a particular design group in a particular architecture.



APPENDICES

APPENDIX-A

DATA SHEETS OF THE DESIGN TOOLS

The MathWorks

System-Level Design Products for DSP and Communications



The
**MATH
WORKS**
Inc.

To succeed in a competitive global marketplace, electronics, telecommunications, and aerospace companies must develop highly innovative products and get them to market fast. With ever-increasing IC density, processor speed, and software complexity, traditional design tools no longer ensure a company's competitive edge.

Today's engineers need tools that streamline the design process and help them discover new ways to achieve technological breakthroughs.

The MathWorks system-level design products address these needs, whether you are analyzing data and developing algorithm concepts, simulating system and component behavior, performing real-time prototyping, or testing the hardware or software implementation. These products provide a complete, integrated software environment that accelerates development cycles and simplifies design verification and reuse.

Our system-level design environment is based on Simulink®, a powerful block diagram simulation environment. Simulink is built on top of MATLAB®, the leading software for DSP algorithm development. Your entire design team can benefit from the tools in the Simulink environment. These tools will dramatically reduce the time you spend programming and correcting design problems, freeing you to explore ideas, develop leading-edge technology, and deliver first-rate product designs on time.



[The MathWorks DSP and Communications Design Environment](#)

4

[Total System Simulation](#)

6

[Tools for Real-World, Real-Time Design](#)

8

[Rapid Prototyping and Design Verification](#)

10

[About The MathWorks](#)

11

The MathWorks DSP and Communication Design Environment

Meeting the escalating demand for higher performance, lower cost, and faster delivery of products requires flawless coordination among specialized development teams. Simulink offers a highly integrated alternative to traditional tools and fragmented design processes, making it easy for design teams to work together. Because Simulink models are portable across PC and UNIX platforms, every engineer on your team gets a consistent view of the design and a clear, executable specification for each hardware and software component of your system.

Simulink streamlines communications and DSP design by providing the fastest path from product concept to validated system model. And it maximizes scarce engineering resources by enabling you to move a design effortlessly through algorithm development, behavioral simulation, and model verification without having to transfer data, rewrite code, or change software environments.

With Simulink, you can test design concepts and tradeoffs earlier in the development process. By verifying your design at the system level, you minimize the risk of expensive errors in your software or silicon. Eliminating these errors early cuts your design time and development costs.



Simulink®

A graphical simulation environment for the system-level design and modeling of digital, analog, and mixed-signal systems

MATLAB®

A high-performance technical computing environment for algorithm development, data analysis, and visualization

DSP Blockset

Simulink block libraries for the design and simulation of real-time digital signal processing systems

Signal Processing Toolbox

MATLAB functions and graphical user interfaces (GUIs) for algorithm development, signal and linear system analysis, and spectral estimation

Communications Toolbox

Simulink block libraries and MATLAB functions for modeling the physical layer of a communications system

Fixed-Point Blockset

Simulink block libraries for the design and simulation of bit-true algorithms

Stateflow®

A graphical environment for the design and simulation of event-driven systems, protocols, and control logic

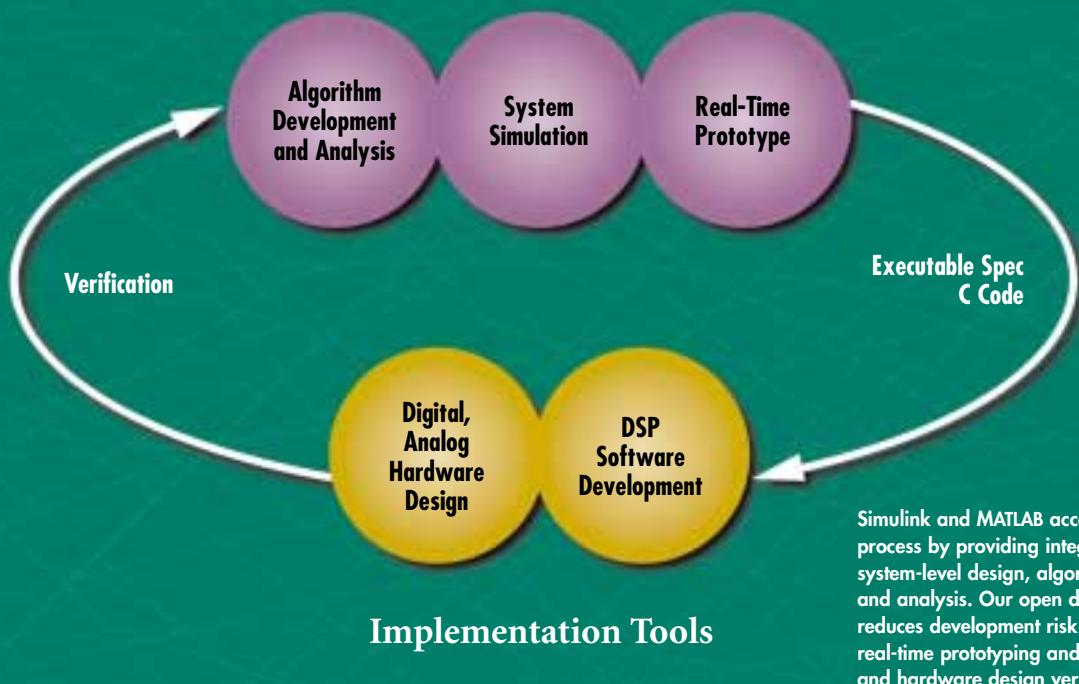
Real-Time Workshop®

A real-time development environment that automatically generates C code directly from Simulink models

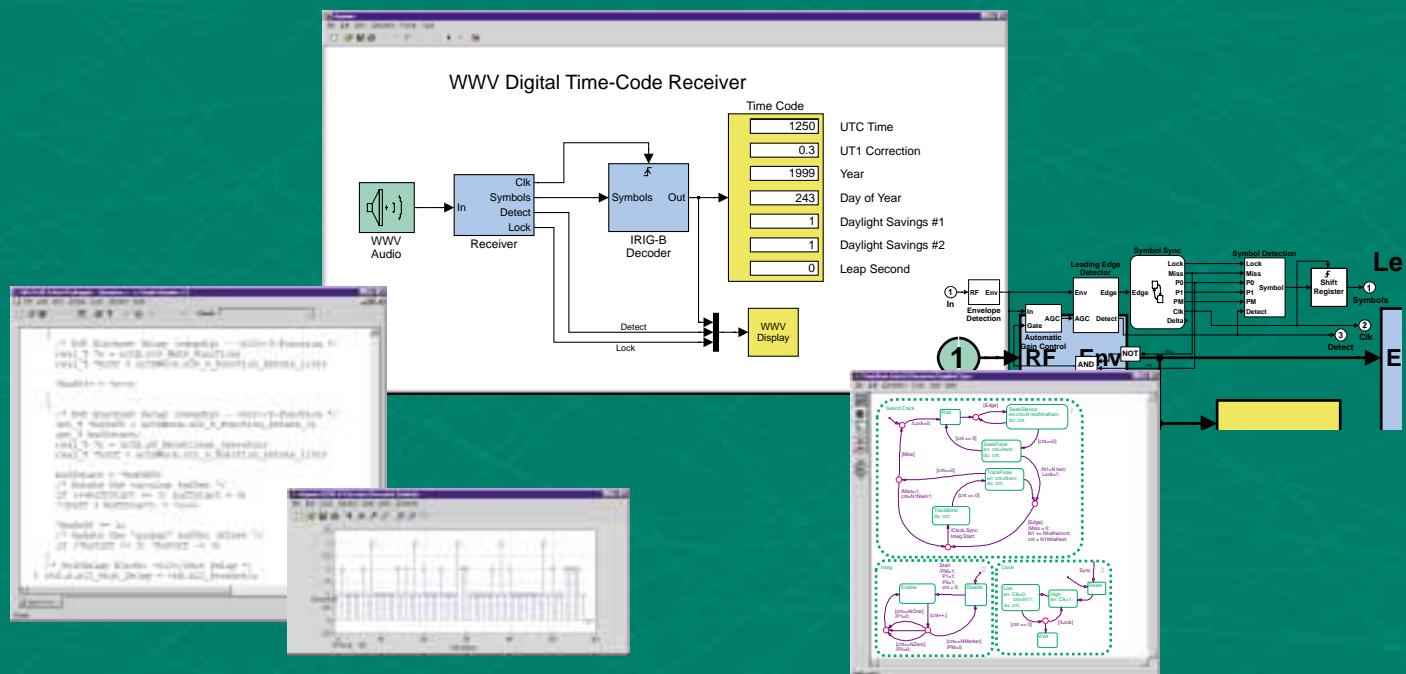
MATLAB and Simulink Report Generators

Data-reporting tools that generate RFT and HTML format documents from MATLAB programs and Simulink models

MathWorks System-Level Design Environment



Simulink and MATLAB accelerate your design process by providing integrated tools for system-level design, algorithm development, and analysis. Our open design environment reduces development risk by providing rapid real-time prototyping and simplified software and hardware design verification.



Digital Receiver Design

This digital time-code receiver simulation illustrates a typical communications design problem that requires integration of signal processing and event-driven functions such as control logic and synchronization. The Simulink system-level design environment, together with the DSP Blockset (top and far right), and Stateflow (bottom right), gives you a single visual environment that lets you hierarchically model multimode systems that include analog, digital, and mixed-signal components. Real-Time Workshop (far left) generates readable, efficient C code from Simulink models for real-time prototyping and high-speed simulation. Simulink is the only simulation tool that is fully integrated with MATLAB (bottom center), giving you instant access to advanced algorithms and analysis capabilities.

Total System Simulation

Simulink is your bridge between R&D and product implementation. It provides a highly productive, easy-to-use, and open block diagram modeling environment that encourages collaboration and complements traditional design and development tools. Simulink accelerates product development by making it easy for systems engineers, DSP developers, and hardware designers to create and maintain complete behavioral models, characterize alternative algorithms and architectures, and verify system performance.

DSP software; digital, analog, and mixed-signal datapath hardware; control logic, communications protocols, and synchronization loops; and channels, acoustics, and other physical effects. These integrated simulations let you see immediately how each

design decision affects the behavior of the whole system.

Rapidly design and validate system models

Simulink enables you to graphically design and simulate large-scale, complex systems with a minimum of effort. Using drag-and-drop editing and intuitive model navigation, you can create and maintain models that are easy to understand and modify—even on systems that have thousands of blocks and multiple hierarchical levels. You'll produce a working, validated model in a fraction of the time it would take with conventional simulation tools and languages.

Instantly visualize and tune your design

Simulink models give you instantaneous feedback. You run simulations with a click of the mouse, tune model parameters to perform “what-if” analysis, and visualize results dynamically on animated scope displays. These interactive features help you rapidly evaluate alternative algorithms and determine optimal parameter settings.

Silicon Arts Inc.—When Silicon Arts designed an ASIC for a TDMA satellite communications link, they chose Simulink over C or alternative products because it integrated all the tools they needed to simulate, analyze, and verify the architecture of their target system. Simulink's integrated modeling and test environment enabled Silicon Arts to eliminate errors in their HDL designs. As a result, their ASICs were built to the customer's specs the first time.



"The customer cannot afford a broken design. Using block diagram tools like Simulink reduces the risk of finding design errors before the design goes to fabrication."

—Robert Schutz, President, Silicon Arts Inc.

Procedural languages are useful for developing specific algorithms but make it difficult to analyze, maintain, and share complex system models. Simulink lets you develop detailed designs—and integrate your MATLAB and C code—with the context of a concise, visual description of system behavior. You build your designs using hundreds of predefined blocks, state-of-the art algorithms, and interactive simulation and display capabilities.

Simulate multidomain systems

Simulink is the only system-level design environment that provides integrated support for discrete- and continuous-time-driven systems and event-driven systems within a single graphical model. Because it supports multichannel, multirate systems, Simulink lets you create cleanly partitioned, efficient representations of real-world systems.

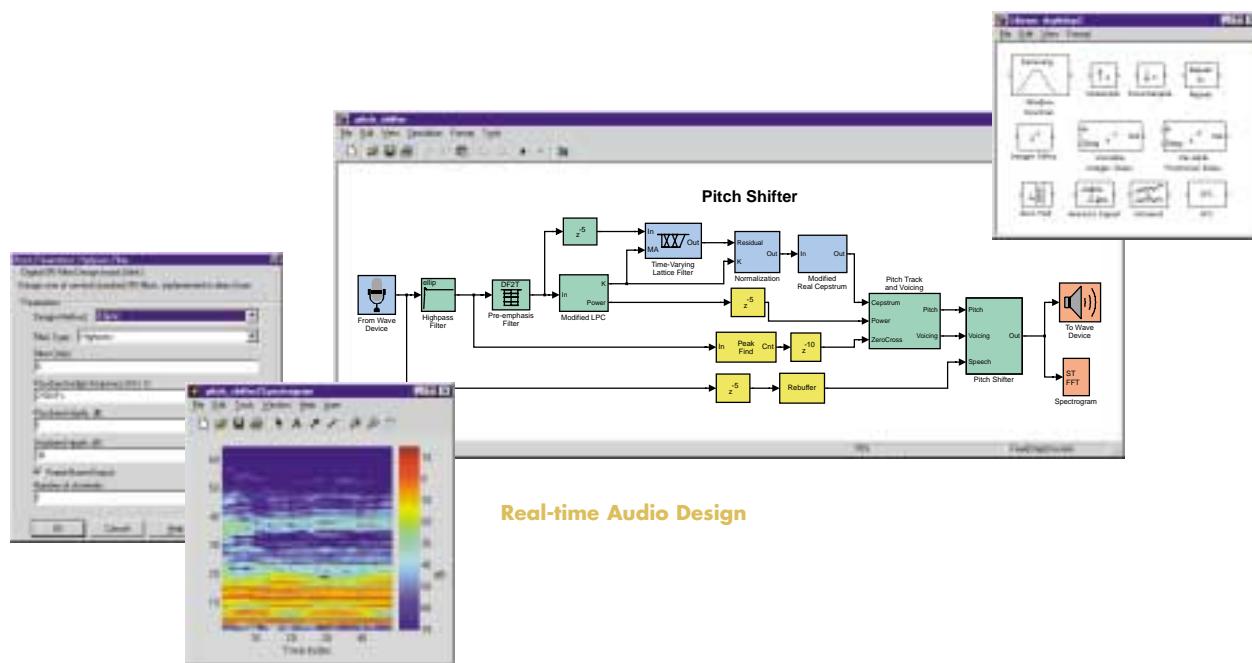
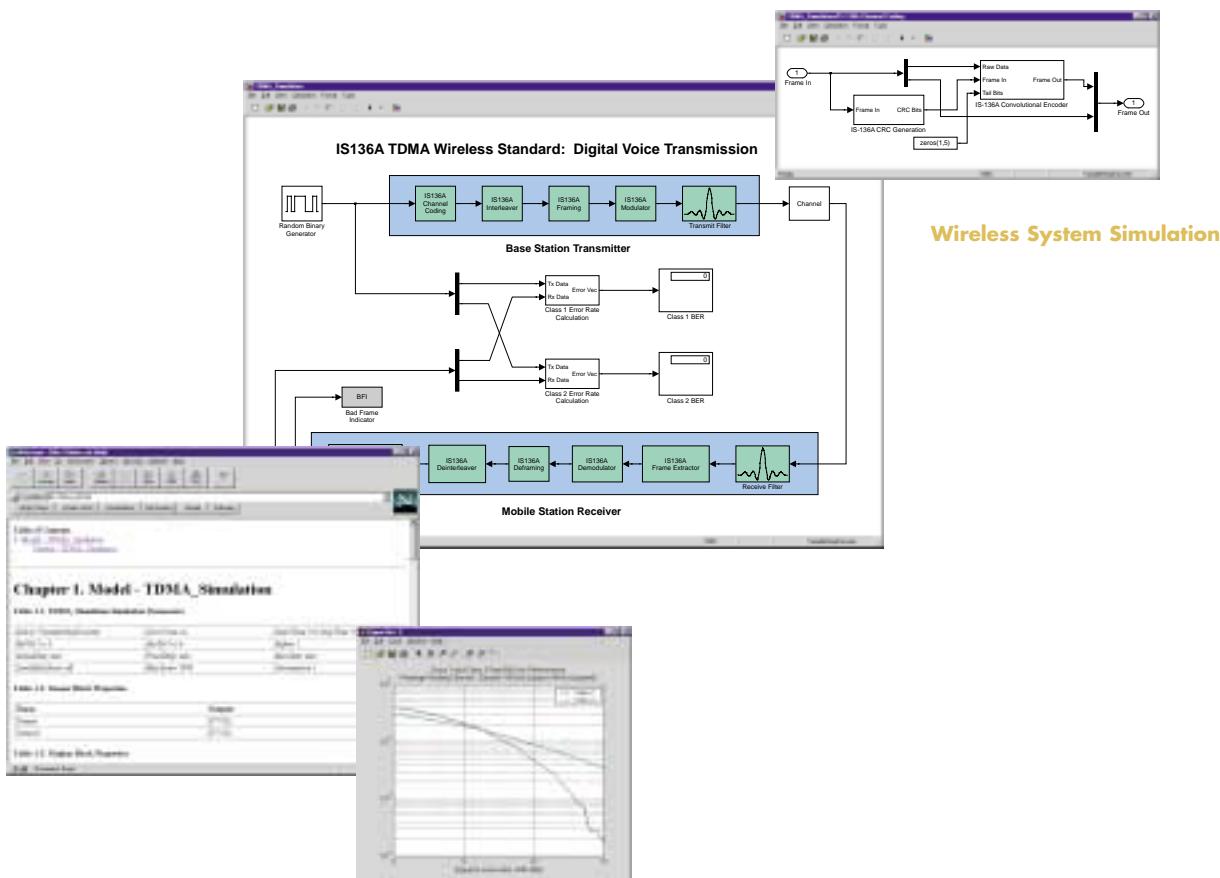
When you use Simulink with Stateflow and the DSP Blockset, you can create fast, accurate behavioral simulations of every element of your system, including real-time

Achieve high-performance simulation

Simulink's high-performance, frame-based DSP processing and state-of-the art continuous time solvers ensure that interactivity doesn't come at the expense of simulation speed. And when you need the optimal performance for batch processing of large data sets and Monte Carlo simulations, you can use Real-Time Workshop to generate standalone executable programs automatically from your Simulink block diagram.

Analyze and optimize with MATLAB

When you use Simulink, MATLAB is always at your fingertips for developing algorithms, exploring and visualizing data, generating test vectors, building custom interactive displays, and creating scripts to automate simulation experiments. You have instant access to dozens of MATLAB toolboxes and hundreds of advanced algorithms, giving you an unparalleled range of tools for analyzing and optimizing your design.



Tools for Real-World, Real-Time Design

When you commit your intellectual property (IP) to HDL design and processor-specific code, you limit your ability to keep pace with the rapid changes in semiconductor and software technology. The Simulink block diagram environment helps you stay ahead by retrieving and reusing your IP—including libraries of real-time DSP algorithms and optimized behavioral models of real-world components. With Simulink, you not only develop, test, and maintain these libraries and models; you also preserve them for use in future products.

efficient, frame-based implementations. The DSP Blockset is ideal for developing real-time speech, audio, and baseband communications algorithms and supports sensor-based signal processing applications.

Accelerating algorithm design

Simulink accelerates the transition from numerical concepts to real-time algorithms and real-world components. Its time-driven simulation engine lets you accurately model the operation of real-world software and hardware and cleanly map your design to an efficient implementation.

Physical layer communications simulation

The Communications Toolbox supplies modulation and channel coding techniques, channel models, and analysis tools for the physical layer design of advanced digital communications systems. The Communications Toolbox works with the

Motorola, Inc.—At the design center of Motorola's Wireless Subscriber System Group, engineers designing mixed-signal Phase-Locked Loop systems needed to speed up their design cycles by cutting simulation run times and improving simulation resolution. Motorola chose Simulink when a benchmark comparison against their mixed-signal circuit simulators proved that Simulink provided the fastest simulation while exceeding resolution specifications.



"The Simulink models

exceeded our project

specifications for

required simulation speed.

Accurate simulations

can now be measured

in minutes rather than

hours or days."

—Yuan Yuan, Motorola, Inc.

The Simulink environment provides a rich set of high-level blocks for defining algorithm behavior, as well as low-level components for building structurally accurate models of your design. To help you build efficient real-time system models, Simulink lets you easily incorporate MATLAB M-files into any Simulink model. You'll find a Simulink or blockset equivalent for virtually every MATLAB signal processing function.

An open, extensible system

Simulink gives you unmatched power to create custom design libraries. Whether you start from built-in blocks or from your own C or MATLAB code, you can present design parameters by giving each block a unique appearance and dialog. And unlike simulations coded only in C, your Simulink models can be instantly reused or shared.

Advanced frame-based DSP simulation

The DSP Blockset provides more than 200 advanced DSP and math functions, including transforms, matrix math, FIR, IIR, adaptive and multirate filters, spectral analysis, and real-time data I/O—all using

DSP Blockset to model all aspects of signal processing in a wide range of systems, including broadband modems, wireless handsets and base stations, and mass storage devices.

Bit-true fixed-point simulation

Using the Fixed-Point Blockset, you can perform bit-true simulations of filters and other signal processing components. The blockset supplies fundamental arithmetic and logical operations, and it lets you control scaling and word length in algorithms designed for fixed-point DSPs, microcontrollers, and ASICs. Simulink's integer and arbitrary user-defined datatypes enable you to cleanly incorporate bit-true C code into Simulink simulations.

Fast, accurate analog and mixed-signal models

Most simulation tools offer only discrete-time approximations of analog behavior. Simulink provides true continuous-time solvers that ensure pinpoint accuracy and rapid simulations. You no longer need to rely on circuit simulations to characterize the performance of nonlinear analog and mixed-signal components such as amplifiers, PLLs, and A/D converters.

Graphical event-driven simulation with Stateflow

Stateflow lets you graphically model event-driven behavior within Simulink. Stateflow simplifies the simulation of protocols, synchronization loops, and control logic signals that activate time-driven datapath subsystems. Together, Simulink and Stateflow provide the only system-level environment that lets you model multiple modes of operation within complex signal processing and communications systems.

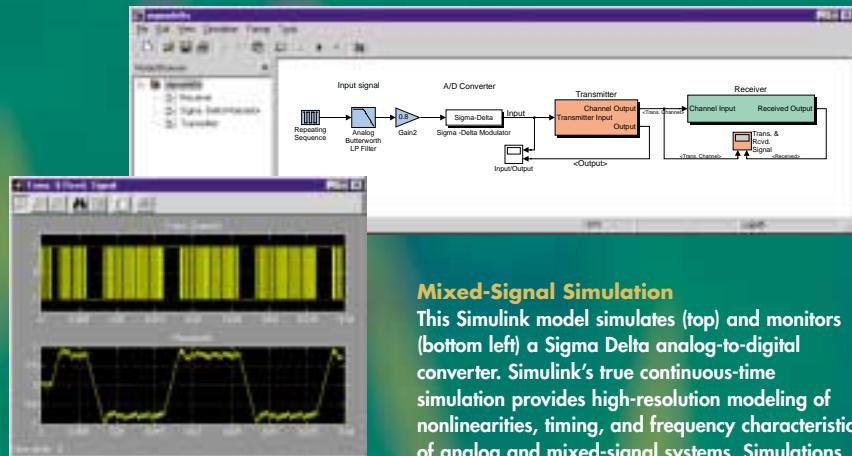
Automatic report generation simplifies teamwork

The MATLAB and Simulink Report Generators automatically generate complete documentation of MATLAB programs and Simulink models in standard formats, including RTF and HTML. These tools ensure an error-free handoff of your design as it progresses from concept to implementation.

Filter Design

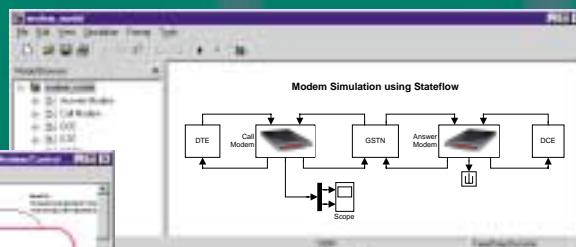
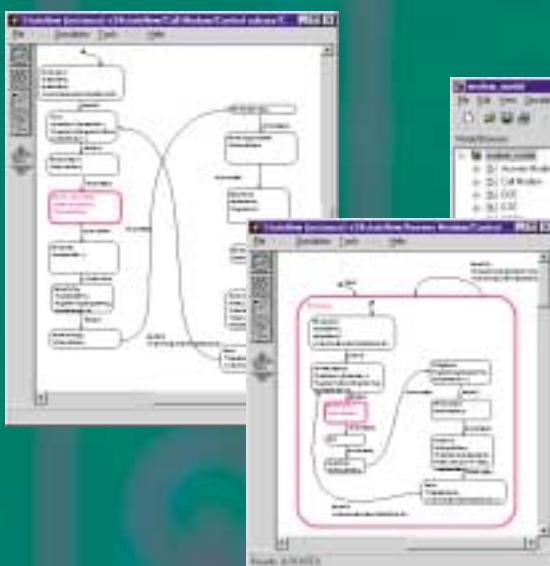
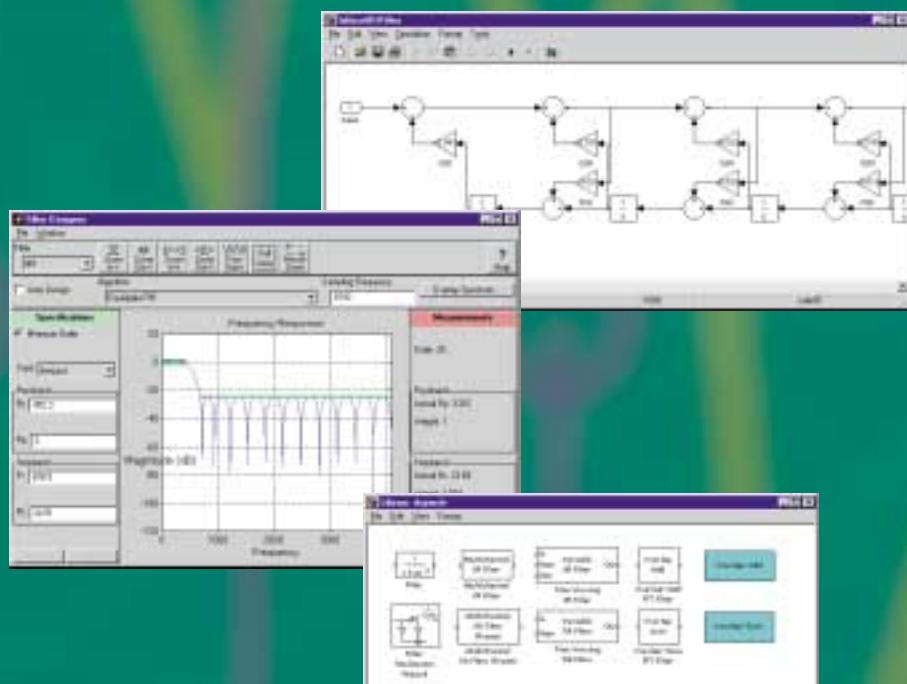
MATLAB and Simulink streamline the creation of efficient real-time filter designs. Simply define filter parameters graphically using the Signal Processing Toolbox's filter designer (center), then drop the equivalent real-time blocks into your Simulink model using the DSP Blockset's filter library (bottom right).

You can even use the DSP Blockset's filter realization wizard (top right) to generate optimized fixed- or floating-point filter structures.



Mixed-Signal Simulation

This Simulink model simulates (top) and monitors (bottom left) a Sigma Delta analog-to-digital converter. Simulink's true continuous-time simulation provides high-resolution modeling of nonlinearities, timing, and frequency characteristics of analog and mixed-signal systems. Simulations run in a fraction of the time taken by circuit simulators and discrete-time system simulators.



Modem Protocol Simulation

Stateflow lets you visually model complex event-driven behavior within a Simulink block diagram. This system simulates a modem protocol exchange (right). As the simulation runs, animated state transitions let you see exactly how the call modem (left) and the answer modem (center) are interacting.

Rapid Prototyping and Design Verification

Test development and design verification can take up two-thirds of the design cycle. Errors in hand-coded designs often slow this already time-consuming process. With its real-time rapid prototyping tools, open co-simulation capabilities, and complementary third-party products, Simulink helps you eliminate these errors, maximizing your programming resources, saving project development time, and reducing design costs.

Efficient code generation and rapid prototyping

The MathWorks code-generation tools—Real-Time Workshop and Stateflow Coder—produce fast, portable C code directly from Simulink and Stateflow models. You can automatically generate code and build real-time executables, giving you an efficient and reliable way to test and iterate your designs on DSP hardware. This approach allows you to make necessary changes before committing your designs to production.

Co-simulation and system verification

With Simulink, it's easy to use your system-level models to verify the specifications of your hardware or software implementation. You can use Simulink's open API to co-simulate with any C-callable instruction set or HDL simulator or to interface with DSP software development tools. The parameterized models, test vector generation, and data analysis capability of Simulink and MATLAB provide a consistent framework in which to regression-test every element of your implementation.

Texas Instruments—TI's goal is to shorten product development cycles by accelerating the flow of real-time DSP algorithms from R&D into product design. Using MATLAB, Simulink, DSP Blockset, and Real-Time Workshop, engineers at the DSP Solutions Research and Development Center refine implementation details directly in a reusable, hierarchical model of the system and produce real-time software prototypes without the need for traditional DSP programming resources.



"Using Simulink helps to tie the two sides of the process together in a way that lets every engineer see the whole project from concept to testing to coding."

—Dr. Randy Cole, Texas Instruments

You can refine your model, generate code, and evaluate results on your target hardware—all within minutes. This rapid prototyping approach helps you produce more effective designs without wasting months on error-prone, manual DSP programming.

Customize and optimize generated code

Real-Time Workshop and Stateflow Coder generate readable, well documented code, letting you easily identify and modify time-critical algorithms to fit the real-time constraints of your target environment.

Using the Target Language Compiler, you can optimize the generated code and include hand-coded routines automatically at code-generation time. This feature makes it easy to create and reuse processor-specific algorithms and tailor code to meet specific application requirements. As a result, you can focus your valuable programming resources on the implementation of critical components.

Additional solutions from our Connections partners

Our MATLAB Connections partners provide simulators, development hardware, real-time operating systems, and application-specific tools that complement the MathWorks DSP and communications design environment. Visit www.mathworks.com for an up-to-date list of Connections partners and available products.

About The MathWorks

Founded in 1984, The MathWorks, Inc., has grown steadily into a company of more than 500 people with over 400,000 users of its software worldwide. The MathWorks diversified product family provides powerful, tailored computational tools for engineers, scientists, and mathematicians in over 100 countries on all seven continents. These technical people work at the world's most innovative technology companies, government research labs, and financial institutions, and at more than 2,000 universities. They rely on The MathWorks because MATLAB and Simulink are the fundamental tools for their engineering and scientific work.

MathWorks Web site

Our Web site, www.mathworks.com, offers a wealth of information on MathWorks products and services. You can get product information, download products, access technical support, and find out about MathWorks-sponsored events.

Services and support

We provide an extensive service and support network through The MathWorks and our international contacts around the globe. Whether you are requesting product information, signing up for a training course, or in need of technical support, you can be assured of an immediate response to your inquiry. Visit www.mathworks.com for more information on these services and programs.

Training

The MathWorks offers training courses designed to help users of all levels of experience become more proficient with The MathWorks software tools. Each course uses the latest software and is taught by a MathWorks engineer with extensive teaching experience. We'll work with you to assess your company's training needs and help you select the best option.

MATLAB Access program

The MATLAB Access program is a free service that provides our customers with a direct link to a number of benefits, including personalized, Web-based self-service resources, early notification of product releases, technical support, and customer service. MATLAB Access members are also eligible to receive our quarterly electronic newsletter, *MATLAB Digest*, and our product newsletter, *MATLAB News & Notes*.



The MathWorks, Inc.

Web: www.mathworks.com

E-mail: info@mathworks.com

Tel: 508-647-7000

Fax: 508-647-7101

MATLAB based books

More than 200 books based on MATLAB and Simulink have been published in a variety of languages. Many books have companion software written for MATLAB and Simulink. The directory *MATLAB Based Books* is available on our Web site and in printed form.

MATLAB newsgroup

Participants from around the world discuss MathWorks products, solve problems, and share code in an unmoderated Usenet newsgroup, *comp.soft-sys.matlab*. The MathWorks follows the newsgroup's activities closely, providing answers to technical queries and posting announcements of general interest.

MathWorks ftp site

The MathWorks maintains a software library, ftp.mathworks.com, that contains hundreds of free, user-contributed MATLAB programs. This archive allows users to share their efforts with others working in similar application areas.



MATLAB® 6

For numeric computation, visualization, algorithm development, and programming

MATLAB PRODUCTS OVERVIEW

MATLAB® is a high-performance environment for applications in engineering and science that includes tools for mathematical computation, analysis, visualization, and algorithm development. The intuitive MATLAB language—available in both interactive and runtime modes—enables technical professionals to express their ideas and solutions naturally and faster than they could with C and other traditional programming languages.

MATLAB environment

The MATLAB desktop front end includes a portfolio of tools for accessing MATLAB features and functions:

- Command Window for interactive analysis, visualization, and programming
- Command History for viewing and reusing commands from previous sessions
- Launch Pad for accessing demos, help, and tools for all installed MathWorks products
- Help to read and search documentation, including cutting and pasting sample code
- Current Directory browser to open, run, and view files
- Workspace Browser to view and change MATLAB data variables
- Array Editor to view and edit array data
- M-file Editor/Debugger to create, edit, and debug MATLAB files

Additional interactive tools are available for importing data, performing basic curve-fitting operations, and editing, viewing, and annotating graphics.

Help and documentation

Built-in help browser with search facility, online documentation (PDF format), examples, and demonstrations. Viewing or printing of PDF-formatted documentation requires Adobe Acrobat Reader (available on the MATLAB CD).

Supported operating systems

Windows 95/98/2000/NT; UNIX: Solaris, HP-UX, IRIX, IRIX64, AIX, Digital UNIX; Linux. MATLAB is also compatible with operating systems that support multibyte characters.

Supported computer platforms

Intel 486 or Pentium PCs, SunSPARC, SunULTRA, DEC Alpha, SGI, IBM RS/6000

Multiplatform interoperability

MATLAB applications are fully transportable across platforms without modification. MATLAB data files (MAT-files) from different environments are converted automatically.

MATLAB toolboxes

Communications, Control System, Data Acquisition, Database, Datafeed, Filter Design, Financial, Financial Derivatives, Financial Time Series, Fuzzy Logic, GARCH, Image Processing, Instrument Control, LMI Control, Mapping, Model Predictive Control, μ -Analysis and Synthesis, Neural Network, Optimization, Partial Differential Equation, Robust Control, Signal Processing, Spline, Statistics, Symbolic Math, System Identification, Wavelet

MATLAB compatible products

Excel Link, MATLAB Compiler, MATLAB C/C++ Math Library, MATLAB C/C++ Graphics Library, MATLAB Report Generator, MATLAB Runtime Server, MATLAB Web Server

Simulink®

Simulink® is an interactive environment for modeling, simulating, and analyzing dynamic systems. Built on top of MATLAB, Simulink offers immediate access to an extensive range of tools for model-based and system-level design.

Simulink blocksets

CDMA Reference, Communications, Dials & Gauges, DSP, Fixed-Point, Nonlinear Control Design, Power System

Simulink compatible products

Motorola DSP Developer's Kit, Real-Time Workshop®, Real-Time Workshop Ada Coder, Real-Time Workshop Embedded Coder, Real-Time Windows Target, Requirements Management Interface, Simulink Performance Tools, Simulink Report Generator, Stateflow®, Stateflow Coder, xPC Target

NUMERIC COMPUTATION

Matrix operators

Add, subtract, multiply, power, left and right divide, transpose, conjugate transpose, Kronecker tensor product

Array operators

Add, subtract, multiply, left and right divide, power, transpose, conjugate transpose

Relational operators

Equal to, not equal to, less than, greater than, less than or equal to, greater than or equal to

Logical operators

AND, OR, NOT, exclusive OR (XOR)

Bit operations

AND, OR, XOR, complement, maximum floating-point integer, set bit, get bit, bit-wise shift

Set operators

Union, unique, intersection, difference, XOR

Elementary matrices

Zeros, ones, identity, uniformly and normally distributed random numbers, linearly and logarithmically spaced vectors, diagonal matrix

Special matrices

Companion, Hadamard, Hankel, Higham, Hilbert, inverse Hilbert, magic square, Pascal, Rosser, Toeplitz, Vandermonde, Wilkinson

Matrix manipulation

Extract diagonal; flip left/right; flip up/down; reshape; rotate; permute; extract lower and upper triangular part; index into matrix; concatenate; select elements, columns, and submatrices; last index; find indices of nonzeros

Elementary functions

Sin, cos, tan, sec, csc, cot, sinh, cosh, tanh, sech, csch, coth, asin, acos, atan, 4-quadrant atan, asec, acsc, acot, asinh, acosh, atanh, asech, acsch, acoth, ceil, fix, floor, round, rem, abs, angle, sqrt, exp, natural log, log base-10, log base-2, signum, modulus, complex conjugate, real part, imaginary part

Specialized math functions

Airy, Bessel, Hankel, beta, incomplete beta, log beta, Jacobi elliptic, complete elliptic integral, error function, complementary error function, scaled complementary error function, inverse error, exponential integral, gamma, incomplete gamma, log gamma, rational approximation and output, Legendre, cross product, least common multiple, greatest common divisor, factorial, prime factors, prime numbers, all possible permutations, coordinate transforms

Numerical linear algebra

Condition number, reciprocal condition estimate, norm, rank, determinant, trace, null space, orthogonalization, reduced row echelon form, linear equation solution, Cholesky factorization, LU factorization, matrix inverse, QR decomposition, non-negative and known covariance least-squares, pseudo inverse, eigenvalues, eigenvectors, characteristic polynomial, Hessenberg form, block form conversions, Schur decomposition, balancing, singular value decomposition, matrix exponential, matrix logarithm, matrix square root

Statistics

Maximum, minimum, mean, median, standard deviation, variance, sort, sum, product, cumulative sum, cumulative product, histogram, numerical integration, difference functions, gradient, Laplacian, correlation coefficients, covariance matrix

Signal processing

1-D and 2-D digital filter; 1-D, 2-D, and multidimensional (N-D) fast Fourier transform (FFT), inverse FFT, and convolution; deconvolution; magnitude; phase angle; phase unwrap

Interpolation

1-D, 2-D, and N-D table lookup and interpolation (methods: linear, cubic, nearest, and spline), 1-D interpolation using FFT method, 1-D piecewise cubic Hermite interpolating polynomial, data gridding, hypersurface fitting

Geometric analysis

2-D, 3-D, and N-D Delaunay triangulation; 2-D and N-D search triangulation for nearest point, closest triangle search, convex hull, Voronoi diagram; rectangle intersection area; area of polygon

Polynomials

Roots, characteristic polynomial, piecewise polynomial, evaluate, evaluate with matrix, partial-fraction expansion, polynomial fit, differentiate, integrate, multiply, divide

Differential equations

Initial value problems: one-step and multistep, low, medium, and variable order solvers for nonstiff and stiff ordinary differential equations (ODEs) and differential-algebraic equations (DAEs)

Time-series, 2-D and 3-D phase plane plots; event location capability; problems involving mass matrices (time- and state-dependent, sparse, singular); consistent initial conditions for DAEs

Boundary value problems: two-point boundary value problems (BVPs) for ODEs; collocation method; general, nonseparated boundary conditions; determining unknown parameters

Partial differential equations (PDEs): initial-boundary value problems for systems of parabolic and elliptic PDEs in 1-D

Sparse matrix operations

Sparse identity matrix; uniform, normally distributed, and symmetric random matrix; diagonal matrix; conversion to full; conversion to external format; nonzero elements and indices; replace nonzeros with ones; allocate memory for nonzeros; test if sparse; visualize sparse pattern; graph theory plot; column and symmetric minimum degree permutation; column and symmetric approximate minimum degree permutation; symmetric indefinite; reverse Cuthill-McKee, Dulmage-Mendelsohn, column, and random permutation; norm and condition estimate; rank; symbolic factorization analysis; least-squares; least-squares augmented system; selective eigenvalues and singular values; incomplete LU and Cholesky factorization; estimation of matrix 2-norm

Iterative methods for sparse linear equations: preconditioned conjugate gradients, biconjugate gradients, biconjugate gradients stabilized, conjugate gradients squared, conjugate gradients on the normal equations via LSQR, minimum residual, generalized minimum residual, quasi-minimal residual, symmetric LQ

Nonlinear numerical methods

1-D and 2-D integral evaluation (methods: Simpson's rule and Lobatto rule), minimize function of several variables, find zero of function, plot function

Time and date

Serial date number, date string, date vector, current date and time (number, string, and vector), calendar, day of week, end of month, date formatted tick labels, wall clock, CPU clock, date, elapsed time, stopwatch, wait time

Miscellaneous variables and constants

Most recent answer, machine epsilon, largest number, smallest number, pi, i, j, infinity, NaN, computer type

GRAPHICS AND VISUALIZATION

2-D graphics

Linear plot, loglog plot, semilog plot, scatter plot, filled-area plot, polar plot, vertical and horizontal bar graph, stem plot, pie chart, staircase plot, error bars, histogram, angle histogram, Pareto chart, stem plot, compass plot, comet plot, feather plot, quiver plot, function plot

3-D graphics

Line plot, filled polygon, contour plot, ribbon plot, stem plot, comet plot, scatter plot, pseudocolor plot, quiver plot, mesh surface plot, triangular mesh and surface plot, combination mesh and contour plot, parametric surface contour, mesh plot with zero plane, vertical and horizontal bar graph, pie chart, shaded surface, combination surface and contour plot, waterfall plot, cylinder, sphere, ellipsoid, patch

Surface and patch properties

Vectorized patch, face coloring and lighting, edge coloring and lighting, surface or wire frame, hidden line removal mode, mesh style (row, column, or both), line style, line width, texture mapping, marker style, marker face and edge color, marker size, face and edge transparency and visibility

Volume and vector visualization

Isosurface extraction, isosurface normals, isosurface end caps, contours in slice planes, streamlines from 2-D or 3-D vector data, 3-D cone plot, extract subset of volume dataset, reduce volume dataset, smooth 3-D data, reduce number of patch faces, reduce size of patch faces, convert surface data to patch data, 2-D and 3-D quiver plot, vector field divergence, curl and angular velocity of vector field, streamtube, streamribbon, streamslice, stremparticle, interpolate streamline vertices from speed

Image display and file I/O

Display image (indexed, grayscale, and RGB), display data as image with arbitrary scaling, display image with color calibration scale, set colormap, read image from file, read header information from image file, write image to file

Supported file formats: TIFF, JPEG, GIF, BMP, PNG, XWD, PCX, HDF, HDF-EOS, CUR, ICO

Color and rendering

Built-in and user-definable colormaps, pseudocolor axis scaling, shading, lighting, HSV/RGB conversion, brighten, spin colormap, plot colormap, custom pointers, transparency

Renderers: Painters, Z-buffer, OpenGL

Lighting control

Face and edge lighting models (Phong, Gouraud, and flat), reflectance properties (ambient, diffuse, and specular), create light object, light position, direction, color, material reflectance (shiny, dull, metal, and user-defined)

Camera control

Perspective and orthographic projection; 3-D data aspect ratio; transformation matrix; camera position, target, upvector, viewpoint, and angle control; orbit, pan/tilt, dolly, zoom, roll, walk, or interactively move camera

Camera Toolbar allows you to interactively control motion, select axis, set scene light, and set projection type.

Graphical object control

Create figure, axes, tiled axes, line, text, patch, rectangle, surface, image, user interface control, user interface menu, user interface context menu

Get current figure and axes handles, close figure, clear figure, clear axes, control axis scaling, control pseudocolor axis scaling, hold graph, get and set object properties, reset object properties, delete object, flush pending graphics events, modal figures, print graph, save graph to file, set paper orientation and position

Axis control

Position, limits, and units; axis direction; tick mark spacing, direction, length, and labels; scaling (log/linear); pseudocolor axis scaling; grid lines; grid line style; color; visibility; axis color; current point; 3-D axis aspect ratio; zoom; camera position; label font style and size

Graph annotation

Title, axes labels, text annotation, grid lines, colorbar, legend, contour plot elevation labels, LaTex-style subscripts and superscripts, mixed fonts and sizes, multiline text, Greek symbols

Animation

Get movie frame, play recorded movie, convert frame to indexed image, convert indexed image to movie frame, read/write AVI movie files

Sound

Play sound, read/write Sun audio (.au) file, convert linear to μ -law encoding, convert μ -law encoding to linear, read/write 8- or 16-bit WAV files, play sound using Windows output device, record sound using Windows input device

Exportable graphics format

Encapsulated PostScript (EPS), EPS with TIFF preview, Adobe Illustrator EPS, Enhanced Windows metafile, Windows bitmap (BMP), HDF, PCX, PPM, JPEG, PNG, HPGL, and TIFF

PROGRAMMING

Data types

Complex double-precision; single-precision; unsigned and signed 8-, 16-, and 32-bit integers; character/string; sparse; cell arrays; structures; objects; multidimensional; logical; empty; user-defined; function handle array; Java array; Java object

Multidimensional arrays

Create, assign, index, concatenate, reshape, number of dimensions, permute, shift dimensions, remove singleton dimensions, flip along specified dimension, pointwise elementary math operations, interpolation, data gridding

Cell arrays and structures

Create, reshape, replace lists of variables, apply functions and operators, nest, display, display cells graphically, get field names, get and set field contents, remove field, convert to double, numeric

Object-oriented programming

Create object class, display class method names, convert object to structure array, loading prebuilt JAVA objects

Overloadable operators for arrays and matrices: add, subtract, multiply, left/right divide, power, equal to, not equal to, less than, greater than, less than or equal to, greater than or equal to, AND, OR, NOT, XOR, subscripting, transpose, colon, concatenation, assignment, referencing, indexing

Programming constructs and argument handling

If, else, elseif, end, for, while, break, continue, return, error, warning, global and persistent variables, switch, case, otherwise, try, catch, evaluate, run, validate number of input and output arguments, number of function input and output arguments

File I/O

Open file, close file, read/write binary data, read/write formatted data, read line, inquire status, get/set file-position indicator, rewind, seek to any relative or absolute file position, read/write formatted string, read/write ASCII delimited file, read/write spreadsheet (.wk1) file, read spreadsheet (.xls) file, read/write

image to/from graphics file, read/write Sun (.au) sound file, read/write Windows (.wav) sound file, read movie (.avi) file, create movie (.avi) file, read/write via serial port, gateways to HDF and HDF-EOS libraries, load Handle Graphics® objects from file, append to binary/text file

Import Wizard allows you to interactively import ASCII text data, AVI, GIF, CUR, HDF, ICO, JPEG, MATLAB MAT, PNG, WAV, AU, SND, CSV, XLS, WK1, and PCX files

String manipulation

Convert number, integer, or matrix to string; convert string to double precision value or numeric array; convert binary and base B to decimal integer; test for string; execute; compare; convert to uppercase or lowercase; convert hexadecimal to IEEE floating-point number or decimal integer; convert decimal to hexadecimal, binary, or base B string; search and replace; identify first token in a string; concatenate; find one string in another; replace string; find match; remove blanks

Graphical user interface (GUI) development

GUI controls: list boxes, pull-down menus, push buttons, radio buttons, check boxes, pop-up menus, editable textboxes, fixed textboxes, sliders, frames, context menus

GUI events: wait for event, mouse-button events, callbacks, rubberband box, mouse selection

Dialog boxes: generic with user-specifiable icon, platform-specific load/save file, input, list, warning, error, help, print, page position/setup, question, message, open/close file, color, font selection

Interactive GUI design interface (GUIDE) includes layout editor, alignment tool, callback editor, property editor, menu editor, and object browser

M-file programming tools

Integrated M-file Editor/Debugger, M-file execution profiler, profiler textual and graphic summary reports in HTML, M-file to P-file (pre-parsed pseudo-code to maintain privacy) converter, list functions in memory, save MATLAB session text, access source control systems

Debugging

Interactive editor and debugger, set and clear breakpoints, continue execution, change workspace context, display stack, show status, single and multiple line step, list file with line numbers, exit debug mode

Files and operating system

List directory, list MATLAB specific files, display M-file contents, edit M-file, search all M-files for keyword, change working directory, print working directory, delete file, get environment variable, execute operating system command, time, date, open Web browser

External interfaces

MATLAB provides interfaces to external protocols, applications, and languages such as C, C++, Fortran, and Java. These interfaces allow you to:

- Transparently call and exchange data with C, C++, Fortran, and Java routines directly from MATLAB
- Communicate and exchange data with devices and instruments via the serial communications port
- Call MATLAB from C and Fortran programs, using it as a math and graphics engine
- Use ActiveX components and DDE (Dynamic Data Exchange)
- Incorporate prebuilt Java objects into MATLAB applications

The MATLAB Notebook (Windows only)

The Notebook combines the word processing of Microsoft Word with the numeric computation and graphics of MATLAB to create live technical reports, electronic class notes, and homework assignments incorporating any MATLAB commands, data, calculations, or graphics.

Hardcopy printers and plotters

PostScript printers, color PostScript printers, Level-2 PostScript printers, HP LaserJet, HP DeskJet, HP PaintJet, Epson 9-pin printers, Epson 24-pin printers, HP 7475A plotter, QuickDraw printers

Desktop publishing support

Supports Microsoft Word, PowerPoint, TeX, QuarkXPress, FrameMaker, PageMaker, and other standard desktop publishing software

For demos, application examples, tutorials, user stories, and pricing:

- Visit www.mathworks.com

- Contact The MathWorks directly

US & Canada 508-647-7000

Benelux	+31 (0)182 53 76 44
France	+33 (0)1 41 14 67 14
Germany	+49 (0)241 470 750
Spain	+34 93 362 13 00
Switzerland	+41 (0)31 954 20 20
UK	+44 (0)1223 423 200

Visit www.mathworks.com to obtain contact information for authorized MathWorks representatives in countries throughout Asia Pacific, Latin America, the Middle East, Africa, and the rest of Europe.

Simulink® 4

for modeling, simulation, and analysis of dynamic systems

Simulink® is an interactive tool for modeling, simulating, and analyzing dynamic systems. It enables you to build graphical block diagrams, simulate dynamic systems, evaluate system performance, and refine your designs. Simulink integrates seamlessly with MATLAB®, providing you with immediate access to an extensive range of analysis and design tools. These benefits make Simulink the tool of choice for control system design, DSP design, communications system design, and other simulation applications.

Creating Models

Simulink provides a complete set of modeling tools that you can use to quickly develop detailed block diagrams of your systems. Features such as block libraries, hierarchical modeling, signal labeling, and subsystem customization provide a powerful set of capabilities for creating, modifying, and maintaining block diagrams. These modeling features, together with Simulink's comprehensive set of predefined blocks, make it easy to create concise representations of your systems, regardless of their complexity.



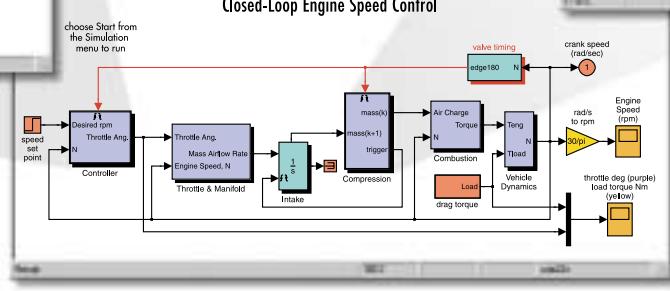
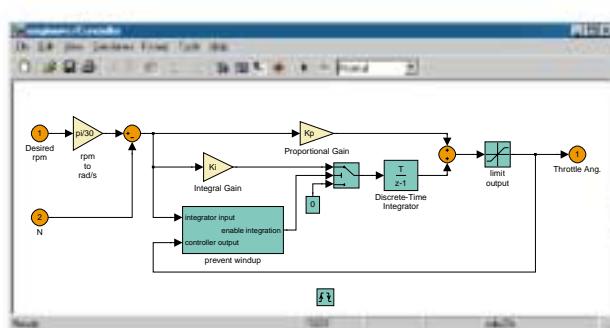
KEY FEATURES

USABILITY

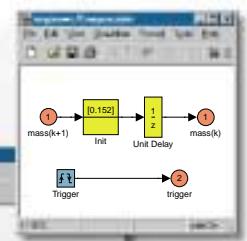
- Extensive library of predefined blocks
- Graphical debugger
- Model Browser for navigating model hierarchies
- Finder for searching models and libraries
- Customizable blocks that can incorporate existing C, Ada, MATLAB, and Fortran code

COMPUTATIONAL SUPPORT

- Linear, nonlinear, continuous-time, discrete-time, multirate, conditionally executed, mixed-signal, and hybrid systems
- Support for matrix signals and operations
- Bitwise Logical Operator block logically masks, inverts, or shifts the bits of an unsigned integer signal



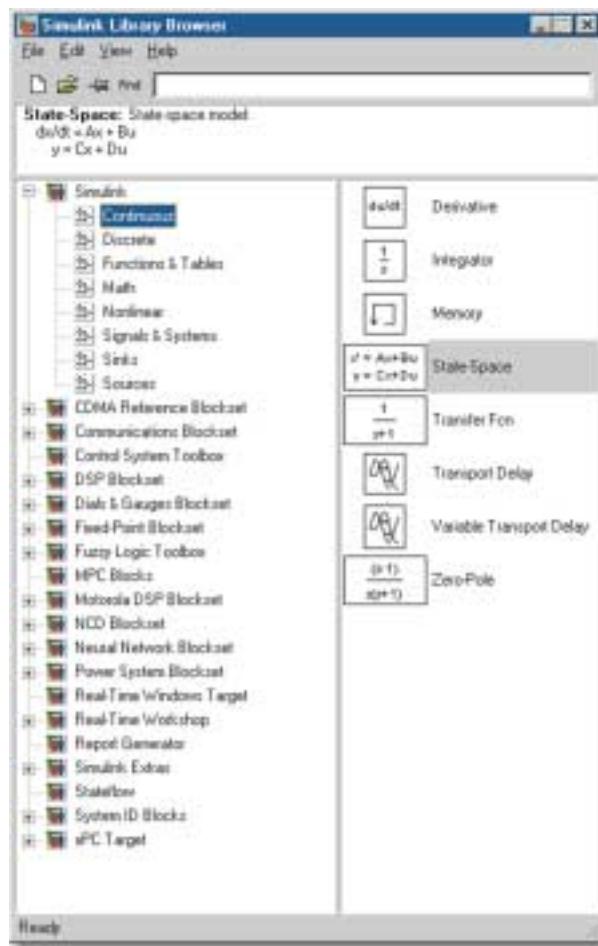
An engine model uses Trigger blocks to model conditionally executed behavior. As a function of the crankshaft angle, a pulse triggers a cylinder to fire.



KEY FEATURES (continued)

LARGE MODEL DEVELOPMENT

- Models can be grouped into hierarchies to create a simplified view of components or subsystems
- Simulink data objects enable you to create application-specific MATLAB data types for your Simulink models
- Simulink Explorer GUI for viewing and editing data objects
- Library Browser for convenient block selection
- Intellectual property protection using S-functions (requires Real-Time Workshop® 4.0)
- Simulations can be run from the MATLAB command line, either interactively or in batch mode



Extensible Block Library

Simulink comes with more than 200 built-in blocks that implement commonly required modeling functions. The blocks are grouped into libraries according to their behavior: Sources, Sinks, Discrete, Continuous, Nonlinear, Math, Functions & Tables, and Signals & Systems.

In addition, Simulink offers features for creating customized blocks and block libraries. You can customize not only the functionality of a block, but also its user interface, using icons and dialog boxes. For example, you can create blocks to model the behavior of specialized mechanical, circuit, or software components, such as motors, inverters, servo-valves, power plants, filters, tires, modems, receivers, or other dynamic components. Custom blocks can be saved in your own block library for future use and can be shared with work groups, vendors, and customers.

The Library Browser makes it easy to navigate through block libraries and then drag and drop selected blocks onto your model.

S-Functions

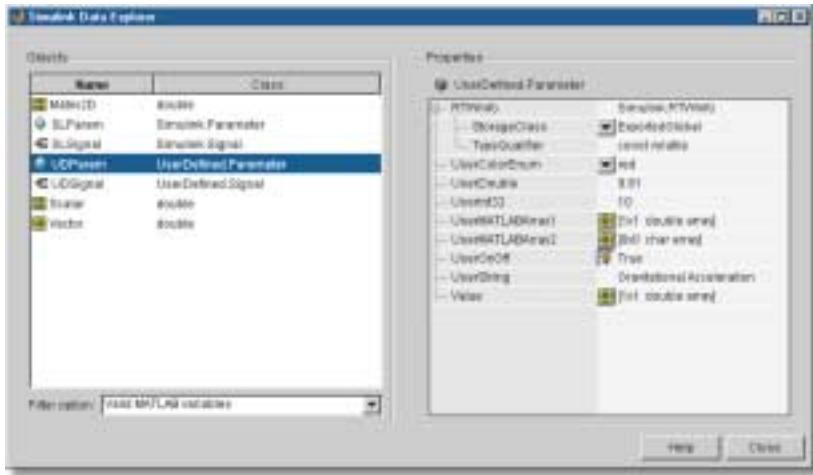
An S-function (system-function) is a custom code module that defines the behavior of a Simulink block. Simulink provides templates for creating your own S-functions using existing or newly-developed code (C, Ada, Fortran, or MATLAB). Once you have created an S-function, you can include it in your model, using Simulink's S-function block.

S-functions reduce the time required to model large-scale systems by allowing you to incorporate existing code into your

model. Simulink provides multi-port and multi-rate S-function support to enhance usability and permit different sample times (C and MATLAB only).

Masks

Simulink's mask editor allows you to create a custom user interface, called a mask, for any subsystem or S-function block. The mask can include a custom icon, parameter dialog, online help, and initialization script. Custom masks allow you to tailor a block's appearance and user interface for specific applications.



The Simulink Explorer provides you with a graphical user interface for viewing and editing Simulink data objects. Using the Simulink Explorer, you can view most classes of variables in the MATLAB workspace, and filter and sort variables by variable name and class. You can also view and edit property values.

Simulink Data Objects

Simulink is used in many applications to create high-fidelity plant models of real-world systems and to design algorithms to control these systems. To represent these systems and algorithms more accurately, you can use Simulink data objects to define new MATLAB data types that are specific to your application and then use them as parameters and signals in your Simulink

models. You can view and edit all Simulink data with the Simulink Explorer.

Model Library Support

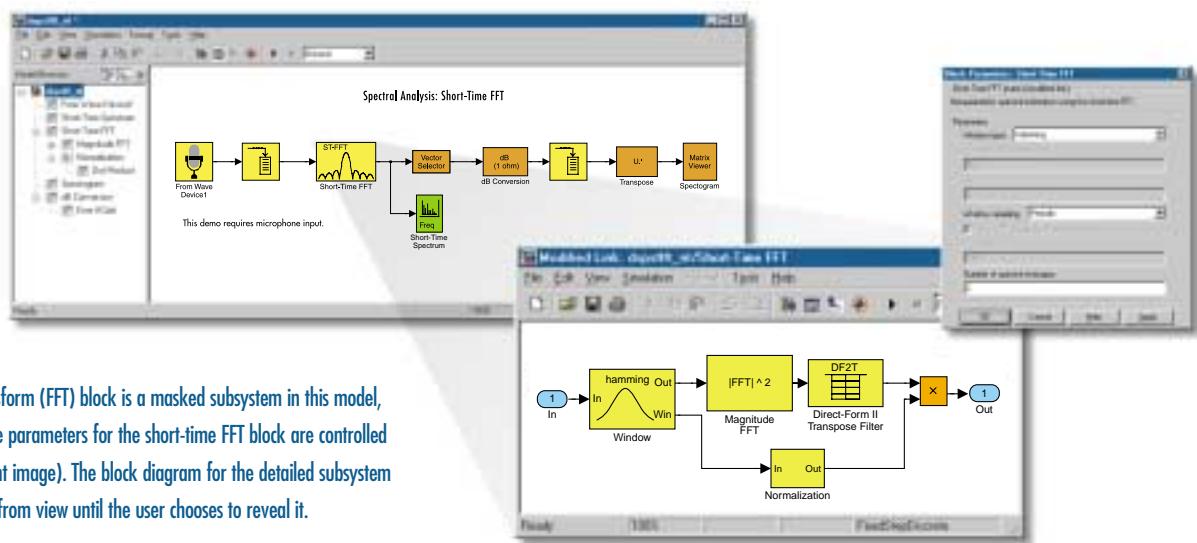
Model library support makes it easy to build and maintain libraries of customized blocks. You can create a block whose properties are defined in the model library. Then, when you make a change to the library version of the block, the change propagates through any models that use that block.

This feature makes it easy to reuse blocks across multiple systems, as well as systems with large numbers of models, and models with many levels. You can modify a block's behavior and its attributes in every model simply by applying the change to the library source.

Configurable Subsystem Block

A Configurable Subsystem block represents any block contained in a specified library of blocks. Using the Configurable Subsystem block's dialog box, you can specify which block in the library it represents. You can also specify the inputs and outputs of the selected block.

Configurable Subsystem blocks simplify the creation of models that represent families of designs. For example, suppose that you want to model an automobile that offers a choice of engines. To model such a design, you would first create a library of models of the engine types available with the car. You would then use a Configurable Subsystem block in your car model to represent the choice of engines. To model a particular variant of the basic car design, you need only choose the engine type, using the configurable engine block's dialog. This enables you to rapidly swap design choices in and out of your model.



The short-time fast Fourier transform (FFT) block is a masked subsystem in this model, built using the DSP Blockset. The parameters for the short-time FFT block are controlled through the dialog box (top right image). The block diagram for the detailed subsystem (center image) remains hidden from view until the user chooses to reveal it.

Model Navigation Tools

Library Browser (Windows only)—provides a tree-structured view of all block libraries installed on your system.

Model Browser (Windows only)—enables you to navigate your model hierarchically, and open systems directly in your model.

Finder dialog box—enables you to search Simulink models for objects that satisfy specified search criteria.

Block diagram zooming—greatly simplifies model viewing by allowing you to enlarge or shrink the view.

Scalar and Vector Connections

Simulink supports the modeling of single-input/single-output (SISO) and multi-input/multi-output (MIMO) systems.

The Mux block is used to collect multiple signals into a vectored signal bundle that can function as a data bus. The Demux block is used to disassemble vectored signals so that they can be accessed as individual signals. The Bus Selector block provides support for larger models by making it easy to select a subset of signals from a bus defined by a Mux or another Bus Selector block.

Because most Simulink blocks support vectored operations, you can greatly reduce the number of blocks needed to model your system. This results in clean, simple, and easy-to-read block diagrams.

Matrix Signal Support

Many Simulink blocks accept or output matrix signals. A matrix signal is a two-dimensional array of signal elements

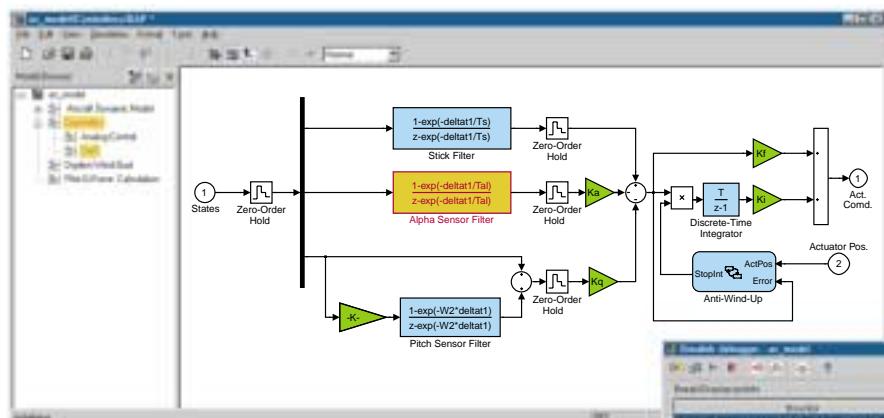
represented by a matrix. Each matrix element represents the value of the corresponding signal element at the current time step. You can use Simulink source blocks (for example, Sine Wave or Constant) to generate matrix signals.

You can use the following Simulink blocks for matrix operations on matrix signals:

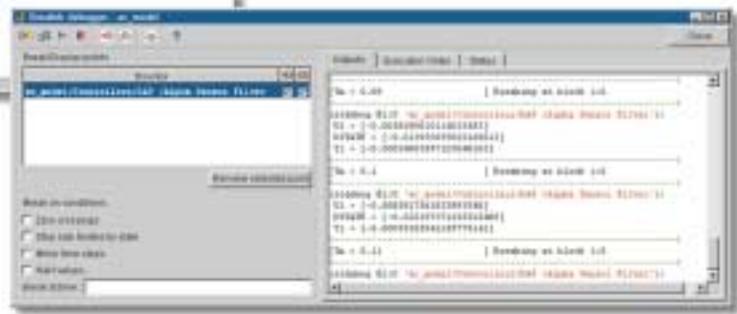
- The Product block supports both element-by-element and matrix multiplication and inversion of inputs.
- The Gain block supports matrix and element-by-element multiplication of the input signal by a gain factor. Both input signals and gain factors can be matrices.

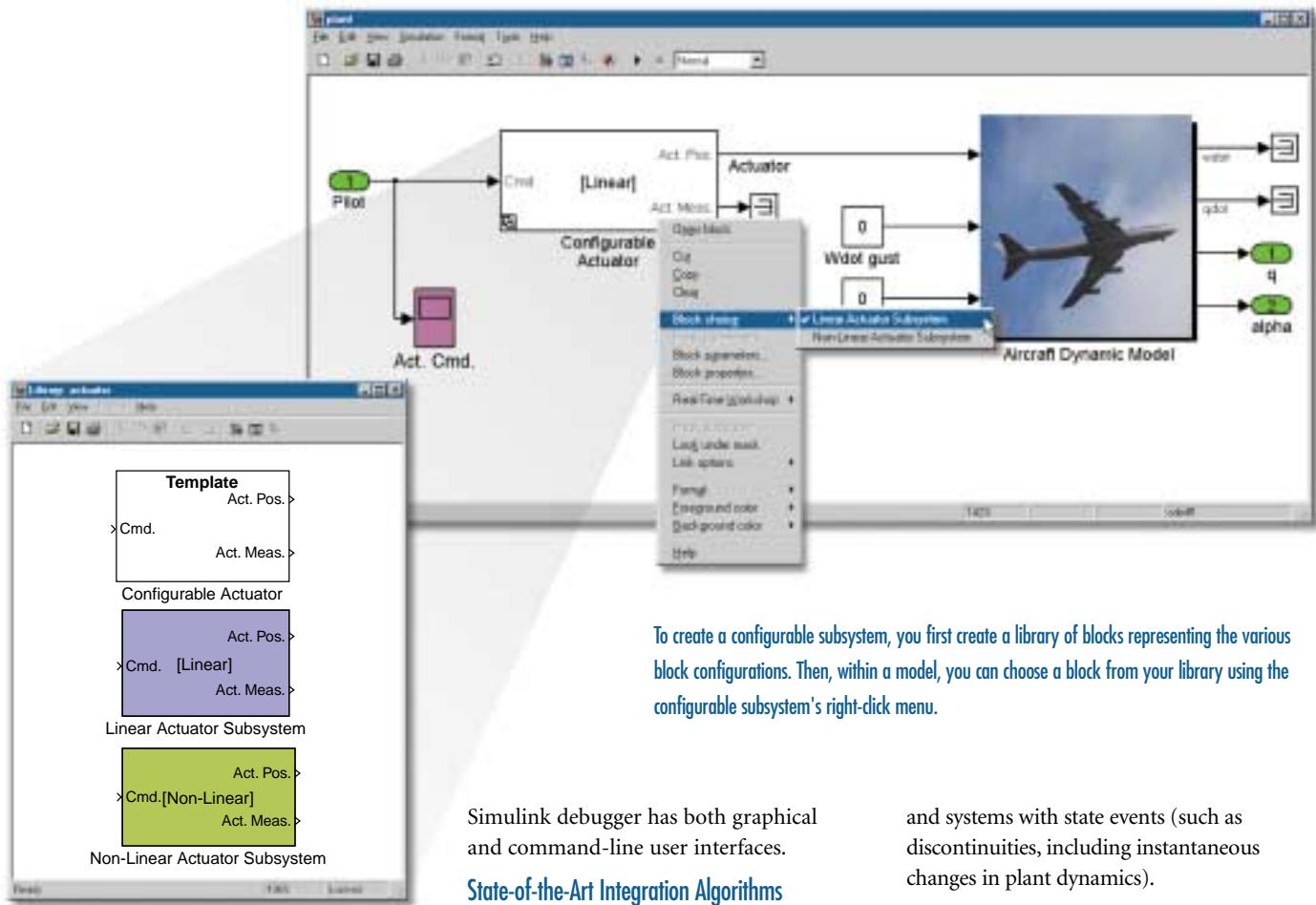
You can use Simulink's Mux and Demux blocks to multiplex matrix signals. For example, you can:

- Generate signal buses by feeding matrix signals into Mux blocks along with vector or scalar signals
- Manipulate the elements of a signal bus by splitting it into its components using a Demux block, and then connecting the demuxed signals to nonvirtual blocks, such as the Gain block



This Simulink model represents a digital control system for an aircraft. The Simulink debugger allows you to graphically diagnose modeling errors. The debugger lets you step through the simulation block by block, or run to a breakpoint. The currently executing block is displayed in yellow. You can also display block states, block inputs and outputs, and other information while running a model.





To create a configurable subsystem, you first create a library of blocks representing the various block configurations. Then, within a model, you can choose a block from your library using the configurable subsystem's right-click menu.

Simulink debugger has both graphical and command-line user interfaces.

State-of-the-Art Integration Algorithms

The Simulink simulation engine offers numerous features for simulating large, challenging systems. Foremost among these is the set of integration algorithms, called solvers, that are based on the MATLAB ordinary differential equation (ODE) suite. These solvers are well suited to continuous-time (analog), discrete-time, hybrid, and mixed-signal simulations of any size. In addition, they provide fast, reliable, and extremely accurate simulation results. For complete handling of discrete systems, the DSP Blockset is also recommended.

The solvers support differential algebraic equations (DAEs) with multichannel algebraic loops. An algebraic constraint block facilitates the solution of a system in which an algebraic constraint applies to the governing set of equations. The solvers also support stiff systems, systems with algebraic loops,

and systems with state events (such as discontinuities, including instantaneous changes in plant dynamics).

Conditionally Executed Subsystems

With Simulink, you can build and simulate models with subsystems that execute conditionally; and are therefore dependent upon controlling logic signals. The signals can either enable or trigger the execution of the subsystem.

Two blocks, the Trigger block and the Enable block, can be placed in any Simulink subsystem. An enabled or triggered subsystem includes an additional input signal to handle controlling logic.

When conditionally executed subsystems are disabled they are not executed during the simulation, which noticeably improves processing speed within multimode systems.

Event-Based Simulation Support

Simulink is tightly integrated with Stateflow®, the MathWorks' solution for modeling event-

Simulation

After building your block diagram in Simulink, you can debug it using the interactive Simulink debugger. Then, you can run interactive simulations and view the results live. The powerful suite of solvers available in Simulink make simulation results extremely accurate.

Simulink Debugger

The Simulink debugger is an interactive tool for locating and diagnosing errors in a Simulink model. It enables you to quickly pinpoint problems in your model by running simulations step-by-step and displaying intermediate block states and input and output values. The

driven behavior. The seamless interaction between Simulink and Stateflow gives you the ability to model and simulate your system's dynamic and event-driven behavior as a single, integrated system. (For example, Simulink and Stateflow share an integrated Finder.) Designers of automotive, aerospace, telecommunications, and many other types of embedded systems have a complete solution to perform faster, more accurate and extensive simulations of complex, large-scale systems.

You can use Stateflow charts to include supervisory control logic within your Simulink model for activating or deactivating conditionally executed subsystems in Simulink. The Stateflow chart receives input from the Simulink model, determines the actions to be taken, changes states appropriately, and sends logic signals to activate or deactivate the triggered and enabled subsystems in Simulink.

Data Typing

Simulink supports complex numbers for basic blocks and complex/real conversions. In addition, the Data Type Conversion block allows you to convert a signal of one type (such as a float) to a signal of another type (int32, for example).

Many of the blocks in Simulink support several data types. The ability to specify the data types of a model's signal and block parameters is particularly useful in real-time applications such as microcontrollers and DSPs. With this capability, you can specify the optimal data types required to represent signals, block parameters, and mathematical operations exactly as they are represented on these devices. Additionally, by choosing the appropriate data types for your model's signals and parameters, you can dramatically increase the performance and decrease the size of code generated from the model. Supported data types are

double-precision floating point; single-precision floating point; signed and unsigned 8-, 16-, and 32-bit integers; and Boolean.

Audits and Revision Histories

Simulink models are compatible with standard configuration control software such as SCCS and RCS. As a result, audits and revision histories are easily maintained for large projects and for models shared within a multi-platform workgroup.

Analysis

Simulink includes many features for detailed system analysis. Key capabilities include: linearization, equilibrium point determination, animation, parameter optimization, and parametric analysis.

Extracting Linear Models

The dynamics of nonlinear block diagrams can be approximated through linearization, enabling you to apply design techniques that require linear model representations. You can use Simulink's linmod function to obtain linear state-space models from your block diagrams.

Animation

Simulink provides immediate access to MATLAB's powerful 2-D and 3-D graphics and animation capabilities. You can use MATLAB to enhance your visual displays and gain deeper insight into your system's behavior as the simulation progresses.

Integration with MATLAB

Because Simulink is built on top of MATLAB, it provides a unique development environment. This system allows you to run simulations either interactively, using Simulink's graphical interface, or systematically, by running sets of experiments in batch mode from the MATLAB command line. You can then generate test vectors and analyze the results collectively.

Related Products

Simulink is the foundation for a family of design solutions, spanning DSP, communications, control, and power system design.

Companion products include:

- Real-Time Workshop for code generation
- Stateflow for event-driven systems and logic design
- Simulink Performance Tools for simulation acceleration and more
- Block libraries for specialized applications, such as the DSP Blockset, the Fixed-Point Blockset, the Power System Blockset, and the Communications Blockset. ■

For demos, application examples, tutorials, user stories, and pricing:

• Visit www.mathworks.com	
• Contact The MathWorks directly	
US & Canada 508-647-7000	
Benelux +31 (0)182 53 76 44	
France +33 (0)1 41 14 67 14	
Germany +49 (0)89 995901 0	
Spain +34 93 362 13 00	
Switzerland +41 (0)31 954 20 20	
UK +44 (0)1223 423 200	

Visit www.mathworks.com to obtain contact information for authorized MathWorks representatives in countries throughout Asia Pacific, Latin America, the Middle East, Africa, and the rest of Europe.

Communications Blockset 2

for designing and simulating communication systems with Simulink

The Communications Blockset builds upon the Simulink® system-level design environment by providing more than 150 blocks to model the components of a communications system's physical layer.

Example systems that can be designed with the Communications Blockset include cellular handsets and base stations, cordless phones, digital subscriber lines (DSLs), cable and dial-up modems, local area networks (LANs), wireless LANs, digital video broadcasting and satellite systems. The blockset can also be used for system-level design of the semiconductors used in such products. Other applications include the design of read channels for mass storage devices such as tape drives, disk drives and DVDs.

The blockset is used in conjunction with other MathWorks products including the DSP Blockset and the Communication Toolbox. For large models or long simulation runs, the Real-time Workshop® can generate a stand-alone C executable. You can also design the link layer of your communications system in Stateflow,® the MathWorks control logic design tool.

The Simulink Environment

Performing system-level design with Simulink and the Communication Blockset allows the rapid high-level design and testing of complete end-to-end communications systems. You can easily explore ideas and evaluate tradeoffs early in the design process. The resultant validated design can be used as an executable specification or reference model for the hardware or embedded software design stage.

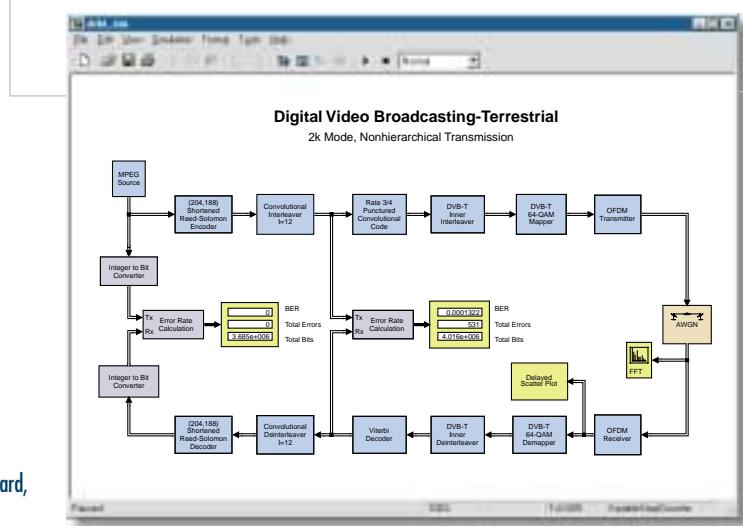
CONTINUED ON BACK PAGE

This model built with the Communications Blockset implements the ETSI EN 300 744 terrestrial digital video broadcasting standard, which utilizes 2048 carrier OFDM.



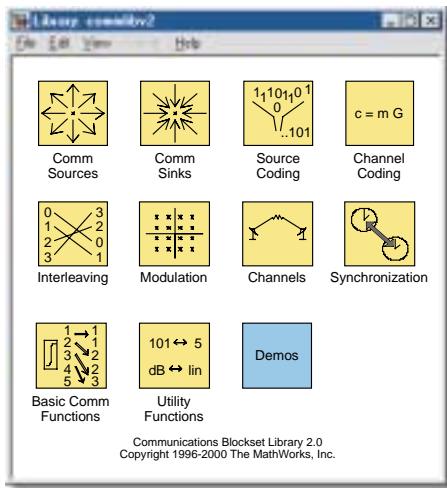
KEY FEATURES

- Convolutional coding including a posteriori probability (APP) and Viterbi decoders
- Block coding with Reed-Solomon, Hamming, BCH, and general cyclic and linear codes
- Block and convolutional interleaving libraries that support general interleaving, as well as several special cases
- Baseband and passband digital modulation libraries including amplitude modulation (PAM, QAM), frequency modulation (FSK), and phase modulation (PSK, DPSK)
- A continuous phase modulation (CPM) library including CPSK, MSK, GMSK and partial response techniques
- Typical channel models including binary symmetric, additive white Gaussian noise (AWGN), Rician and multipath Rayleigh fading
- Sequence operations for the manipulation of data including conversion, repeating, phase shifting, interlacing, and puncturing
- Display devices such as eye-diagram and scatter plot to visualize modulated signals and an error meter to calculate bit or symbol error rates
- Full C source code for all transmitter and receiver blocks allowing you to modify or add other proprietary functionality
- Signal sources for the generation of test signals





Blocks



The Communications Blockset library contains 10 sub-libraries, each providing blocks for different aspects of communications system design.

COMMUNICATION SOURCES



- Bernoulli random binary generator
- Binary vector noise generator
- Gaussian noise generator
- PN sequence generator
- Poisson integer generator
- Random integer generator
- Rayleigh noise generator
- Rician noise generator
- Uniform noise generator
- Voltage-controlled oscillator
- Discrete-time VCO
- Triggered read from file

COMMUNICATION SINKS



- Continuous-time eye and scatter diagrams
- Discrete-time eye and scatter diagrams
- Error rate calculation
- Triggered write to file

SOURCE CODING



- A-Law compressor and expander
- μ -Law compressor and expander
- Differential encode and decode
- DPCM encode and decode
- Sampled quantizer encode
- Enabled quantizer encode
- Quantizer decode

CHANNEL CODING



Block Coding

- BCH encoder and decoder
- Binary cyclic encoder and decoder
- Binary RS encoder and decoder
- Binary linear encoder and decoder
- Hamming encoder and decoder
- Integer RS encoder and decoder

Convolutional Coding

- Convolutional encoder
- APP decoder
- Viterbi decoder

INTERLEAVING

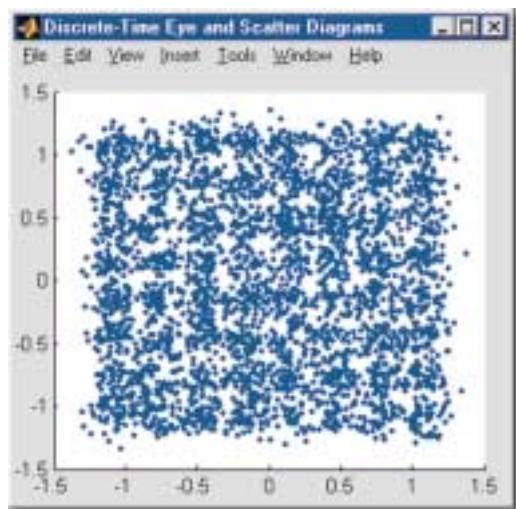


Block Interleaving

- Algebraic interleaver and deinterleaver
- General block interleaver and deinterleaver
- Matrix interleaver and deinterleaver
- Matrix helical scan interleaver and deinterleaver
- Random interleaver and deinterleaver

Convolutional Interleaving

- Convolutional interleaver and deinterleaver
- General multiplexed interleaver and deinterleaver
- Helical interleaver and deinterleaver



Here the output of the scatter plot block shows the constellation of a noisy 64-QAM signal.

MODULATION



Digital Baseband and Passband Amplitude Modulation

- General QAM modulator and demodulator
- M-PAM modulator and demodulator
- Rectangular QAM modulator and demodulator

Phase Modulation

- BPSK modulator and demodulator
- DBPSK modulator and demodulator
- DQPSK modulator and demodulator
- M-DPSK modulator and demodulator
- M-PSK modulator and demodulator
- OQPSK modulator and demodulator
- QPSK modulator and demodulator

Frequency Modulation

- M-FSK modulator and demodulator

Continuous Phase Modulation

- CPFSK modulator and demodulator
- CPM modulator and demodulator
- GMSK modulator and demodulator
- MSK modulator and demodulator

Analog Baseband and Passband

- DSB AM modulator and demodulator
- DSBSC AM modulator and demodulator
- FM modulator and demodulator
- PM modulator and demodulator
- SSB AM modulator and demodulator

Here a tutorial example shows how you can quickly build a communication system comprising a channel, modulation scheme, and coding.

CHANNELS



- AWGN channel
- Binary symmetric channel
- Multipath Rayleigh fading channel
- Rician fading channel

SYNCHRONIZATION



- Phase-locked loop
- Baseband PLL
- Charge pump PLL
- Linearized baseband PLL

BASIC COMM FUNCTIONS



Integrators

- Discrete modulo integrator
- Integrate and dump
- Modulo integrator
- Windowed integrator

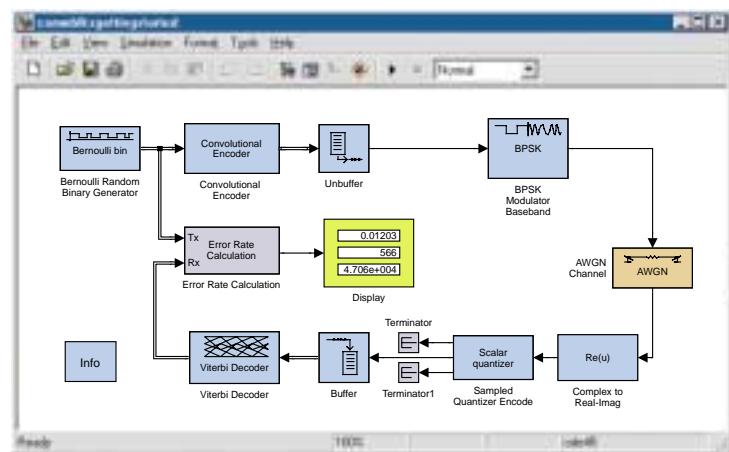
Sequence Operations

- Complex phase difference
- Complex phase shift
- Interlacer and deinterlacer
- Repeat and derepeat
- Puncture and insert zero
- Scrambler and descrambler

UTILITY FUNCTIONS

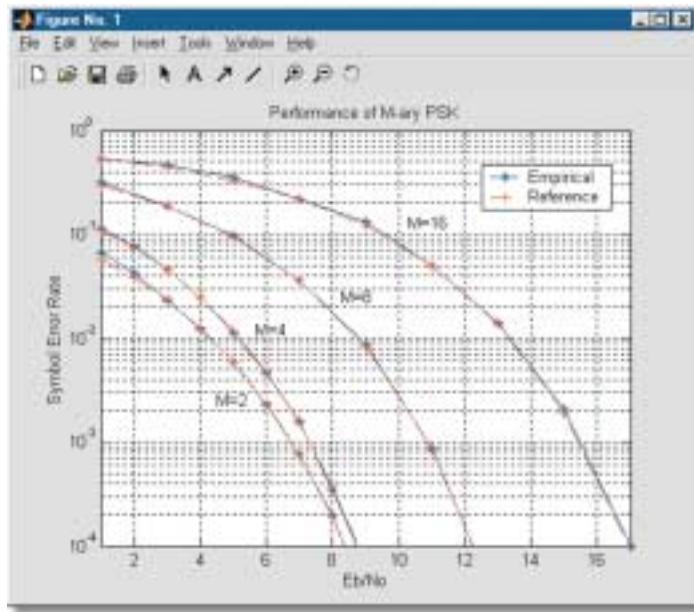


- Bit to integer converter
- Data mapper
- dB



Being based on Simulink, the Communications Blockset handles arbitrarily complex systems by allowing you to build and navigate models hierarchically. You can process all the multi-rate digital signals that are typical in communications systems, such as frames, bits and symbols. And you can make use of Simulink's continuous time features to model analog signals.

Simulink provides the interactive, block diagram simulation environment including model construction, navigation, simulation management and debugging. It also provides primitive analog and discrete, linear and non-linear building blocks, such as arithmetic, logic and relational operators, subsystems, Laplace transforms, z-transforms, look-up tables, polynomials and switches. You also have the ability to add your own custom C code or M code modules using the Simulink S-function block. ■



This BER plot can be calculated by a MATLAB script that runs your simulation many times for different Eb/No values.

USING THE COMMUNICATIONS BLOCKSET WITH OTHER MATHWORKS PRODUCTS

To run the Communications Blockset, the Communications Toolbox, the Signal Processing Toolbox, Simulink and the DSP Blockset must also be installed.

The DSP Blockset This provides all the key DSP blocks common in any digital communications system. These blocks include filters, adaptive filters, interpolation, signal operations, transforms, vector math, matrix math, linear algebra, and frequency scopes. The Communications Blockset also makes extensive internal use of the DSP Blockset.

The Communication Toolbox This provides a number of support functions for error correction coding including polynomial creation and Galois field computations. The Communications Blockset also makes extensive internal use of the Communications Toolbox.

MATLAB® With MATLAB you can create scripts to automate the running of your simulation multiple times to calculate bit-error plots. You can also use it for post processing of simulation data as well as numerous ancillary parameter manipulation and generation tasks.

Real-Time Workshop For large models or long simulation runs, Real-time Workshop can generate a standalone C executable for running multiple simulations or for co-simulation with low-level EDA tools.

Stateflow® You can also integrate your physical layer design in Simulink and the Communications Blockset with your link-layer design in Stateflow, The MathWorks control logic design product.

For demos, application examples, tutorials, user stories, and pricing:

- Visit www.mathworks.com
- Contact The MathWorks directly
- US & Canada 508-647-7000
- Benelux +31 (0)182 53 76 44
- France +33 (0)1 41 14 67 14
- Germany +49 (0)89 995901 0
- Spain +34 93 362 13 00
- Switzerland +41 (0)31 954 20 20
- UK +44 (0)1223 423 200

Visit www.mathworks.com to obtain contact information for authorized MathWorks representatives in countries throughout Asia Pacific, Latin America, the Middle East, Africa, and the rest of Europe.

Communications Toolbox 2

for designing and simulating communication system algorithms with MATLAB®

The Communications Toolbox is a library of MATLAB® functions that facilitate the design of communication system algorithms and components.

This toolbox builds upon the powerful capabilities of MATLAB and the Signal Processing Toolbox by providing functions to model the components of a communications system's physical layer. These functions can be used to analyze and develop components in products such as cellular handsets and base stations, cordless phones, digital subscriber lines (DSLs), cable and dial-up modems, local area networks (LANs), wireless LANs, and satellite systems. The toolbox also offers a foundation for research and education in communication systems engineering.

For complete end-to-end communication system-level design, The Mathworks also offers the Communications Blockset. Together with Simulink, this blockset provides a block diagram simulation environment that is ideal for modeling all the analog and multirate digital signals typical in communication systems, such as frames, bits, and symbols.

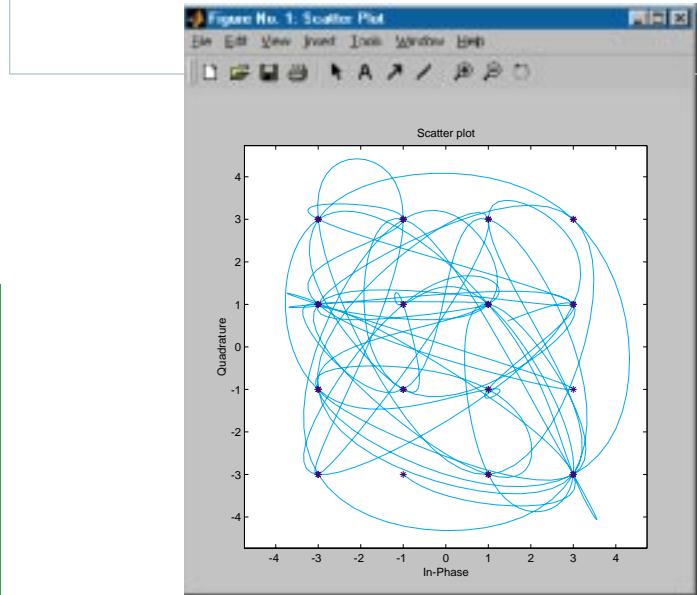
Creating a 32-tone baseband FSK modulated signal is as easy as calling the `dmodce` function with appropriate parameters.

```
% Number of symbols and tone spacing  
M = 32, tone = .25;  
  
% Symbol and sample rates  
Fd = 1; Fs = 6;  
  
% Generate 200 random symbols  
x = randint(200,1,M);  
  
% Perform FSK modulation  
y = dmodce(x,Fd,Fs,'fsk',M,tone);
```



KEY FEATURES

- Signal generator functions for creating test signals
- Display functions, such as eye diagram and scatter plots, to visualize modulated signals
- Source coding, including quantizers, differential pulse code modulation (DPCM), μ -law, and A-Law companders
- Block coding with Reed-Solomon, Hamming, BCH, general cyclic, and linear codes
- Convolutional coding, including Viterbi decoders
- Baseband and passband digital modulation functions, including amplitude shift keying (ASK), phase shift keying (PSK), and frequency shift keying (FSK)
- Baseband and passband analog modulation libraries, including analog modulation (AM), frequency modulation (FM), and phase modulation (PM)
- An additive white Gaussian noise (AWGN) channel
- Galois field calculations for GF(q^m) polynomial manipulation and representation



Signal display functions like this scatter plot help you to quickly visualize a modulated signal with a single function call.



Functions

Signal Sources

<code>randerr</code>	Generate bit error patterns
<code>randint</code>	Generate matrix of uniformly distributed random integers
<code>randsrc</code>	Generate random matrix using prescribed alphabet
<code>wgn</code>	Generate white Gaussian noise

Signal Analysis Functions

<code>biterr</code> , <code>symperr</code>	Compute number of bit or symbol errors and bit or symbol error rate
<code>eyediagram</code>	Generate an eye diagram
<code>scatterplot</code>	Generate a scatter diagram

Source Coding

<code>compand</code>	Source code μ -law or A-law compressor or expander
<code>dpcmenco</code> , <code>dpcmdeco</code>	Encode and decode using differential pulse code modulation
<code>dpcmopt</code>	Optimize differential pulse code modulation parameters
<code>lloyds</code>	Optimize quantization parameters using the Lloyd algorithm
<code>quantiz</code>	Produce a quantization index and a quantized output value

Error-Control Coding

<code>bchpoly</code> , <code>cyclpoly</code> , <code>rspoly</code>	Produce parameters or generator polynomial for BCH, cyclic, or Reed-Solomon code
<code>convenc</code>	Convolutionally encode binary data
<code>cyclgen</code> , <code>hammgen</code>	Produce parity check and generator matrix for cyclic and Hamming code
<code>encode</code> , <code>decode</code>	Block encoder and decoder
<code>gen2par</code>	Convert between parity-check and generator matrices
<code>gfweight</code>	Calculate the minimum distance of a linear block code
<code>rsencof</code> , <code>rsdecof</code>	Encode or decode an ASCII file using Reed-Solomon code
<code>syndtable</code>	Produce syndrome decoding table
<code>vitdec</code>	Decode convolutionally encoded binary data using the Viterbi algorithm

Lower-Level Functions for Error-Control Coding

<code>bchenco</code> , <code>bchdeco</code>	BCH encoder and decoder
<code>rsenco</code> , <code>rsdeco</code>	Reed-Solomon encoder and decoder
<code>rsenocode</code> , <code>rsdecode</code>	Reed-Solomon encoding and decoding using the exponential format

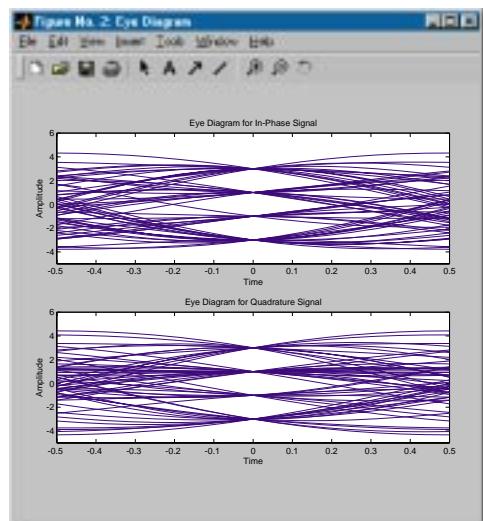
Modulation

Analog Passband and Baseband Modulation and Demodulation

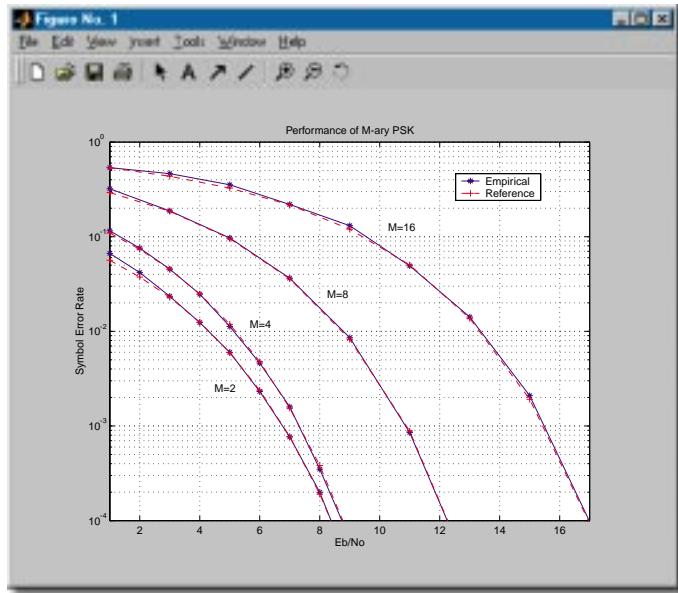
<code>amod</code> , <code>ademod</code>	Passband modulate and demodulate an analog signal with double sideband AM, single sideband AM, QAM, FM, or PM
<code>amodce</code> , <code>ademodce</code>	Baseband modulate and demodulate an analog signal using complex envelope with double sideband AM, single sideband AM, QAM, FM, or PM

Digital Passband and Baseband Modulation and Demodulation

<code>dmod</code> , <code>ddemod</code>	Passband modulate and demodulate a digital signal with ASK, PSK, QASK, FSK, or MSK
<code>dmodce</code> , <code>ddemodce</code>	Baseband modulate and demodulate a digital signal using complex envelope with ASK, PSK, QASK, FSK, or MSK



A plot from the eye diagram function shows the decision point of a modulated signal.



Here, the symbol error function is used to calculate the symbol error rate of M-ary PSK for a range of Eb/No value.

Digital Mapping/Demapping

modmap	Map a digital signal to an analog signal
demodmap	Demap a digital message from a demodulated signal
apkconst	Plot a combined circular APK-PSK signal constellation
qaskdeco	Demap a message from a QASK square signal constellation
qaskenco	Map a message to a QASK square signal constellation

Special Filters

hilbiir	Design a Hilbert transform IIR filter
hank2sys	Convert a Hankel matrix to a linear system model
rcosfir, rcosiir	Design a raised cosine FIR or IIR filter (lower-level function)
rcosflt	Filter the input signal using a raised cosine filter
rcosine	Design a raised cosine filter

Channel Functions

awgn	Add white Gaussian noise to a signal
------	--------------------------------------

Galois Field Computation Functions

flior	Perform integer exclusive OR (XOR) computation
gfadd, gfdeconv, gfconv,gfsub	Add, divide, multiply, and subtract polynomials over a Galois field
gfmul, gfdinv	Multiply and divide elements of a Galois field
gfcosets	Produce cyclotomic cosets for a Galois field
gfplus	Add elements of a Galois field of characteristic two
fffilter	Filter data using polynomials over a prime Galois field
gftrunc	Minimize the length of a polynomial representation
gflineq	Solve the linear equation Ax=b over a prime Galois field
gfminpol	Find the minimal polynomial of an element of a Galois field
gfroots	Find the roots of a polynomial over a prime Galois field
gfprimck	Check whether a polynomial over a Galois field is primitive
gfprimdf, gfprimfd	Provide default primitive polynomials and find primitive polynomials for a Galois field
gfrank	Compute the rank of a matrix over a Galois field
gfrepcov	Convert one GF(2) polynomial representation to another
gftuple	Simplify or convert the format of elements of a Galois field
gfpretty	Display a polynomial in traditional format

Utilities

bi2de, de2bi	Convert between binary vectors and decimal numbers
erf	Error function (in MATLAB)
erfc	Complementary error function (in MATLAB)
istrellis	Check if input is a valid trellis structure
marcumq	Generalized Marcum Q function
oct2dec	Convert octal numbers to decimal numbers
poly2trellis	Convert convolutional code polynomials to trellis description
vec2mat	Convert a vector into a matrix

```

% Define symbol rate and sampling rate
Fd = 1; Fs = 4;

% Define alphabet (quaternary).
M = 4;

% Generate 2048 random integers in the range [0,M-1]
msg_mod = randsrc(2048,1,[0:M-1]);

% Digitally modulate and upsample the signal
msg_tx = dmodce(msg_mod, Fd, Fs, 'psk', M);

% Add Gaussian noise to the signal at 10dB SNR. The noise signal is
% calibrated using the 'measured' option. The noise power is scaled for
% oversampling.
msg_rx = awgn(msg_tx, 10, 'measured', [], 'dB');

% Demodulate, detect, and downsample the signal
msg_demod = ddemodce(msg_rx, Fd, Fs, 'psk', M);

% Calculate bit error count, BER, symbol error count and SER
[errBit ratBit] = biterr(msg_mod, msg_demod, log2(M));
[errSym ratSym] = symerr(msg_mod, msg_demod);

```

In this code sample, Communications Toolbox functions generate random integers, modulate with QPSK, add Gaussian noise and demodulate, then calculate bit error and symbol error rates.

PRODUCT REQUIREMENTS

The Communications Toolbox is available on all MathWorks supported platforms. It requires the Signal Processing Toolbox.

For demos, application examples, tutorials, user stories, and pricing:

- Visit www.mathworks.com

- Contact The MathWorks directly

US & Canada 508-647-7000

Benelux	+31 (0)182 53 76 44
France	+33 (0)1 41 14 67 14
Germany	+49 (0)89 995901 0
Spain	+34 93 362 13 00
Switzerland	+41 (0)31 954 20 20
UK	+44 (0)1223 423 200

Visit www.mathworks.com to obtain contact information for authorized MathWorks representatives in countries throughout Asia Pacific, Latin America, the Middle East, Africa, and the rest of Europe.

DSP Blockset 4

for designing, simulating, and prototyping digital signal processing systems

The DSP Blockset is an intuitive tool, composed of Simulink® block libraries, for the rapid design, graphical simulation, and prototyping of DSP systems. It is a component of the DSP and Communications Design Suites, which provide complete system-level design capability.

DSP Blockset models are constructed within the Simulink environment. You select blocks from the available libraries and interconnect them in various configurations using the mouse. Signal source blocks are available for testing your models. You can visualize your simulation interactively or pass simulation results to MATLAB® for post-processing. In addition to the built-in blocks, you can incorporate C code, MATLAB functions, and M-files. You can use Real-Time Workshop® to generate ANSI standard C code directly from your model.

The DSP Blockset provides the algorithmic foundation for many applications in speech and audio processing, telephone, wireless and broadband communications, computer peripherals, radar/sonar, and medical electronics. It contains blocks for filter design, spectral estimation, and transforms, among others.

Working with the DSP Blockset

The DSP Blockset complements the powerful algorithm development and signal analysis tools in MATLAB by providing an interactive block diagram environment for system-level simulation and real-time algorithm design.

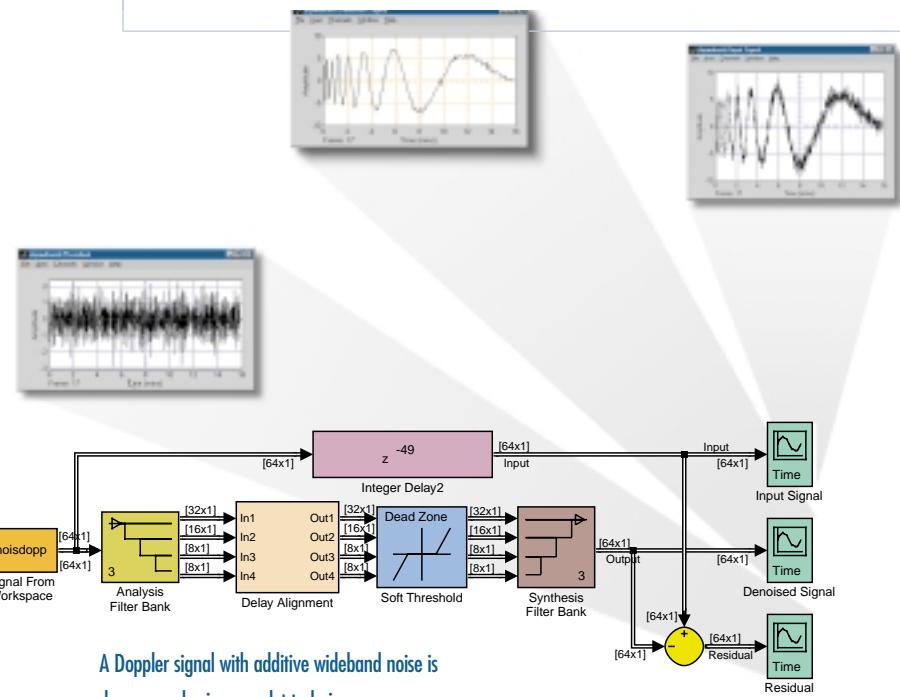
Advanced Signal Processing

The DSP Blockset offers a wide range of built-in DSP techniques, including transforms, buffering, filter design, and linear algebra. You can create sophisticated DSP simulations without low-level programming and easily add your own custom algorithms.



KEY FEATURES

- Signal processing blocks—including FFT, DFT, and their inverses; window functions; decimation/interpolation; and linear prediction
- Spectral estimation blocks—including short-time FFT, Yule-Walker AR, Burg, and modified covariance methods
- Filter design blocks—including traditional FIR and IIR filters, adaptive filters, lattice filters, and multirate filters
- Filter structure blocks—including Direct Form II Transpose, biquadratic, and lattice filters
- Filter Realization Wizard for automatically generating block diagram models of digital filters from your specifications
- Math functions—including normalization and cumulative sum, matrices and linear algebra (such as QR factorization, and singular value decomposition), and polynomial functions (such as least squares polynomial fit)
- Numerous blocks for statistics, quantizers, signal operations, signal management, and a variety of other operations



Sample Blocks



DSP Sources

Chirp, Identity Matrix, Constant Diagonal Matrix, Multiphase Clock, Constant Ramp, N-Sample Enable, Counter, Random Source, DSP Constant, Signal From Workspace, Discrete Impulse, Sine Wave, From Wave Device, Triggered Signal From Workspace, From Wave File, Window Function



DSP Sinks

Display, Time Scope, Matrix Viewer, To Wave Device, Signal To Workspace, To Wave File, Spectrum Scope, Triggered To Workspace, Vector Scope

Filtering



Filter Designs

Analog Filter Design, Least Squares FIR Filter Design, Digital FIR Filter Design, Remez FIR Filter Design, Digital FIR Raised Cosine Filter Design, Yule-Walker IIR Filter Design, Digital IIR Filter Design



Filter Structures

Analog Filter Design, Overlap-Add FFT Filter, Biquadratic Filter, Overlap-Save FFT Filter, Direct-Form II Transpose Filter, Time-Varying Direct-Form II Transpose Filter, Filter Realization Wizard, Time-Varying Lattice Filter



Multirate Filters

Dyadic Analysis Filter Bank, FIR Rate Conversion, Dyadic Synthesis Filter Bank, Wavelet Analysis, FIR Decimation, Wavelet Synthesis, FIR Interpolation



Adaptive Filters

Kalman Adaptive Filter, RLS Adaptive Filter, LMS Adaptive Filter



Estimation

Linear Prediction

Autocorrelation LPC



Parametric Estimation

Burg AR Estimator, Modified Covariance AR Estimator, Covariance AR Estimator, Yule-Walker AR Estimator



Power Spectrum Estimation

Burg Method, Modified Covariance Method, Covariance Method, Short-Time FFT, Magnitude FFT, Yule-Walker Method

Math Functions

Matrices and Linear Algebra

Matrix Operations

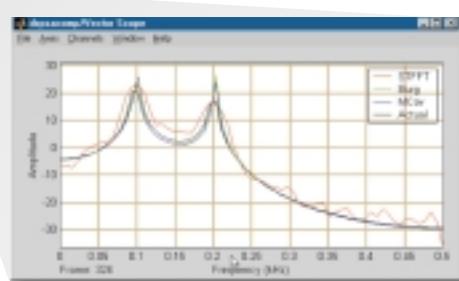
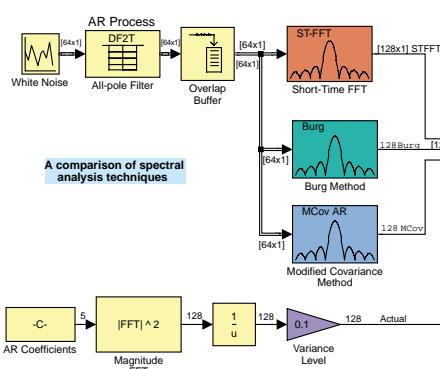


Constant Diagonal Matrix, Matrix Scaling, Create Diagonal Matrix, Matrix Square, Extract Diagonal, Matrix Sum, Extract Triangular Matrix, Permute Matrix, Identity Matrix, Reciprocal Condition, Matrix Concatenation (Simulink block), Submatrix, Matrix 1-Norm, Toeplitz, Matrix Multiply, Transpose, Matrix Product



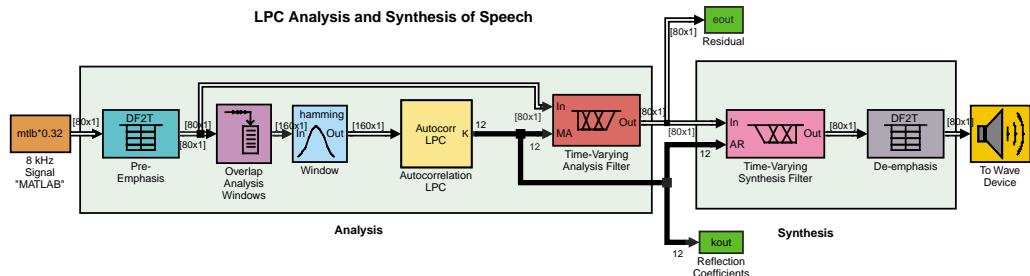
Linear System Solvers

Levinson-Durbin, Cholesky Solver, LU Solver, Backward Substitution, Forward Substitution, QR Solver, LDL Solver, SVD Solver



This DSP Blockset model compares several of the spectral estimation methods available in the DSP Blockset.

DSP Blockset features such as frame-based processing, real-time audio, and a linear algebra library enable the design of real-time speech and audio algorithms.



Matrix Factorizations

Cholesky Factorization, QR Factorization, LDL Factorization, Singular Value Decomposition, LU Factorization



Matrix Inverses

Cholesky Inverse, LU Inverse, LDL Inverse, Pseudoinverse



Math Operations

Complex Exponential, dB Gain, Cumulative Sum, Normalization, dB Conversion, Difference

Polynomial Functions



Least Squares Polynomial Fit, Polynomial Stability Test, Polynomial Evaluation

Quantizers



Quantizer, Uniform Encoder, Uniform Decoder

Signal Management



Buffers

Buffer, Stack, Delay Line, Triggered Delay Line, Queue, Unbuffer



Indexing

Flip, Submatrix, Multiport Selector, Variable Selector, Selector



Signal Attributes

Check Signal Attributes, Convert 2-D to 1-D, Contiguous Copy, Frame Status Conversion, Convert 1-D to 2-D, Inherit Complexity



Switches and Counters

Counter, Multiphase Clock, Edge Detector, N-Sample Enable, Event-Count Comparator, N-Sample Switch



Signal Operations

Convolution, Unwrap, Downsample, Upsample, Integer Delay, Variable Fractional Delay, Pad, Variable Integer Delay, Repeat, Window Function, Sample and Hold, Zero Pad



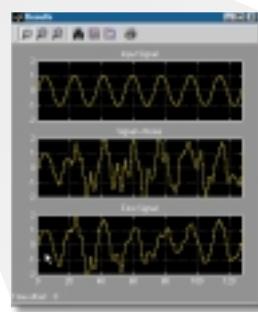
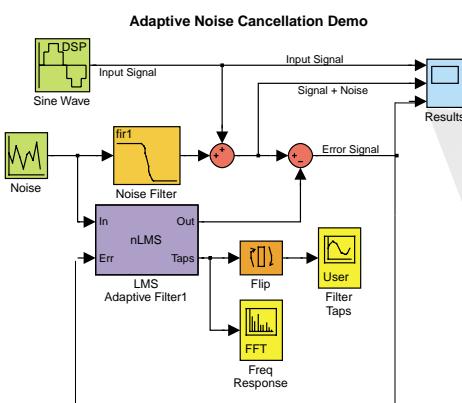
Statistics

Autocorrelation, Median, Correlation, Minimum, Detrend, RMS, Histogram, Sort, Maximum, Standard Deviation, Mean, Variance

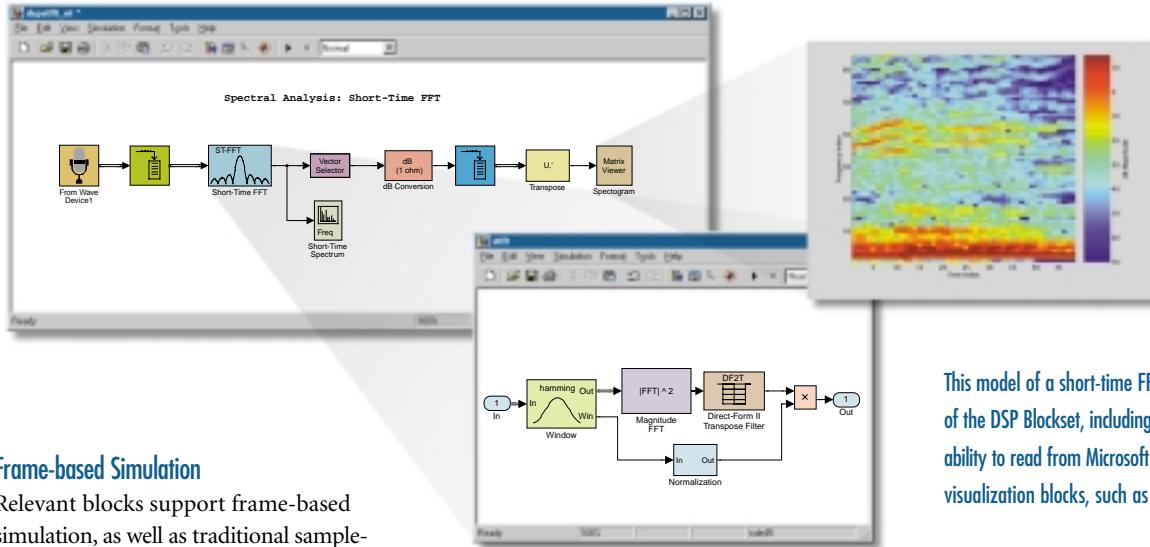


Transforms

Analytic Signal, IDCT, Complex Cepstrum, IFFT, DCT, Real Cepstrum, FFT



The block diagram shown here simulates an adaptive equalizer using the DSP Blockset's LMS adaptive filter algorithm. The scope shows the input signal (top), the signal plus noise (middle), and the error signal (bottom).



Frame-based Simulation

Relevant blocks support frame-based simulation, as well as traditional sample-based simulation, for greater speed and more accurate representation of real-time systems. A frame is a collection of consecutive samples that have been stored in a single vector. By propagating these multisample frames instead of the individual signal samples, DSP systems can take advantage of the speed of DSP algorithm execution while reducing overhead and handling demands.

Multichannel Processing

DSP Blockset blocks can generate and process matrix signals with standard MATLAB matrix notation. Matrix dimensions are automatically detected by all blocks. The combination of matrix support and frame-based processing means that you can use the DSP Blockset for processing real-time multichannel data. For example, a frame-based matrix with three columns contains three channels of data, one frame per channel.

Simple Handling of Real and Complex Values

Each block implicitly handles real and complex input signals. Separate blocks for real and complex data processing values are not required.

Multirate Processing

DSP Blockset can support multirate processing without requiring you to explicitly set the sample time of multirate blocks. The correct sample time and frame size are automatically propagated from block to block.

Portable C Code Generation

DSP Blockset interfaces seamlessly with Real-Time Workshop[®], allowing you to automatically generate real-time ANSI C code from your Simulink DSP simulations that is suitable for use in embedded real-time applications. This code-generation facility lets you streamline prototyping and implementation on programmable floating-point DSP hardware.

DSP Blockset and Simulink

Simulink is an interactive environment for modeling, analyzing, and simulating a wide variety of dynamic systems, including discrete, analog, and hybrid systems. The DSP Blockset adds fundamental simulation capability to the Simulink environment (efficient processing of frame-based data, and DSP algorithms such as filters, transforms, and windows). DSP Blockset and Simulink together let you develop efficient DSP applications and evaluate them within your end-to-end system simulation.

DSP Blockset and the Signal Processing and Filter Design Toolboxes

You can use the DSP Blockset with the Signal Processing Toolbox to design digital and analog filters. Adding the Filter Design Toolbox enables you to design advanced digital filters. Through a Simulink dialog box, you can specify filter parameters that the Signal

This model of a short-time FFT incorporates many features of the DSP Blockset, including frame-based processing, the ability to read from Microsoft Windows audio devices, and visualization blocks, such as the matrix viewer.

Processing Toolbox and the Filter Design Toolbox can use to generate filter coefficients. You can use the resulting filter as part of your Simulink DSP simulation. Similarly, the Filter Realization Wizard of the DSP Blockset can accept filter parameters created by the Signal Processing and Filter Design toolboxes to create a custom filter in block-diagram format.

Product Requirements

The DSP Blockset runs on all MathWorks supported platforms. It requires the Signal Processing Toolbox 5, Simulink 4, and MATLAB 6. ■

For demos, application examples, tutorials, user stories, and pricing:

- Visit www.mathworks.com
 - Contact The MathWorks directly
- | | |
|-------------|----------------------|
| US & Canada | 508-647-7000 |
| Benelux | +31 (0)182 53 76 44 |
| France | +33 (0)1 41 14 67 14 |
| Germany | +49 (0)89 995901 0 |
| Spain | +34 93 362 13 00 |
| Switzerland | +41 (0)31 954 20 20 |
| UK | +44 (0)1223 423 200 |

Visit www.mathworks.com to obtain contact information for authorized MathWorks representatives in countries throughout Asia Pacific, Latin America, the Middle East, Africa, and the rest of Europe.

Signal Processing Toolbox 5

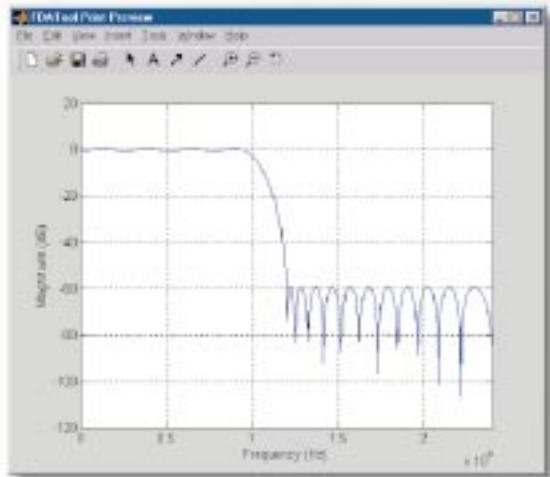
for algorithm development, signal and linear system analysis, and time-series modeling

The Signal Processing Toolbox is a collection of MATLAB® functions that provides a rich, customizable framework for analog and digital signal processing (DSP). Graphical user interfaces (GUIs) support interactive designs and analyses, while command-line functions support advanced algorithm development.

The Signal Processing Toolbox is the ideal environment for signal analysis and DSP algorithm development. It uses industry-tested signal processing algorithms that have been carefully chosen and implemented for maximum efficiency and numeric reliability.

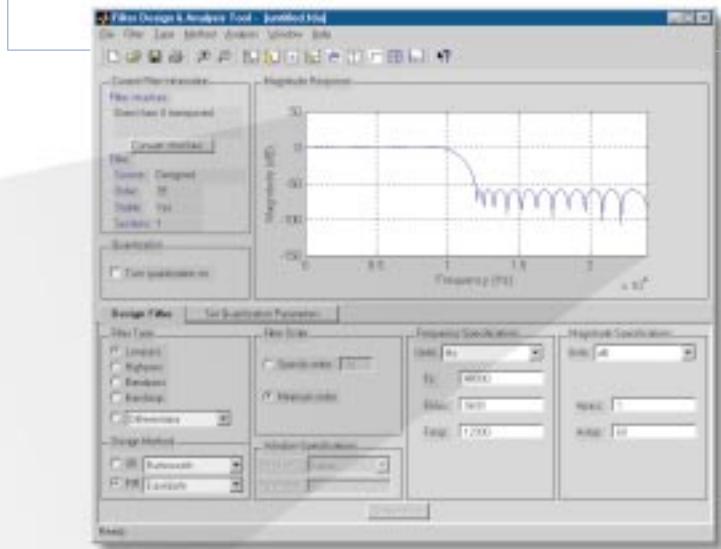
Signal Processing Toolbox functions are implemented as M-files routines written in the MATLAB language, which give you access to the source code and algorithms. The open-system philosophy of MATLAB and the toolboxes enables you to make changes to existing functions or add your own.

You can use the toolbox in speech and audio processing, communications, digital control, radar, geophysics, test instrumentation, real-time control, finance, medicine, and other applications.



KEY FEATURES

- A comprehensive set of signal and linear system models
- Tools for analog filter design
- Tools for Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) digital filter design, analysis, and implementation
- The most widely used transforms, such as fast Fourier transform (FFT) and discrete cosine transform (DCT)
- Methods for spectrum estimation and statistical signal processing
- Functions for parametric time-series modeling
- Routines for waveform generation, including a Gaussian pulse generator, a periodic sinc generator, and a pulse train generator
- Data windowing algorithms



FDATool (above) is a built-in GUI that lets you design many types of FIR and IIR filters. You select the filter types from the available methods in the GUI. This diagram shows the GUI with Filter Design Toolbox installed. The figure at left shows an annotatable print preview of the filter's magnitude response.



Sample Functions

Filter Analysis

abs	Magnitude
angle	Phase angle
freqs	Laplace transform frequency response
freqspace	Frequency spacing for frequency response.
freqz	Z-transform frequency response
freqzplot	Plot frequency response data
grpdelay	Group delay
impz	Discrete impulse response
unwrap	Unwrap phase
zplane	Discrete pole-zero plot

Filter Implementation

conv	Convolution
conv2	2-D convolution
deconv	Deconvolution
fftfilt	Overlap-add filter implementation
filter	Filter implementation
filter2	Two-dimensional digital filtering
filtfilt	Zero-phase version of filter
filtic	Determine filter initial conditions
latcfilt	Lattice filter implementation
medfilt1	1-Dimensional median filtering
sgolayfilt	Savitzky-Golay filter implementation
sosfilt	Second-order sections (biquad) filter implementation
upfirdn	Up sample, FIR filter, down sample

FIR Filter Design

convmtx	Convolution matrix
cremez	Complex and nonlinear phase equiripple FIR filter design

fir1

Window based FIR filter design - low, high, band, stop, multi

fir2

FIR arbitrary shape filter design using the frequency sampling method

fircls

Constrained Least Squares filter design – arbitrary response

fircls1

Constrained Least Squares FIR filter design – low and highpass

firls

Optimal least-squares FIR filter design

firrcos

Raised cosine FIR filter design

intfilt

Interpolation FIR filter design

kaiserord

Kaiser window design based filter order estimation

remez

Optimal Chebyshev-norm FIR filter design

remezord

Remez design based filter order estimation

sgolay

Savitzky-Golay FIR smoothing filter design

IIR Digital Filter Design

butter

Butterworth filter design

cheby1

Chebyshev type I filter design

cheby2

Chebyshev type II filter design

ellip

Elliptic filter design

maxflat

Generalized Butterworth lowpass filter design

yulewalk

Yule-Walker filter design

IIR Filter Order Estimation

buttord

Butterworth filter order estimation

cheb1ord

Chebyshev type I filter order estimation

cheb2ord

Chebyshev type II filter order estimation

ellipord

Elliptic filter order estimation

Analog Lowpass Filter Prototypes

besselap

Bessel filter prototype

buttap

Butterworth filter prototype

cheb1ap

Chebyshev type I filter prototype (passband ripple)

cheb2ap

Chebyshev type II filter prototype (stopband ripple)

ellipap

Elliptic filter prototype

Analog Filter Design

besself

Bessel analog filter design

butter

Butterworth filter design

cheby1

Chebyshev type I filter design

cheby2

Chebyshev type II filter design

ellip

Elliptic filter design

Analog Filter Transformation

lp2bp

Lowpass to bandpass analog filter transformation

lp2bs

Lowpass to bandstop analog filter transformation

lp2hp

Lowpass to highpass analog filter transformation

lp2lp

Lowpass to lowpass analog filter transformation

Filter Discretization

bilinear

Bilinear transformation with optional prewarping

impinvar

Impulse invariance analog to digital conversion

Linear System Transformations

latc2tf

Lattice or lattice ladder to transfer function conversion

polystab

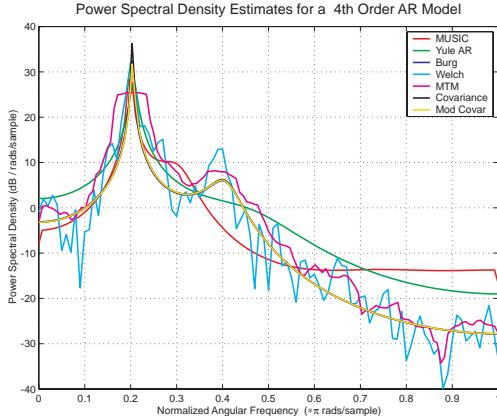
Polynomial stabilization

polyscale

Scale roots of polynomial

residuez	Z-transform partial fraction expansion	fft	Fast Fourier transform	pmusic	Power spectral density estimate via the MUSIC method
sos2ss	Second-order sections to state-space conversion	fft2	2-D fast Fourier transform	pwelch	Power spectral density estimate via Welch's method
sos2tf	Second-order sections to transfer function conversion	fftshift	Swap vector halves	pyulear	Power spectral density estimate via the Yule-Walker AR Method
sos2zp	Second-order sections to zero-pole conversion	hilbert	Discrete-time analytic signal via Hilbert transform	rooteig	Sinusoid frequency and power estimation via the eigenvector algorithm
ss2sos	State-space to second-order sections conversion	idct	Inverse discrete cosine transform	rootmusic	Sinusoid frequency and power estimation via the MUSIC algorithm
ss2tf	State-space to transfer function conversion	ifft	Inverse fast Fourier transform	tfe	Transfer function estimate
ss2zp	State-space to zero-pole conversion	ifft2	Inverse 2-D fast Fourier transform	xcorr	Cross-correlation function
tf2latc	Transfer function to lattice or lattice ladder conversion	cceps	Complex cepstrum	xcorr2	2-D cross-correlation
tf2sos	Transfer function to second-order sections conversion	icceps	Inverse complex cepstrum	xcov	Covariance function
tf2ss	Transfer function to state-space conversion	rceps	Real cepstrum and minimum phase reconstruction		
tf2zp	Transfer function to zero-pole conversion				
zp2sos	Zero-pole to second-order sections conversion				
zp2ss	Zero-pole to state-space conversion				
zp2tf	Zero-pole to transfer function conversion				
<hr/>					
Windows					
bartlett	Bartlett window	pcov	Power spectral density estimate via the covariance method	arburg	AR parametric modeling via Burg's method
blackman	Blackman window			arcov	AR parametric modeling via covariance method
boxcar	Rectangular window	peig	Correlation coefficients	armcov	AR parametric modeling via modified covariance method
chebwin	Chebyshev window			aryule	AR parametric modeling via the Yule-Walker method
hamming	Hamming window			invfreqs	Analog filter fit to frequency response
hann	Hanning window	periodogram	Power spectral density estimate via the eigenvalue method	invfreqz	Discrete filter fit to frequency response
kaiser	Kaiser window		Power spectral density estimate via the periodogram method	prony	Prony's discrete filter fit to time response
triang	Triangular window	pmcov	Power spectral density estimate via the modified covariance method	stmcbs	Steiglitz-McBride iteration for ARMA modeling
<hr/>					
Transforms					
czt	Chirp-z transform	pmtm	Power spectral density estimate via the Thomson multitaper method		
dct	Discrete cosine transform				
dftmtx	Discrete Fourier transform matrix				

Linear Prediction		Multirate Signal Processing		Graphical User Interfaces	
ac2rc	Autocorrelation sequence to reflection coefficients conversion	decimate	Resample data at a lower sample rate	dpss	Discrete prolate spheroidal sequences (Slepian sequences)
ac2poly	Autocorrelation sequence to prediction polynomial conversion	interp	Resample data at a higher sample rate	eqtflength	Equalize the length of a discrete-time transfer function
is2rc	Inverse sine parameters to reflection coefficients conversion	interp1	General 1-D interpolation. (MATLAB Toolbox)	modulate	Modulation for communications simulation
lar2rc	Log area ratios to reflection coefficients conversion	resample	Resample sequence with new sampling rate	seqperiod	Find minimum-length repeating sequence in a vector
levinson	Levinson-Durbin recursion	spline	Cubic spline interpolation	sos2cell	Convert second-order-section matrix to cell array
lpc	Linear predictive coefficients using autocorrelation method	upfirdn	Up sample, FIR filter, down sample	specgram	Spectrogram, for speech signals
lsf2poly	Line spectral frequencies to prediction polynomial conversion	chirp	Swept-frequency cosine generator	stem	Plot discrete data sequence
poly2ac	Prediction polynomial to autocorrelation sequence conversion	diric	Dirichlet (periodic sinc) function	stripes	Strip plot
poly2lsf	Prediction polynomial to line spectral frequencies conversion	gauspuls	Gaussian RF pulse generator	udecode	Uniform decoding of the input
poly2rc	Prediction polynomial to reflection coefficients conversion	gmonopuls	Gaussian monopulse generator	uencode	Uniform quantization and encoding of the input into N-bits
rc2ac	Reflection coefficients to autocorrelation sequence conversion	pulstran	Pulse train generator		
rc2is	Reflection coefficients to inverse sine parameters conversion	rectpuls	Sampled aperiodic rectangle generator		
rc2lar	Reflection coefficients to log area ratios conversion	sawtooth	Sawtooth function		
rc2poly	Reflection coefficients to prediction polynomial conversion	sinc	Sinc or $\sin(\pi x)/(\pi x)$ function	fdatool	Filter Design and Analysis Tool
rlevinson	Reverse Levinson-Durbin recursion	square	Square wave function	sptool	Signal Processing Tool
schurrc	Schur algorithm	tripuls	Sampled aperiodic triangle generator		
		vco	Voltage controlled oscillator		
Specialized Operations					
		buffer	Buffer a signal vector into a matrix of data frames		
		cell2sos	Convert cell array to second-order-section matrix		
		cplxpair	Order vector into complex conjugate pairs		
		demod	Demodulation for communications simulation		



Spectral analysis of a signal using a range of parametric and nonparametric techniques.

Signal and Linear System Models

The Signal Processing Toolbox provides a broad range of models for representing signals and linear time-invariant systems, allowing you to choose the method that best suits your application, including representations for transfer functions state space, and zero-pole-gain. The toolbox also includes functions for transforming models from one representation to another.

Filter Design

The Signal Processing Toolbox features a full suite of design methods for finite impulse response (FIR) and infinite impulse response (IIR) digital filters. These methods support the rapid design and evaluation of lowpass, highpass, bandpass, bandstop, and multi-band filters such as Butterworth, Chebyshev, elliptic, Yule-Walker, window-based, least-squares, and Parks-McClellan. The filter structures available include the direct forms I and II, lattice, lattice-ladder, and second-order sections. You can comment among the various realizations with tools provided.

Spectral Analysis

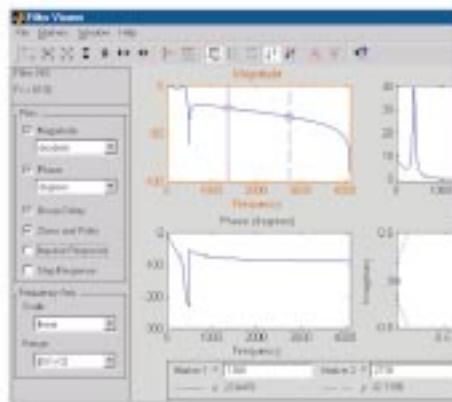
The Signal Processing Toolbox provides unsurpassed facilities for frequency-domain analysis and spectral estimation. Several of these methods are based on a highly optimized FFT. The toolbox includes functions

for computing the discrete Fourier, discrete cosine, Hilbert, and other transforms useful in analysis, coding, and filtering. The spectral analysis methods available include Welch's, Burg's, modified covariance, Yule-Walker, the multitaper method, and the MUSIC method.

Visualization

The GUIs in the Signal Processing Toolbox let you interactively view and measure signals, design and apply filters, and perform spectral analysis while exploring the effects of different analysis parameters and methods.

SPTool's Filter Designer includes a Pole/Zero editor that lets you design a filter through the graphical placement of poles and zeroes. The Filter Viewer lets you view all characteristics of the filter.

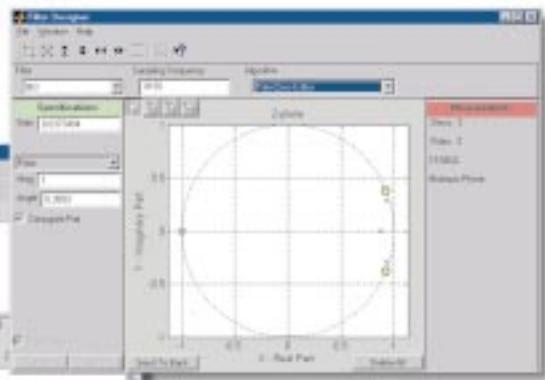


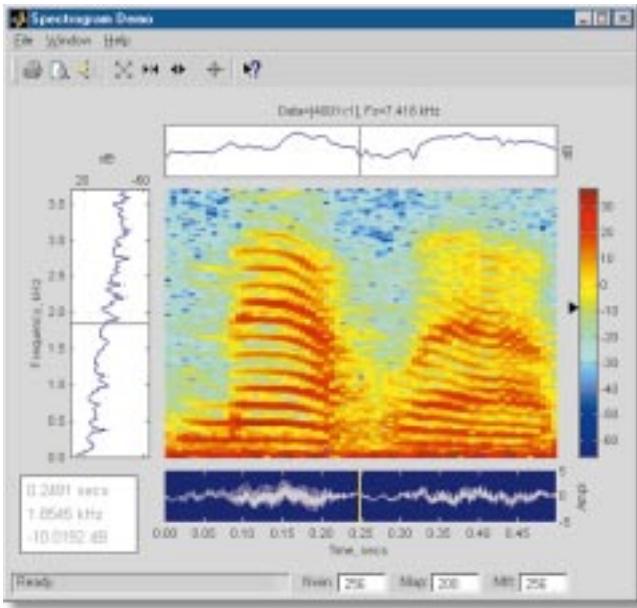
They are particularly useful for visualizing time-frequency information, spectra, and pole-zero locations. For example, you can interactively design a filter by graphically placing the poles and zeroes in the z-plane.

The Signal Processing Toolbox provides two GUIs:

FDATool is a comprehensive tool for designing and analyzing digital filters that helps you:

- Access most FIR and IIR filter design methods in the toolbox using a simplified, graphical interface
- Analyze filters by exchanging magnitude, phase, impulse, and step responses and by calculating group delay and pole-zero plots
- Import previously designed filters and filter coefficients that you have stored in the MATLAB workspace. Export filter coefficients
- Access additional filter design methods and quantization features of the Filter Design Toolbox (when that optional product is installed)
- Print filter response directly from the GUI with the option to annotate plots





Specgramdemo is a user-friendly GUI that provides interactive calculations of a signal's time-frequency distribution.

SPTool is a suite of GUI tools providing access to many of the signal, filter, and spectral analysis functions that helps you:

- Measure and analyze the time-domain information of one or more signals and send audio signal to the PC's sound card
- Design and edit FIR and IIR filters of various lengths and types and with standard (lowpass, highpass, bandpass, bandstop, and multiband) configurations, as well as design filters by graphically placing poles and zeroes in the z-plane
- View the characteristics of a designed or imported filter, including its magnitude response, phase response, group delay, pole-zero plot, impulse response, and step response
- Apply the filter to a selected signal

- Graphically analyze frequency-domain data using a variety of spectral estimation methods, including Burg, FFT, multitaper (MTM), MUSIC, eigenvector, Welch, and Yule-Walker AR

An Interactive Demo

The Signal Processing Toolbox provides **specgramdemo**, a user-friendly GUI that interactively calculates a signal's time-frequency distribution. **Specgramdemo** presents:

- The original time series data
- The spectrogram of the input signal
- The power spectral density of the input signal
- A colorbar indicating the color scale of the spectrogram
- A signal panner that lets you focus in and out on the signal

- A crosshair locator that locates individual data points on the spectrogram

You can evaluate time/frequency information in the spectrogram by using the signal panner or the crosshair locator. This will allow you to locate data points in the spectrogram. They will display and interactively update a frequency slice of the input signal, a time slice of signal, and a readout of time and frequency values.

You can call the **specgramdemo** from the MATLAB command line by typing **specgramdemo(y, FS)** where **y** is the input signal and **FS** is the signal's sampling rate. Context-sensitive help is available for **specgramdemo**.

Product Requirements

The Signal Processing Toolbox runs on all MathWorks supported platforms. It requires MATLAB 6. ■

For demos, application examples, tutorials, user stories, and pricing:

- Visit www.mathworks.com
 - Contact The MathWorks directly
- | | |
|-------------|----------------------|
| US & Canada | 508-647-7000 |
| Benelux | +31 (0)182 53 76 44 |
| France | +33 (0)1 41 14 67 14 |
| Germany | +49 (0)89 995901 0 |
| Spain | +34 93 362 13 00 |
| Switzerland | +41 (0)31 954 20 20 |
| UK | +44 (0)1223 423 200 |

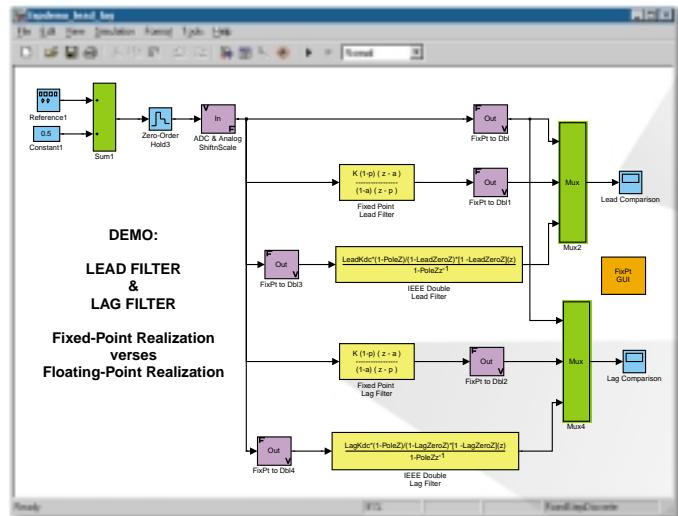
Visit www.mathworks.com to obtain contact information for authorized MathWorks representatives in countries throughout Asia Pacific, Latin America, the Middle East, Africa, and the rest of Europe.

Fixed-Point Blockset 3

for modeling, simulation, testing, and automatically generating pure integer code for fixed-point applications

The Fixed-Point Blockset allows engineers to efficiently design control systems and digital filters that will be implemented using fixed-point arithmetic. A block diagram containing detailed fixed-point information about the system model is constructed in Simulink®. You can perform a bit-true simulation to observe the effects of limited range and precision.

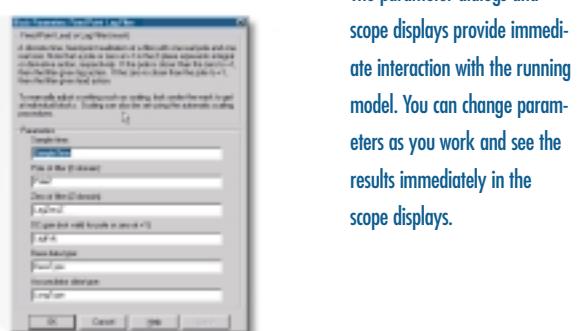
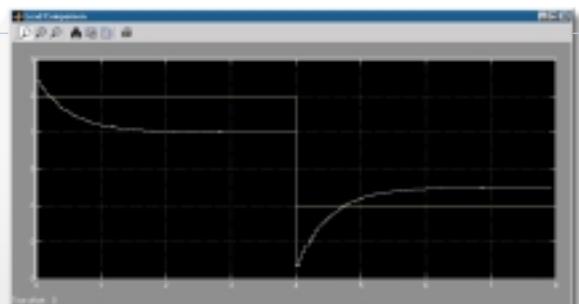
Simulations are automatically instrumented to log overflows, saturations, and signal extremes. Tools are provided to automate scaling decisions and to compare the fixed-point implementation against a floating-point benchmark. When combined with Real-Time Workshop®, an efficient, integer-only C code representation of the design can be automatically generated. This C code can be used in a production target or for rapid prototyping. When Real-Time Workshop Embedded Coder is used, real-time C code can be generated for use on an integer production, embedded target.



With the Fixed-Point Blockset, you can compare fixed-point and floating-point realizations of filters.

KEY FEATURES

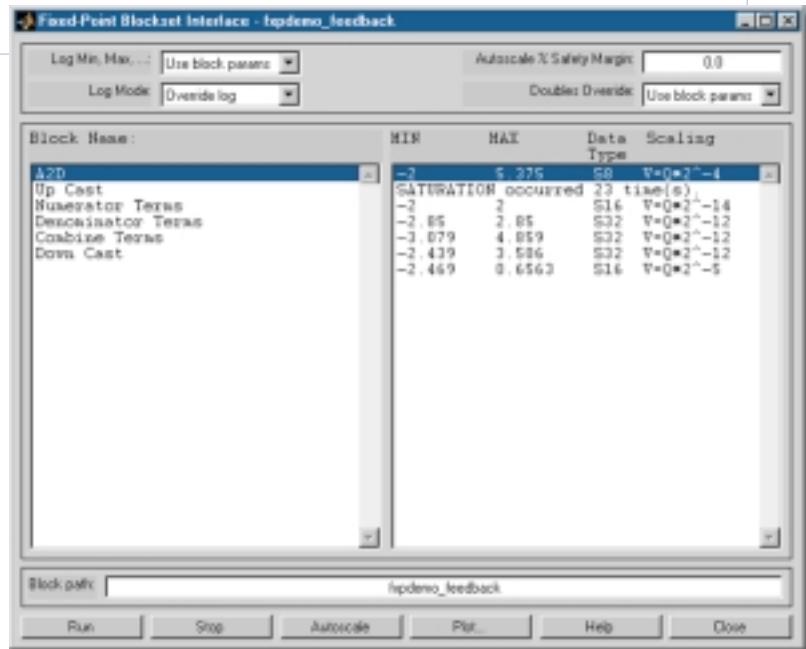
- Supports fixed-point bit-widths from 1 to 128 bits for bit-true simulation
- Supports complex numbers, and single, double, and custom floating-point types
- Provides automatic scaling tools. Allows data type and scaling to be configured independently for each signal
- Allows modes for rounding and overflow handling to be independently configured for each block
- Eases layout of models with high-level blocks (including filters) such as tapped delay line, dot product, matrix gain, and FIR filter. Automatic layout of fixed-point filters using the DSP Blockset Filter Realization Wizard
- Obtains floating-point simulations for debugging and benchmarking



The parameter dialogs and scope displays provide immediate interaction with the running model. You can change parameters as you work and see the results immediately in the scope displays.

KEY FEATURES (CONTINUED)

- Reduces specification overhead with mechanisms that automatically propagate data type and scaling choices
- Allows sensor-driven and actuator-driven scaling to be used
- Generates code that includes every operation (such as shifts) needed to account for differences in fixed-point locations
- Allows non-standard bit-widths to support implementation of custom hardware designs such as ASICs or FPGAs
- Provides full compatibility with rapid prototyping systems for bit-true emulation on floating-point chips
- Exploits Simulink capabilities for closed-loop simulation of fixed-point digital system interaction with analog systems



The Fixed-Point Blockset GUI provides convenient access to global overrides and min/max logging settings, logged min/max data, the automatic scaling tool, and the signal comparison tool.

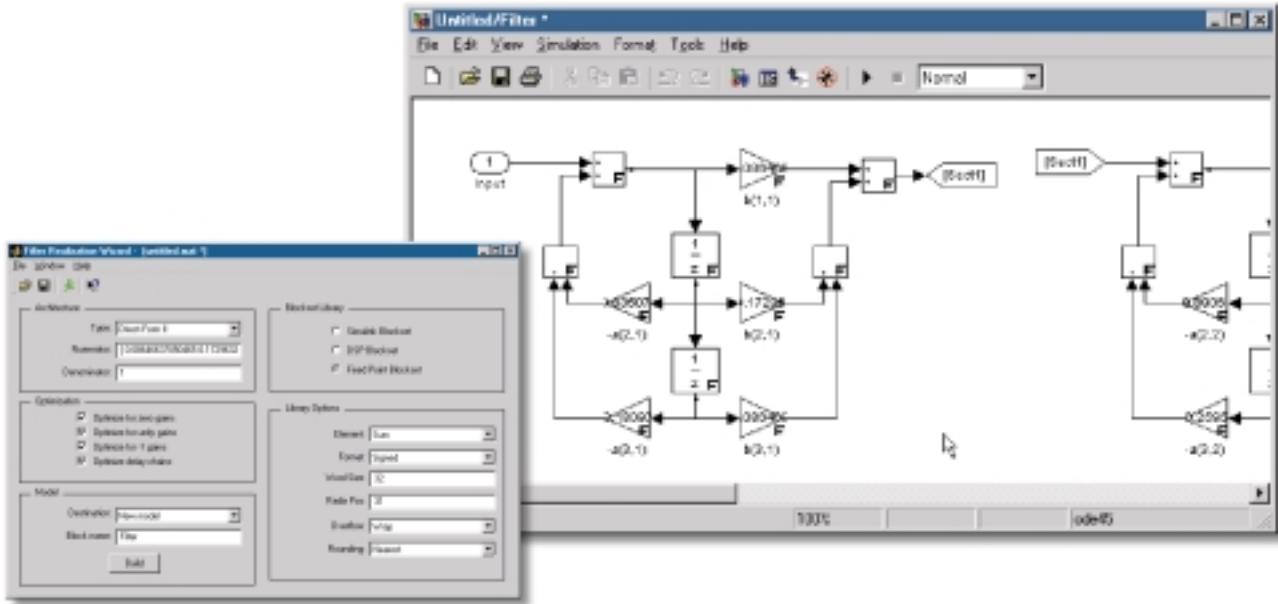
The Fixed-Point Blockset streamlines the process of developing embedded software for use on fixed-point processors. It provides blocks to support operations normally used in embedded control systems and digital filtering, including:

- Arithmetic blocks—for multiplying, dividing, adding, or subtracting input values
- Delay blocks—for describing memory and states
- Conversion blocks—for converting from one data type to another
- Decision logic blocks—for relational operations, Boolean logic, bitwise logic, and switches
- Nonlinear blocks—for saturation, dead zone, and relay
- Look-up table blocks—for approximating one- or two-dimensional functions
- Discrete-time blocks—for creating a discrete-time system
- Filter blocks—for modeling discrete-time filters
- Additional key blocks that support fixed-point designs such as multiplexers, demultiplexers, scopes, and displays are available in Simulink

Digital Filter Design

The Fixed-Point Blockset lets you design discrete-time digital filters using one of many transformation methods. When you combine the Fixed-Point Blockset and the DSP Blockset, you can use the DSP Filter Realization Wizard to automatically lay out large or small fixed-point filters in a variety of useful realizations.

The Fixed-Point Blockset lets you fine-tune realizations, reducing costs and improving signal quality. When you have achieved the desired performance, you can use Real-Time Workshop to generate rapid



Customized single-precision and fixed-point filters can be created using the Filter Realization Wizard of the DSP Blockset. Each filter is constructed from the FixPt Sum, FixPt Gain, and FixPt Unit Delay blocks from the Fixed-Point Blockset.

prototyping C code and evaluate the code's performance with respect to your system's real-time constraints. You can alter the model based on feedback from the rapid prototyping system. When you are satisfied with the performance of the rapid prototyping system, you can use the model and the generated code as specifications or as components of your implementation.

Automatic Scaling

The minimum and maximum values encountered during a simulation can be logged to the MATLAB® workspace. These values can then be accessed by the automatic scaling script, `autofixexp`. This script automatically changes the scaling for signals in the model.

Scaling is automatically modified to cover the simulation range and optimize precision. If the output data type is a generalized fixed-point number, then you have the

option of locking the output scaling. Otherwise, scaling will be automatically optimized.

Fixed-Point Blockset GUI

The Fixed-Point Blockset GUI allows you to easily control the parameters associated with automatic scaling and display the simulation results for a given model. The Fixed-Point Blockset GUI lets you:

- Turn logging on or off for all blocks
- Override the output data type with doubles for all blocks
- Invoke automatic scaling
- Run the simulation
- Display the data type and scaling results
- View logged overflows and saturations
- View logged minimum and maximum values

Data Type Support

The Fixed-Point Blockset offers extensive support for data types. The blocks support inheritance and propagation of data type and scaling information between blocks in the model. This capability can be used to significantly reduce the amount of configuration information you need to supply.

Code Generation Support

All blocks support code generation with Real-Time Workshop. The following features are supported:

Pure integer code. All blocks generate pure integer code when used with fixed-point signals and parameters. If blocks handle floating-point signals or parameters, then Real-Time Workshop generates the necessary floating-point code.

Languages. ANSI C

Storage class of variables. Typically, code can be generated for any word size from 1-32 bits or 1-64 bits. The upper limit is determined by the largest integer size that the target compiler provides. Odd sizes are emulated; this provides great flexibility in rapid prototyping. The most efficient code is produced when the target's native sizes are used.

Storage class of parameters. Code can be generated for parameters from 1-32 bits, in most cases. The larger sizes (such as 64 bits) are currently not supported.

Rounding modes. The following four rounding modes are supported: Toward Zero, Toward Nearest, Toward Ceiling, and Toward Floor. Toward Floor generates the most efficient code in most cases.

Overflow handling. Saturation mode and Wrapping mode are supported. Wrapping must be selected for Real-Time Workshop to exclude saturation code.

Blocks. All blocks generate code for all operations, with the following exceptions: the FixPt Look-Up Table and FixPt Look-Up Table (2-D) blocks generate code for all look-up methods except extrapolation.

Scaling. Radix point-only scaling is supported. Slope/bias scaling is supported for all blocks except when it leads to highly inefficient code.

Demos

The Fixed-Point Blockset provides basic and advanced demos that allow you to explore the capabilities of the product by changing

block parameters and observing the effects of those changes. Basic demos illustrate the basic functionality of the Fixed-Point Blockset. These can be accessed through the Fixed-Point Library's Demos block.

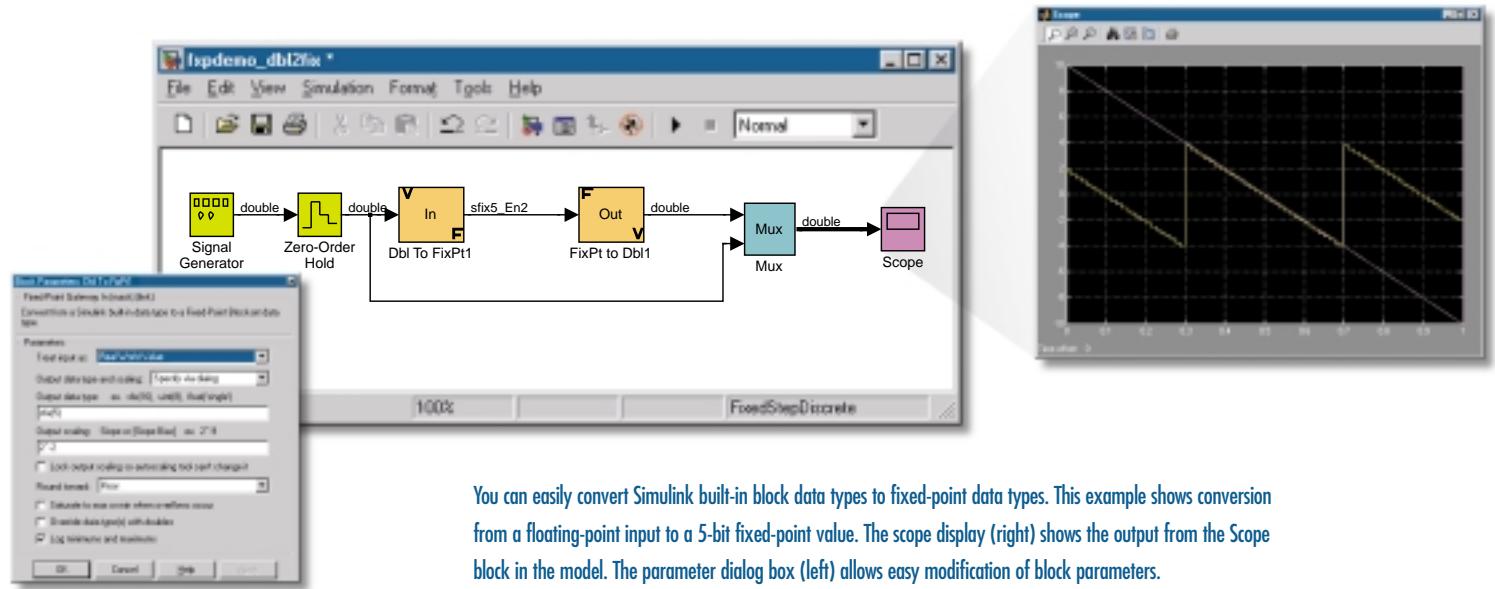
Advanced demos illustrate the functionality of systems and filters built with fixed-point blocks. The output of these demos is analogous to that of built-in Simulink blocks with identical input. Advanced demos can be accessed through the Fixed-Point Library's Filters & Systems Examples block or by typing `fixptsys` at the command line.

Basic Demos

Converting Doubles to Fixed-Point Value	Convert a double-precision value to fixed-point value.
Converting Fixed-Point to Fixed-Point	Convert a fixed-point value to another fixed-point value.
Inherit Fixed-Point to Fixed-Point Conversion	Convert a fixed-point value to an inherited fixed-point value.
Fixed-Point Sine Wave Example	Add and multiply two fixed-point values.
Automatic Scaling in Feedback Control	Simulate a fixed-point feedback design.

Advanced Demos

Derivatives	Compare output from the FixPt Derivative blocks to output from analogous Simulink derivatives built using the Discrete Filter and Transfer Fcn blocks.
Integrators	Compare output from the FixPt Integrator blocks (Trapezoidal, Backward, and Forward) to output from analogous Simulink integrators supported by the Discrete Integrator block.
Lead and Lag	Compare output from the FixPt Lead and Lag Filter block to Output from analogous Simulink filters built using the Discrete Filter block.
State-Space	Compare output from the FixPt State-Space Realization block to output from the analogous built-in Simulink block.
Fixed-Point Direct Form Filters	Model fixed-point time-varying and time-invariant filters using direct form realizations.



Sample Blocks

	Fixed-point absolute value		Fixed-point gateway in		Fixed-point product
	Fixed-point bitwise operator		Fixed-point gateway in inherited		Fixed-point relational operator
	Fixed-point constant		Fixed-point gateway out		Fixed-point relay
	Fixed-point conversion		Fixed-point integer delay		Fixed-point saturation
	Fixed-point inherited conversion		Fixed-point logical operator		Fixed-point sign
	Fixed-point dead zone		Fixed-point look-up table		Fixed-point sum
	Fixed-point dot product		Fixed-point look-up table (2-D)		Fixed-point switch
	Fixed-point dynamic look-up table		Fixed-point matrix gain		Fixed-point tapped delay
	Fixed-point finite impulse response filter		Fixed-point minmax		Fixed-point unary minus
	Fixed-point gain		Fixed-point multiport switch		Fixed-point unit delay
					Fixed-point zero-order hold

Fixed-Point Blockset Specifications

Supported Data Types

- Fixed-point:
 - Integer, fractional, and generalized fixed-point
 - Unsigned and two's complement formats
 - Word size from 1 to 128 bits
- Floating-point:
 - IEEE-style singles and doubles
 - A nonstandard IEEE-style data type, mantissa 1 to 52 bits, exponent 2 to 11 bits

Supported Overflow Handling Operations

- Saturate
- Wrap

Supported Scaling Modes

- General scaling modes:
 - Radix point-only
 - Noncontiguous radix point with fixed-point word
 - Slope/bias
- Constant scaling:
 - Constant vector scaling
 - Constant matrix scaling

Supported Rounding Methods

- Toward Zero
- Toward Nearest
- Toward Ceiling
- Toward Floor

System Requirements

For simulation

MATLAB and Simulink

For code generation

MATLAB, Simulink, Real-Time Workshop,

For embedded code generation

MATLAB, Simulink, and Real-Time Workshop, Real-Time Workshop Embedded Coder

To create an executable from the generated code, you must have the appropriate C compiler and linker.

Related Products

Control System Toolbox—a MATLAB toolbox for implementing the most prevalent classical and modern linear control techniques for the design and analysis of automatic control systems

DSP Blockset—a Simulink block library for the design, simulation, and prototyping of digital signal processing systems

For more information on these and other related products visit www.mathworks.com.

Platforms Supported

Windows 95, 98, 2000 Windows NT 4.0, and UNIX systems

For demos, application examples, tutorials, user stories, and pricing:

• Visit www.mathworks.com

• Contact The MathWorks directly

US & Canada 508-647-7000

Benelux +31 (0)182 53 76 44

France +33 (0)1 41 14 67 14

Germany +49 (0)89 995901 0

Spain +34 93 362 13 00

Switzerland +41 (0)31 954 20 20

UK +44 (0)1223 423 200

Visit www.mathworks.com to obtain contact information for authorized MathWorks representatives in countries throughout Asia Pacific, Latin America, the Middle East, Africa, and the rest of Europe.

March 30, 2001

Product Datasheet

Xilinx Inc.
 2100 Logic Drive
 San Jose, CA 95124
 Phone: +1 408-559-7778
 Fax: +1 408-559-7114
 E-Mail: logicore@xilinx.com
 URL: <http://www.xilinx.com/>

Introduction

Xilinx announces the System Generator™ V1.1 for Simulink®. The System Generator enables you to develop high-performance DSP systems for Xilinx Virtex™/E, Virtex™-II, and Spartan®-II FPGAs using The MathWorks products, the MATLAB® and Simulink.

Features

- System-level abstraction of FPGA circuits
 - Visual data-flow paradigm
 - Bit-/cycle-true Simulink library for common functions
 - Sample rate vs. explicit clocking
- Automatic code generation from a Simulink model
 - Synthesizable VHDL for a Xilinx Blockset model
 - Simulink hierarchy is preserved in VHDL
 - HDL testbench
 - ModelSim script files
- Support for user-created Simulink library elements using the Black Box
- Transparent access to Xilinx IP via the Xilinx CORE Generator™ System
 - FPGA designs are generated using Xilinx LogiCORE algorithms ensuring that the most efficient code is being generated.

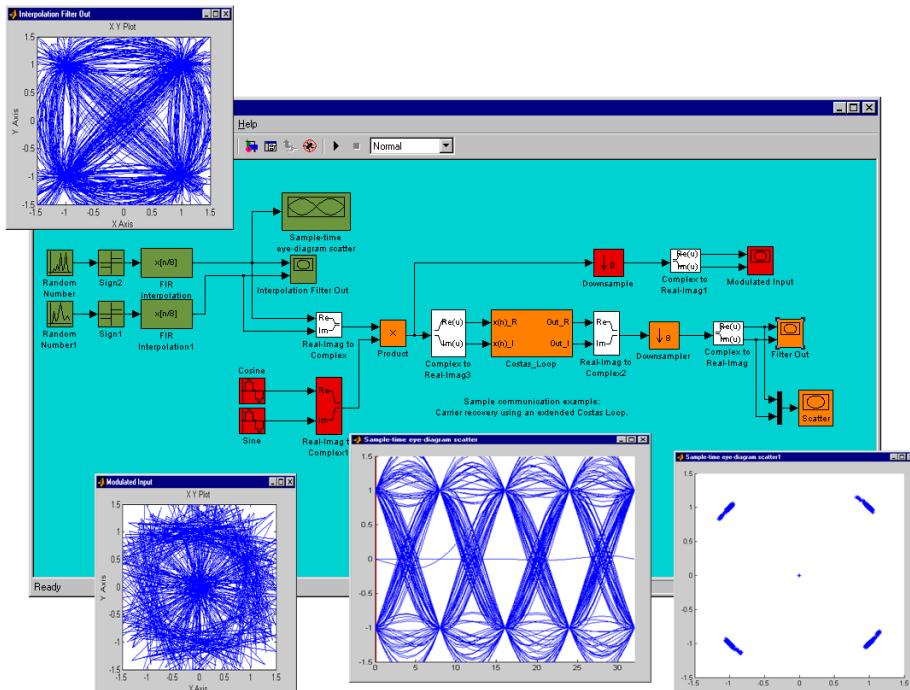


Figure 1: Xilinx System Generator for Simulink

Functional Description

Figure 2 shows the general flow of the System Generator functionality as it fits in with The MathWorks and Xilinx implementation software tools. The Simulink Block Library contains blocksets used to model systems within the Simulink GUI. The System Generator software provides an additional blockset to the library: the Xilinx Blockset. As shown in the flow diagram, the blockset elements can be instantiated within a Simulink model (within the MATLAB environment) just like any other Simulink block. You can model and simulate with the Xilinx Blockset as you are accustomed to doing within Simulink.

When you are through modeling you can then add the System Generator token at the level of hierarchy you would

like to generate code. When the HDL code generation software is invoked, VHDL code, cores, and test vectors are generated according to system parameters defined within the model. The cores are created using the Xilinx CORE Generator. The VHDL source can be compiled and simulated in a VHDL simulator, and an FPGA implementation can be obtained by applying a synthesis tool to the VHDL. After synthesis, the System Generator project can be run through the Xilinx implementation tools (build, map, place, and route) to produce a bitstream for download to an FPGA device.

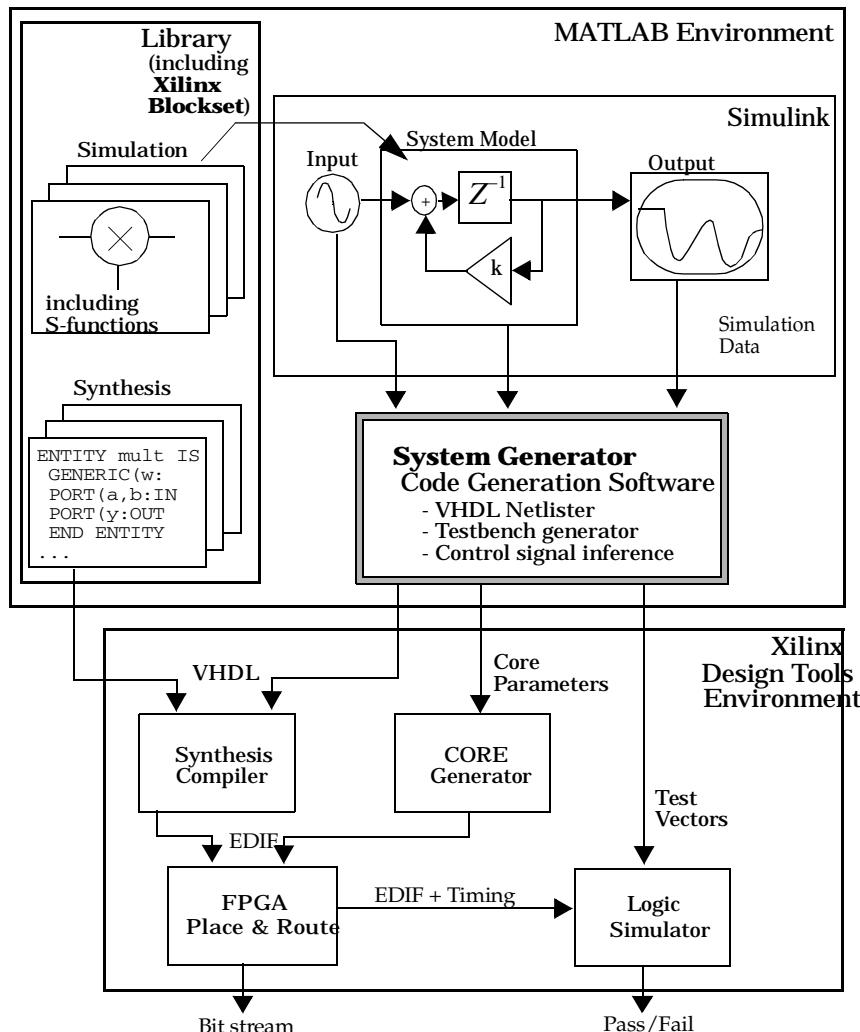


Figure 2: System Generator Flow Diagram

Xilinx Blockset

The Xilinx Blockset is a major component of this release of the System Generator.

Like other Simulink Blocksets, the Xilinx Blockset contains elements that can be used to build simulation models. In addition, models built from the Xilinx Blockset can be translated using the System Generator into synthesizable VHDL circuits. After the System Generator has been installed, the Xilinx Blockset will be visible in the Simulink Library Browser.

Xilinx Blockset elements include VHDL models and association with Xilinx LogiCOREs. These models enable VHDL code to be generated for Simulink designs made up of Xilinx blocks.

Currently, the Xilinx Blockset contains the following elements:

- Basic Elements
 - System Generator
 - Black Box
 - Concat
 - Constant
 - Convert
 - Counter
 - Delay
 - Down Sample
 - Get Valid Bit
 - Mux
 - Register
 - Set Valid Bit
 - Slice
 - Sync
 - Up Sample
- DSP
 - DDS
 - FFT
 - FIR
- Math
 - Accumulator
 - AddSub
 - CMult
 - Inverter
 - Logical
 - Mult
 - Negate
 - Relational
 - Scale
 - Shift
 - SineCosine
 - Threshold
- MATLAB I/O
 - Clear Quantization Error
 - Display
 - Enable Adapter
 - Gateway In

- Gateway Out
 - Quantization Error
 - Sample Time
- Memory
 - Dual Port RAM
 - FIFO
 - ROM
 - Single Port RAM

System Generator Token

A special Xilinx Blockset element is the System Generator token. This token can be selected from the Simulink Library Browser, from within the basic elements of the Xilinx Blockset.

The System Generator token invokes the Code Generation Software, the second major portion of the tool. By placing the System Generator token on your Simulink project sheet, you can generate VHDL code and cores for all the Xilinx Blockset elements on that sheet and on any sheets beneath it in its project hierarchy. This also enables you to simulate a mixed mode design and then generate a digital realization of the digital portion of the design by placing the token in the digital hierarchy only.

VHDL Code Generation Software

The System Generator includes software to enable translation and simulation. The translation software is invoked from Simulink and provides an interface to the Xilinx FPGA software. This interface includes a compiler to translate a Simulink model into a synthesizable VHDL model, including generation of Xilinx cores where appropriate. FPGA designs are generated using Xilinx LogiCOREs, ensuring that the most efficient code is being generated.

Simulation software provides C++ fixed-point arithmetic libraries to support Xilinx Blockset and user-written, run-time parameterizable Simulink S-functions, including support for rounding and overflow. The Simulation software set also includes classes which allow a user to create a C++ executable model and easily incorporate it as a Simulink S-function for simulation.

Testbench Generation

When enabled, a VHDL testbench "wrapper" file is created for your generated designs. The testbench "wrapper" file is named to match the top-level VHDL file generated for your project. For example, if your top-level VHDL file is named `integrate`, the System Generator will create a wrapper file `integrate_testbench.vhd`. The top level of the project is determined by the name of the Simulink sheet from which you have invoked the System Generator token. You may run the testbench (which uses these test vectors) in a behavioral simulator such as ModelSim from Model Technology. It should report any discrepancies between the Simulink simulation and the VHDL simulation. You can

verify the translation of your Simulink design using this method.

Black Box Token

The Xilinx Blockset "Black Box" token gives you the ability to instantiate your own specialized functions in your design, and subsequently into a generated model. Any Simulink subsystem may be treated as a "Black Box" if you so choose. You may want to build a model out of non-Xilinx blocks, or you may have a VHDL-representation of functionality that you wish to turn into a Simulink model. Similar to the System Generator token, the Black Box token may be placed on any Simulink subsystem, identifying the subsystem as a Black Box.

Documentation

When you have purchased the System Generator, you may access the online software manuals from the Xilinx home page (<http://www.xilinx.com>). The System Generator tool from Xilinx is released with the following documents:

- *System Generator Quick Start Guide*
- *System Generator Tutorial*
- *System Generator Reference Guide*
- *System Generator Datasheet*

The MathWorks Documentation

The MathWorks provides a printed documentation suite that you should have received with your purchase of MATLAB, and Simulink. The software manuals include the following:

- *Using MATLAB*
- *MATLAB New Features Guide*
- *Getting Started with MATLAB*
- *User's Guide for MATLAB Toolboxes*, including:
 - *Communications Toolbox*
 - *Signal Processing Toolbox*
 - *Wavelet Toolbox*
 - *Control System Toolbox*
 - *Image Processing Toolbox*
 - *Using Simulink*
 - *DSP Blockset User's Guide - for use with Simulink*

Related Information

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.

Copyright 1991-2000 Xilinx, Inc. All Rights Reserved.

MATLAB and Simulink are registered trademarks of The MathWorks, Inc.

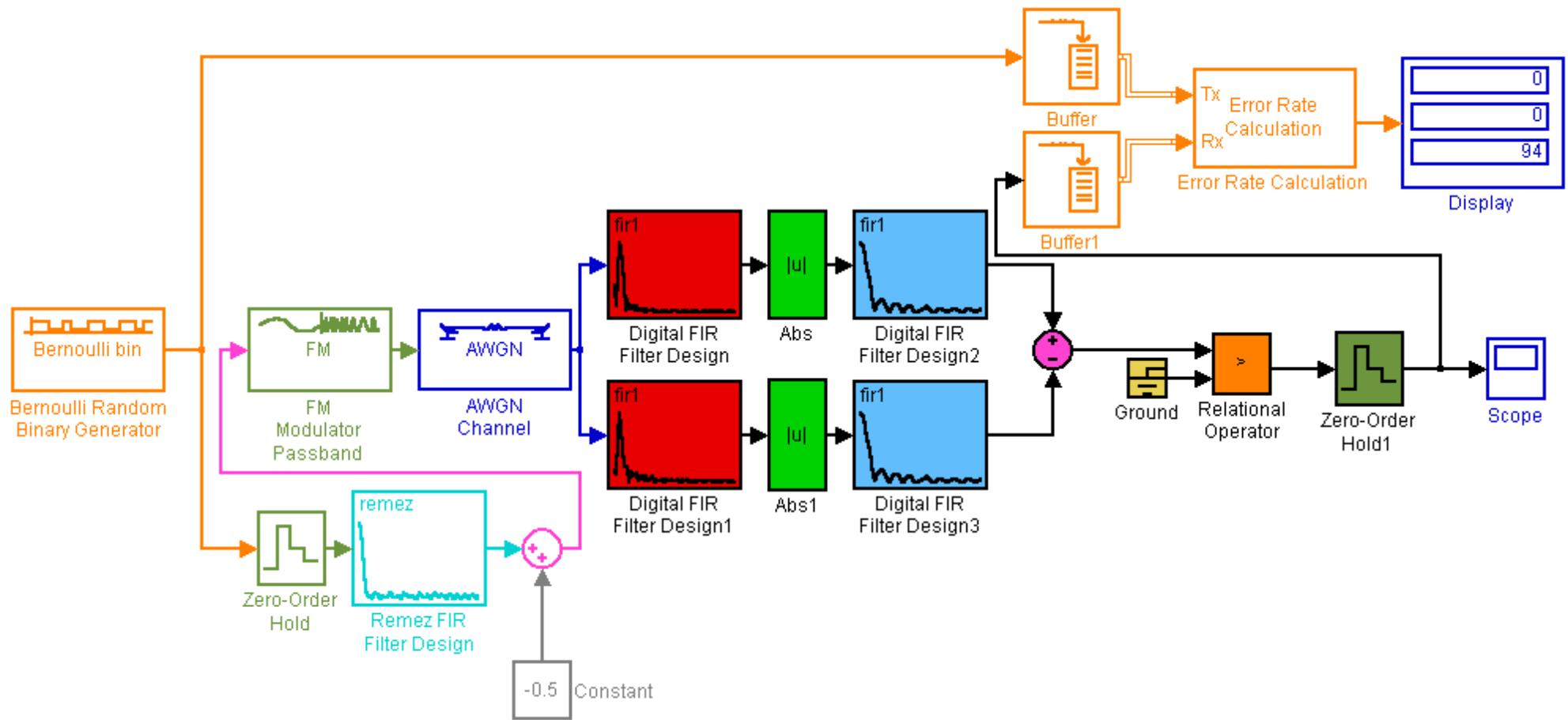
Ordering Information

The Xilinx System Generator is provided under *Xilinx Time-Based Software License Agreement*. For purchase, price, and availability information, please visit the Xilinx IP Center at www.xilinx.com/ipcenter or contact your local Xilinx Sales Representative.

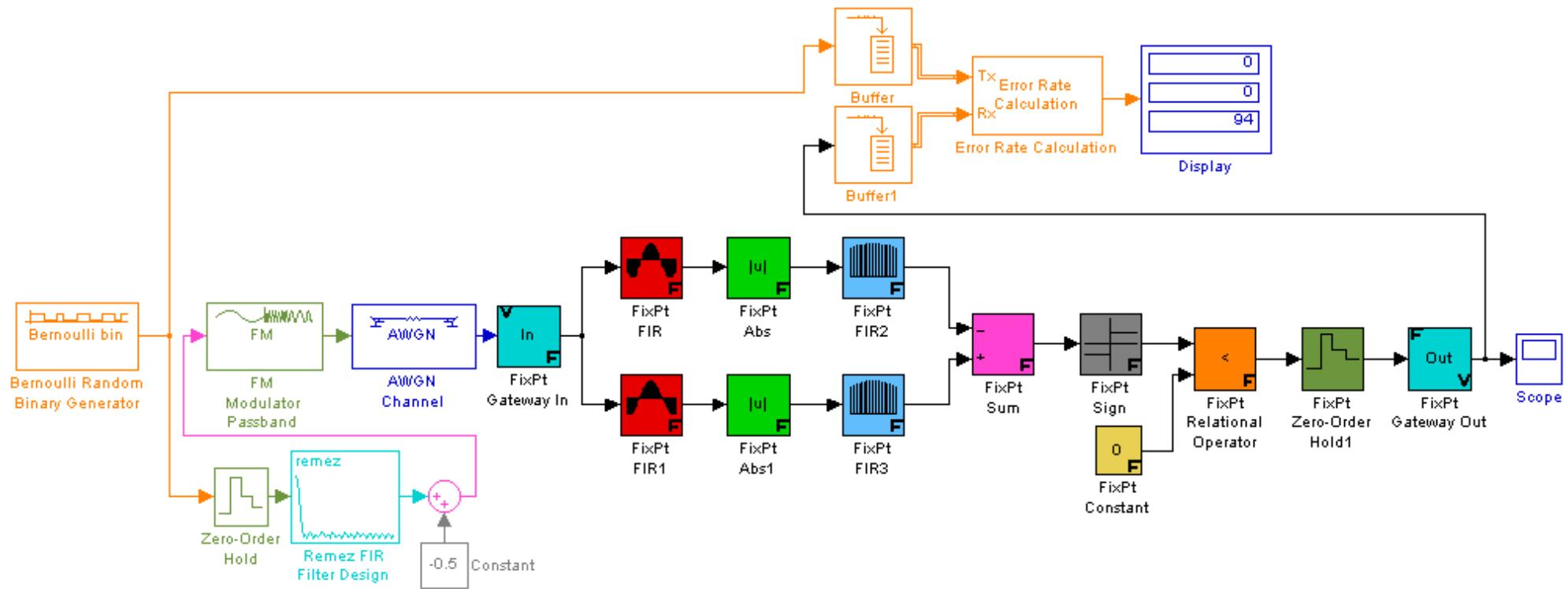
APPENDIX-B

SYSTEM MODELS & SIMULATION WAVEFORMS

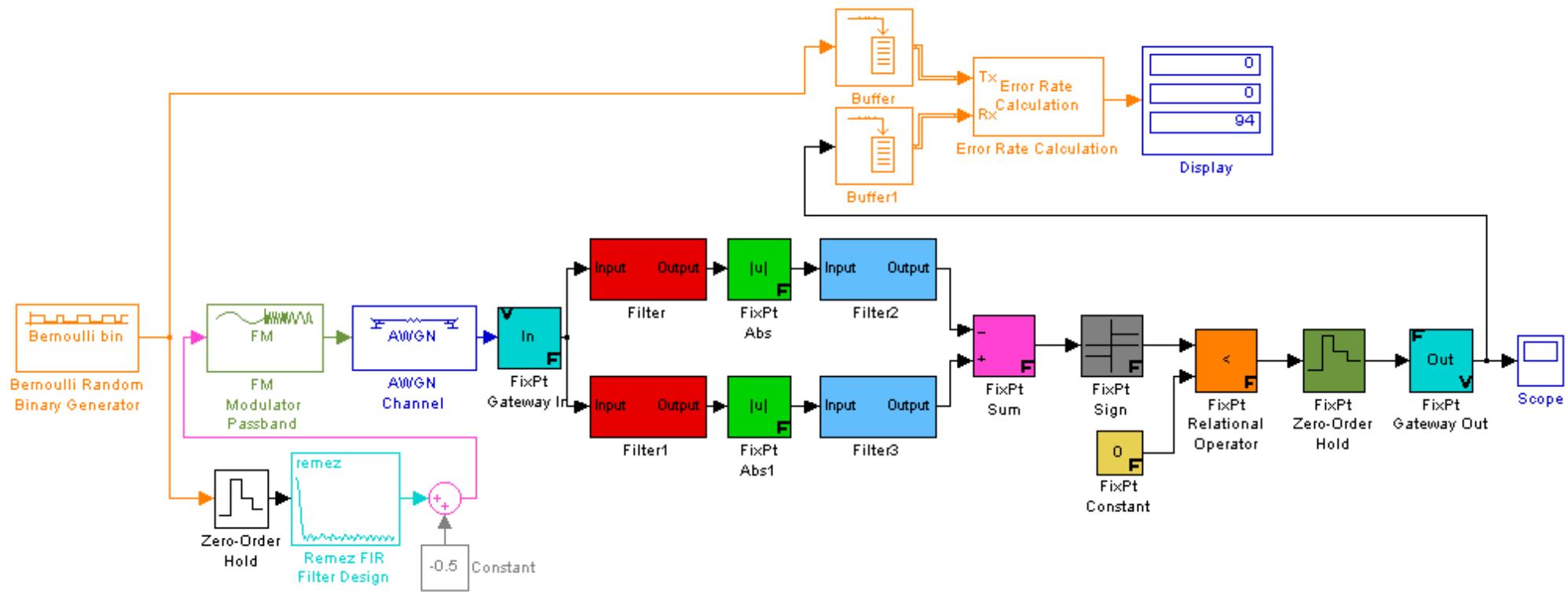
FLOATING-POINT SIMULATION MODEL OF THE GFSK MODULATOR/DEMODULATOR



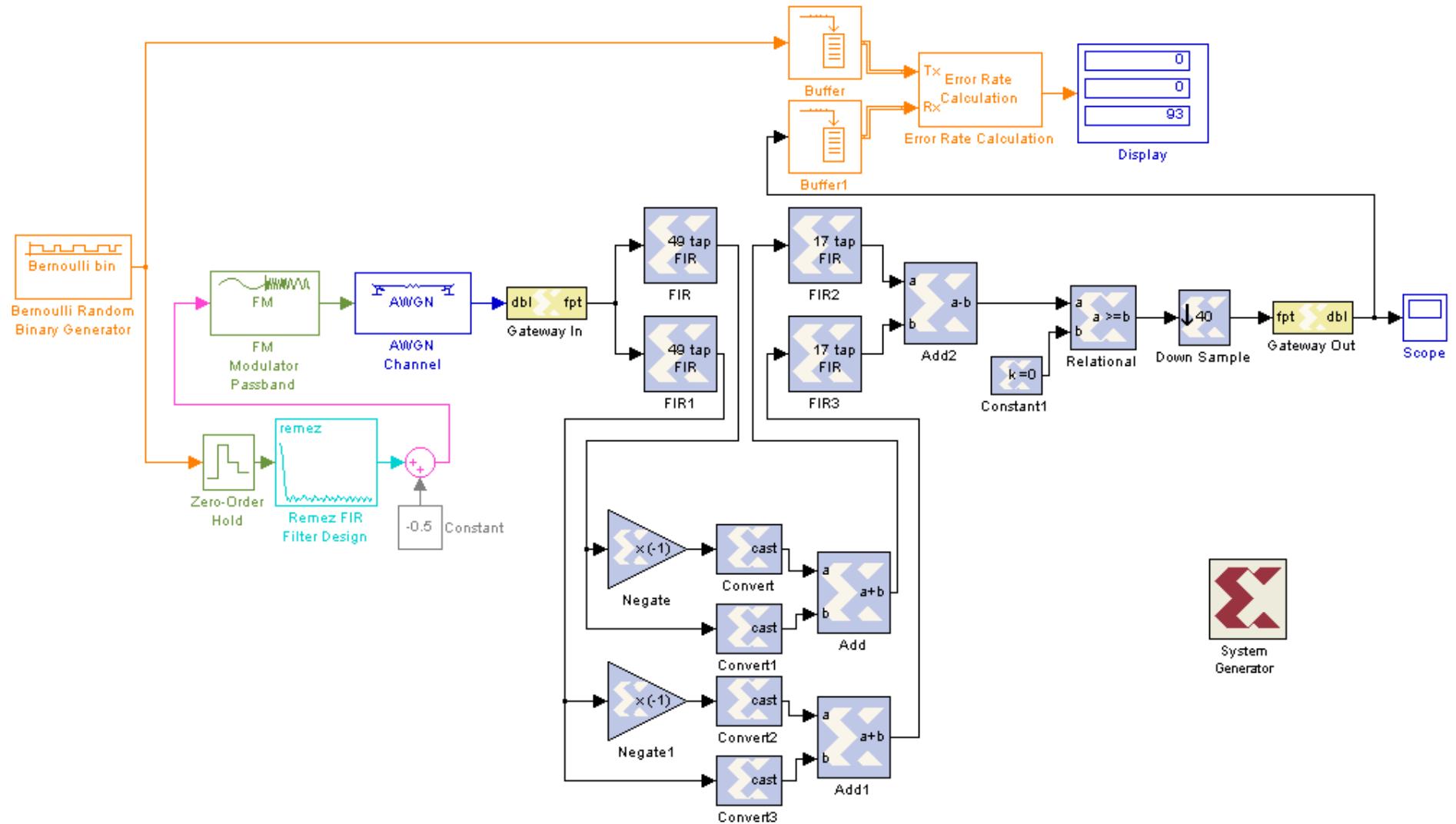
FIXED-POINT SIMULATION MODEL OF THE GFSK MODULATOR/DEMODULATOR



REALIZED FIXED-POINT SIMULATION MODEL OF THE GFSK MODULATOR/DEMODULATOR



XILINX SYSTEM GENERATOR TRANSFORMED SIMULATION MODEL OF THE GFSK MODULATOR/DEMODULATOR



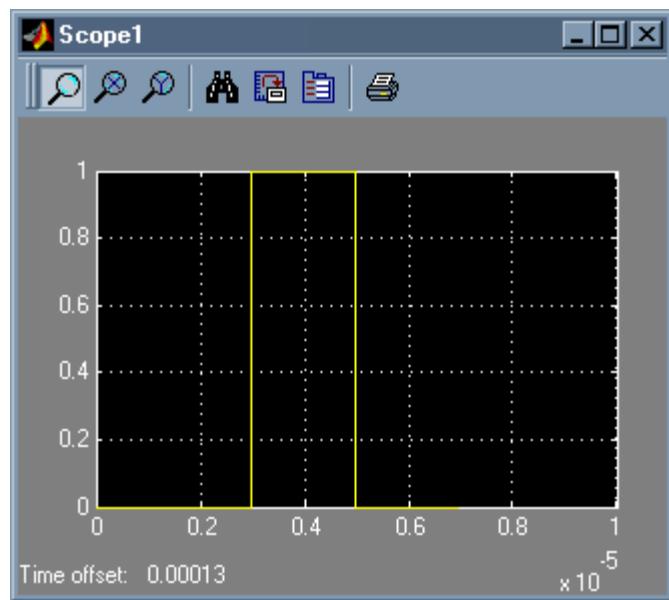


Figure B.1 Output after the Bernoulli Random Binary Generator Block

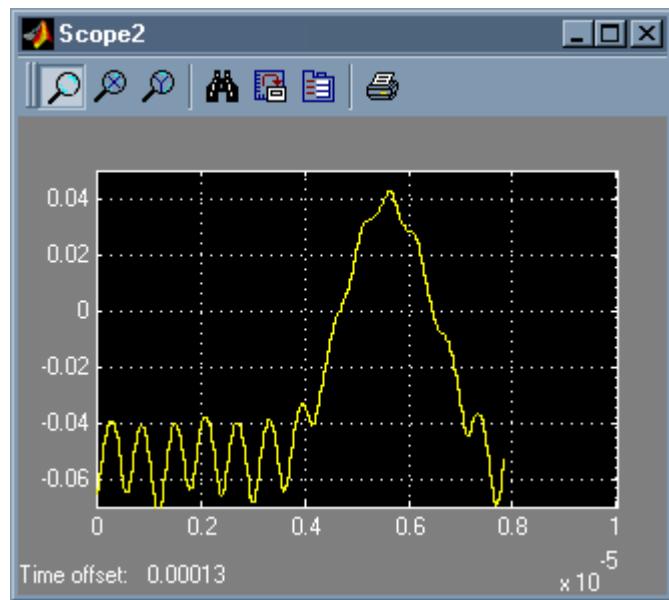


Figure B.2 Output after the Sum Block

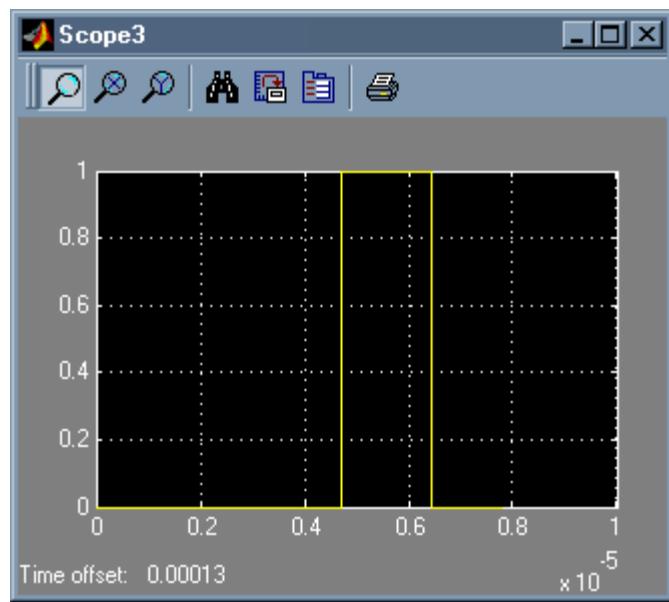


Figure B.3 Output after the Relational Operator Block

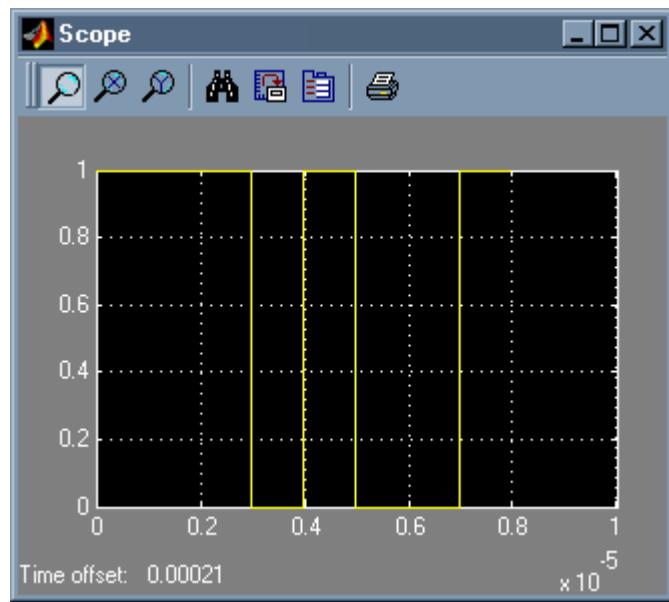
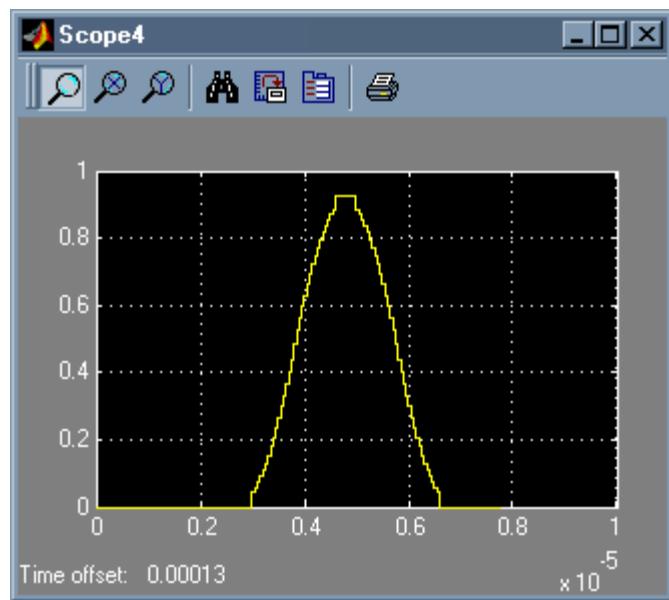
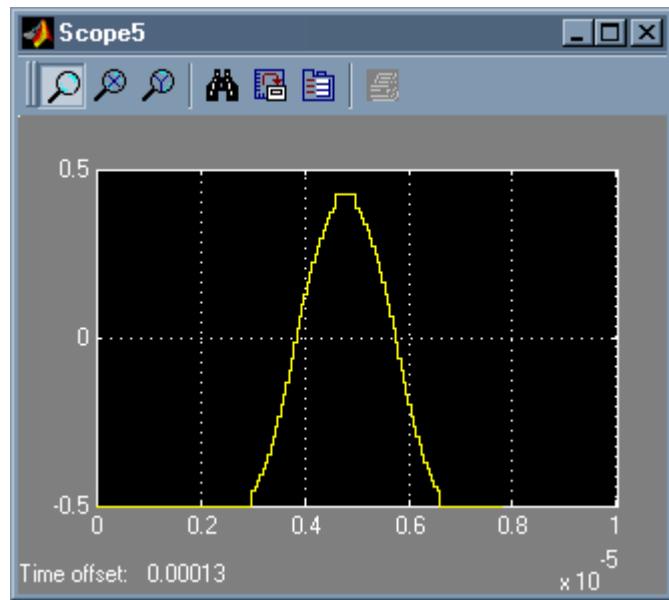


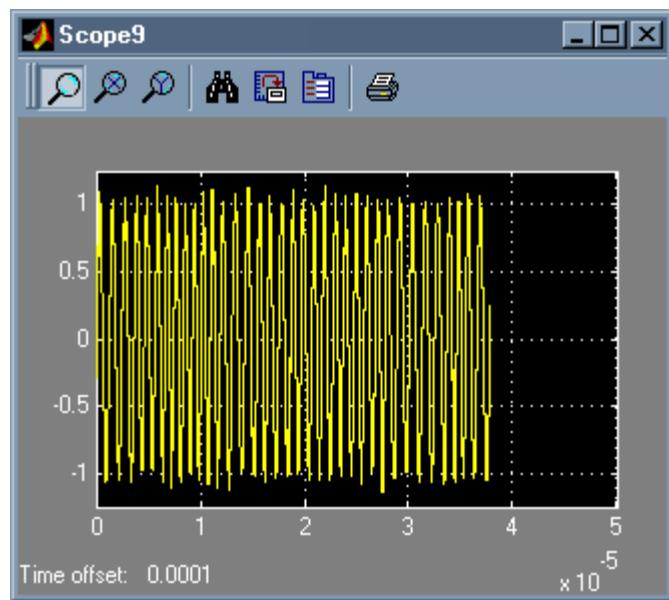
Figure B.4 Output after the Zero Order-Hold1 Block



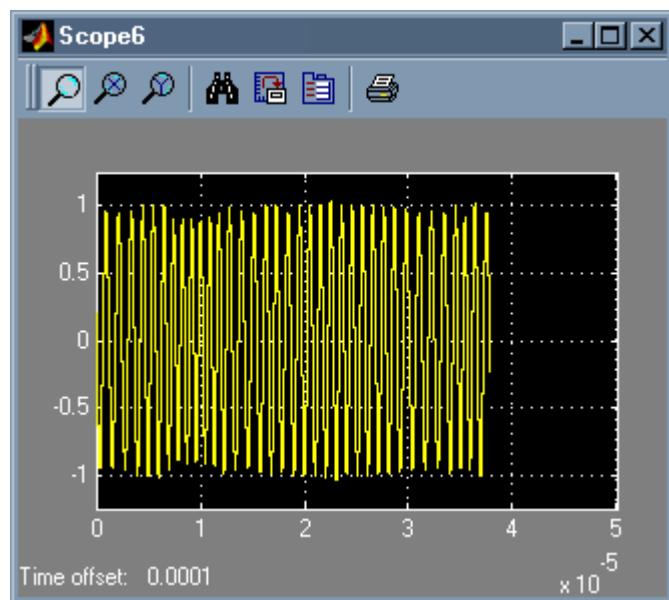
*Figure B.5 Output after the Remez FIR Filter Design Block
(note the half sinusoidal shape given to the rectangular pulse by the Gaussian Filter)*



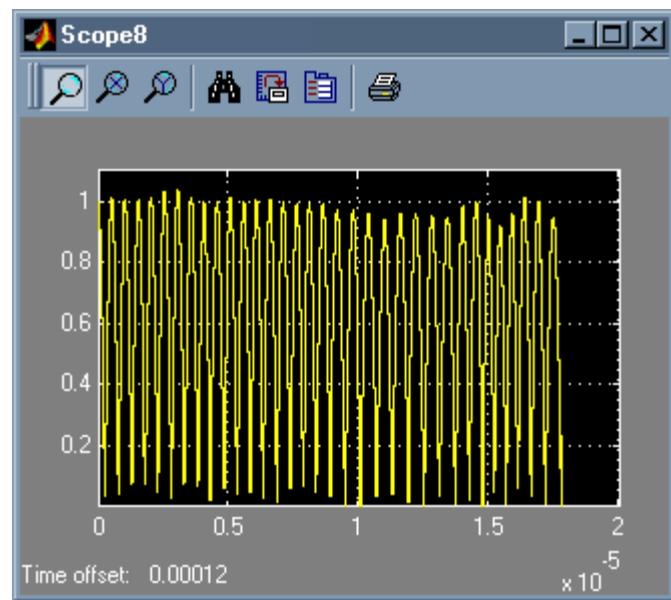
*Figure B.6 Output after the -0.5 Constant & Sum Blocks
(note the polar signal almost evenly balanced above and below the zero axis)*



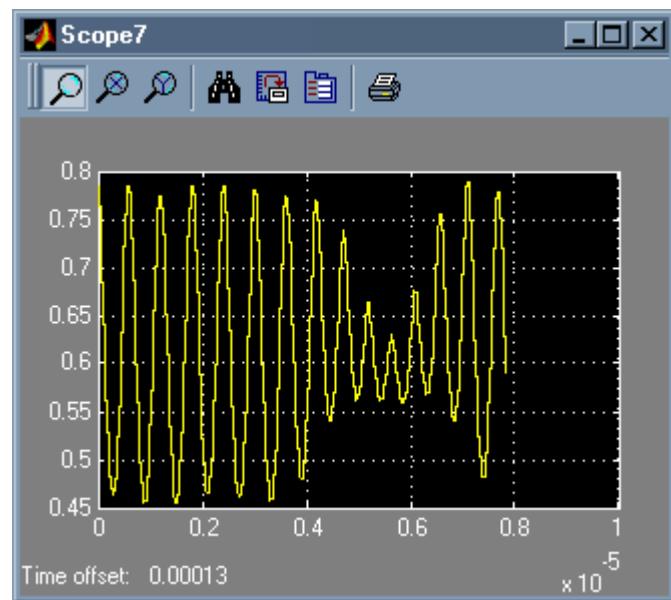
*Figure B.7 Output after the AWGN Channel Block
(note the Gaussian noise added to the Transmitted GFSK Modulated waveform sampled at 40 Msamples/sec)*



*Figure B.8 Output after the Digital FIR Filter Design1 Block
(note the amplitude variation in the envelope of the waveform due to Bandpass Filtering)*



*Figure B.9 Output after the Abs1 Block
(note the full-wave rectification action on the Bandpass filtered waveform)*



*Figure B.10 Output after the Digital FIR Filter Design3 Block
(note the Lowpass filtering action on the full-wave rectified waveform)*

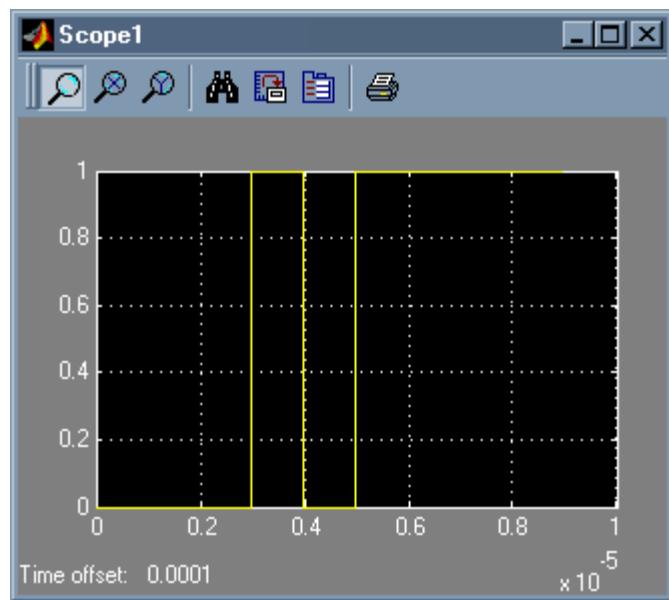


Figure B.11 Output after the Bernoulli Random Binary Generator Block

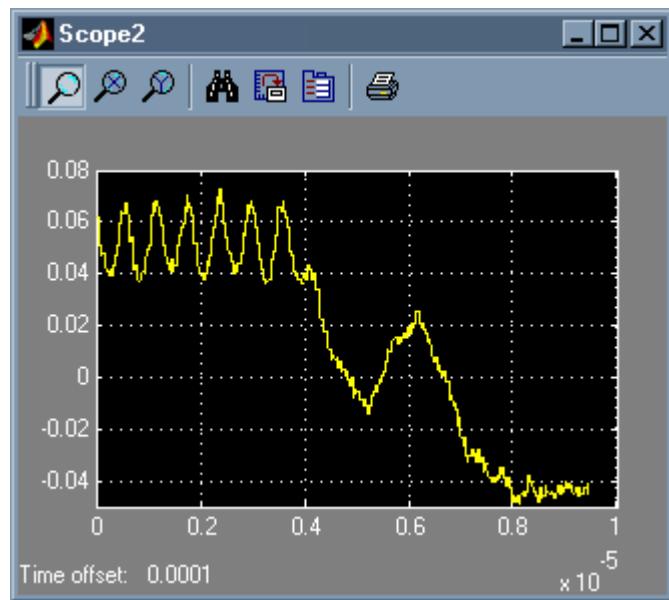


Figure B.12 Output after the FixPt Sum Block
(note the coarseness of the waveform due to quantization effects)

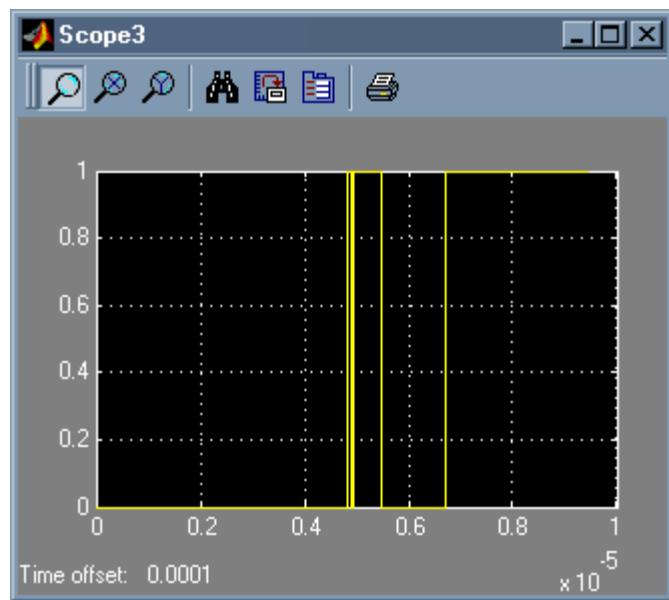


Figure B.13 Output after the FixPt Relational Operator Block
(note the jitter in the waveform due to quantization effects)

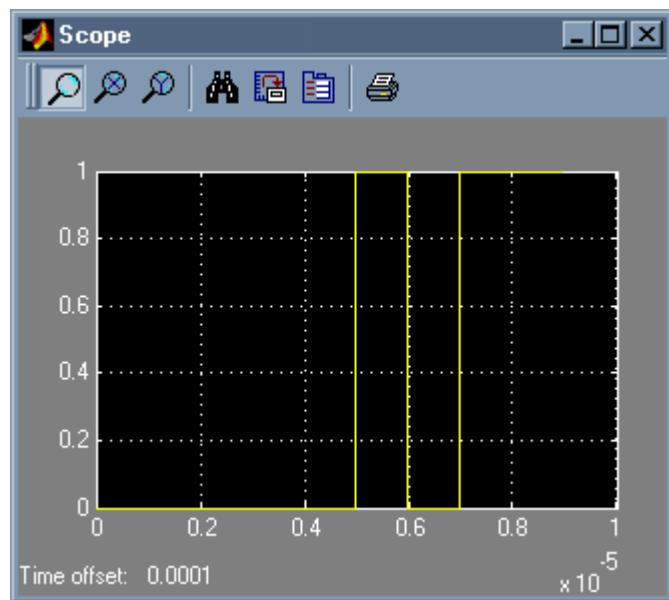


Figure B.14 Output after the FixPt Gateway Out Block

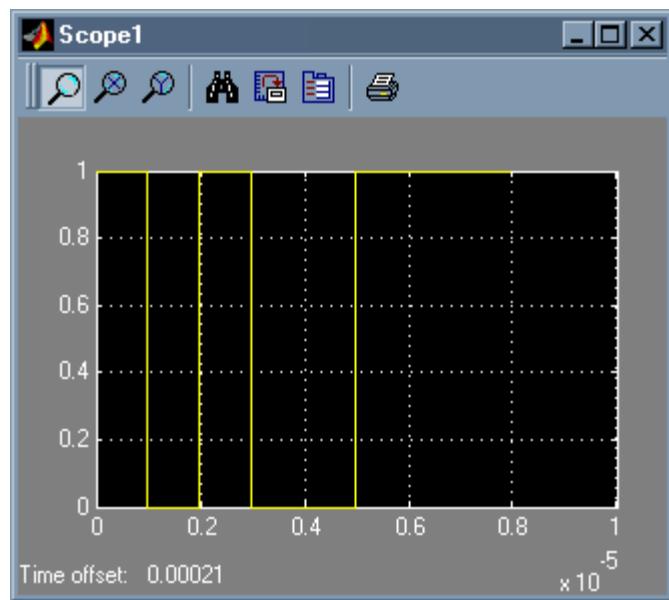


Figure B.15 Output after the Bernoulli Random Binary Generator Block

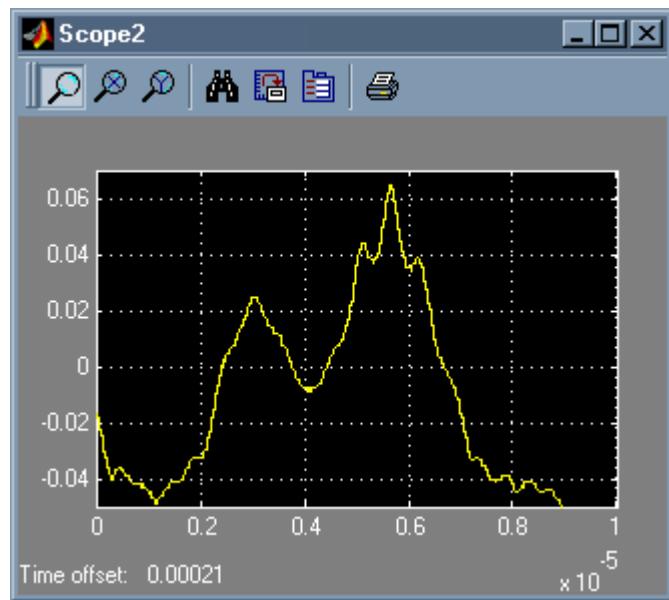
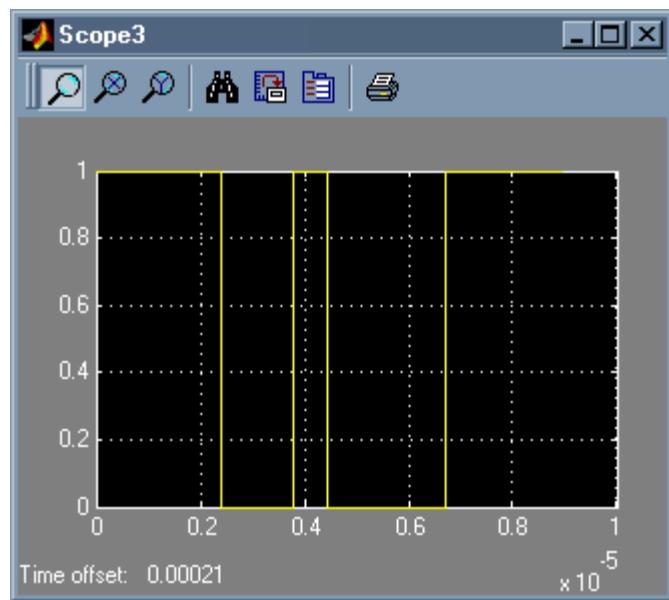


Figure B.16 Output after the FixPt Sum Block
(note the decrease in the coarseness of the waveform because of the realization structure)



*Figure B.17 Output after the FixPt Relational Operator Block
(note the absence of jitter in the waveform because of the realization structure)*

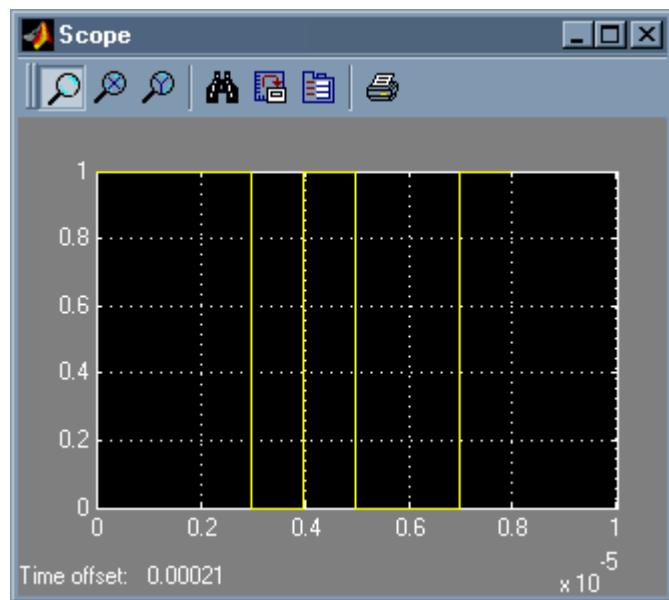


Figure B.18 Output after the FixPt Gateway Out Block

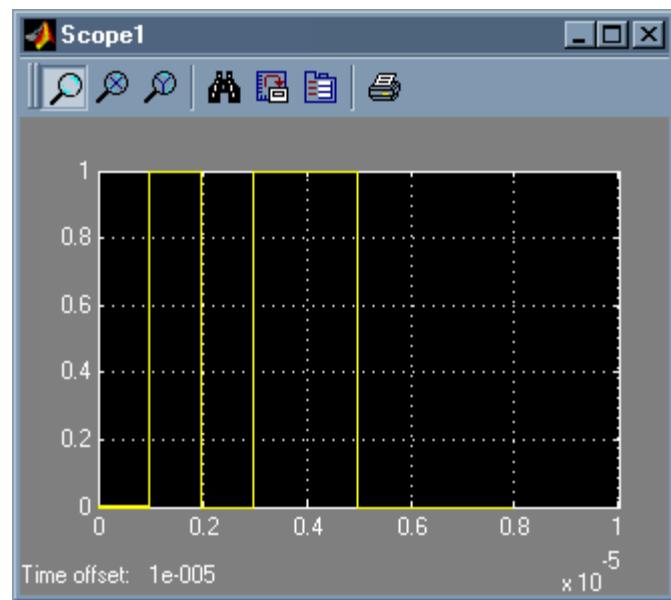


Figure B.19 Output after the Bernoulli Random Binary Generator Block

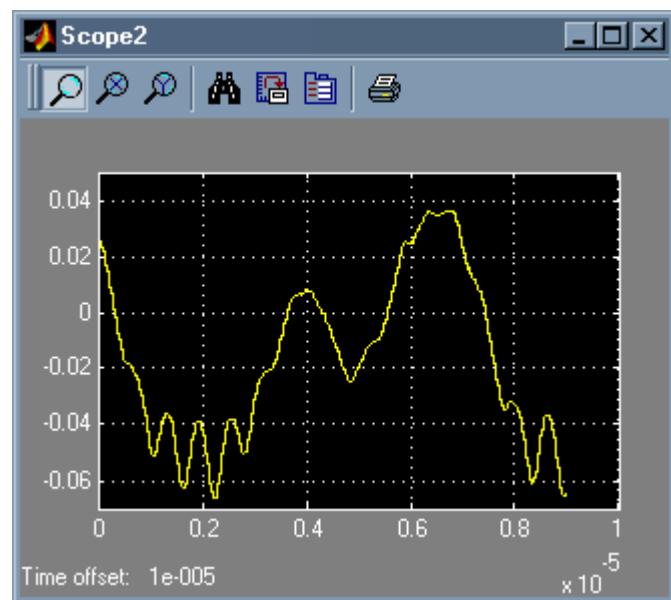


Figure B.20 Output after the Adder2 Block

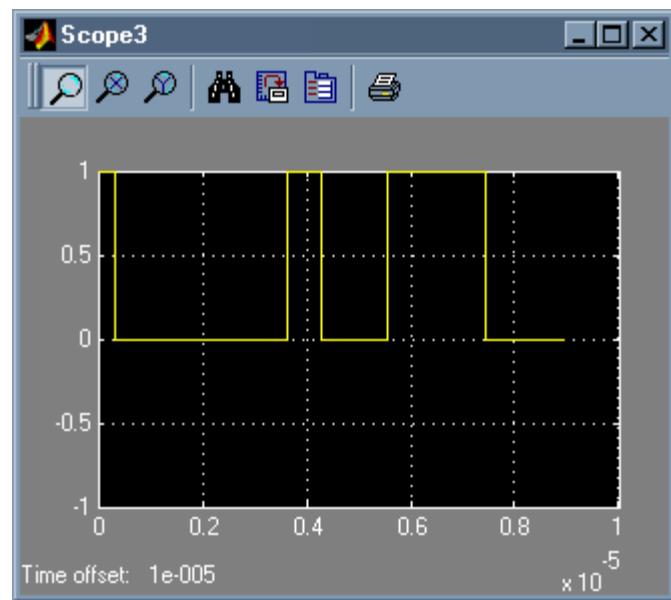


Figure B.21 Output after the Relational Block

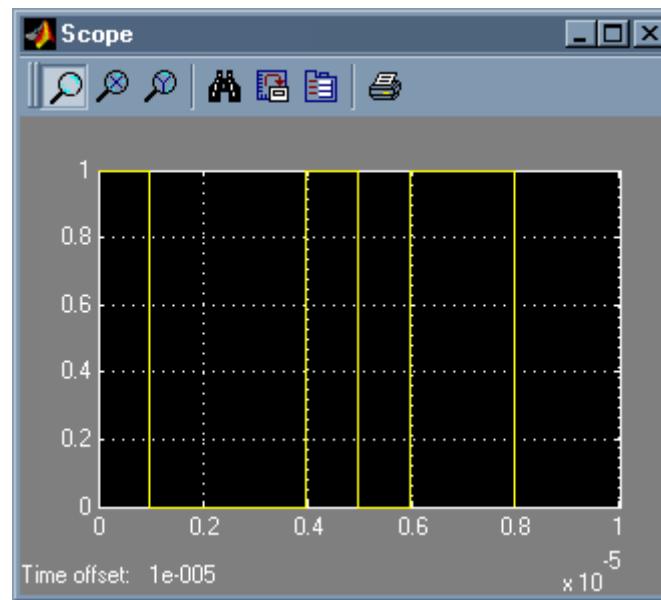
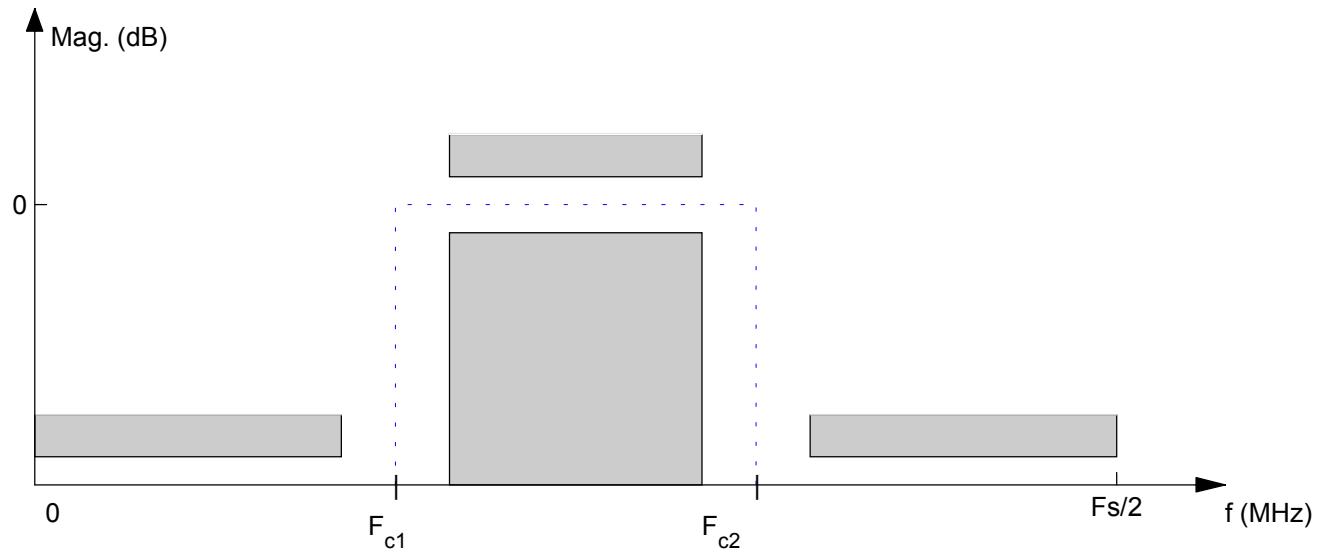


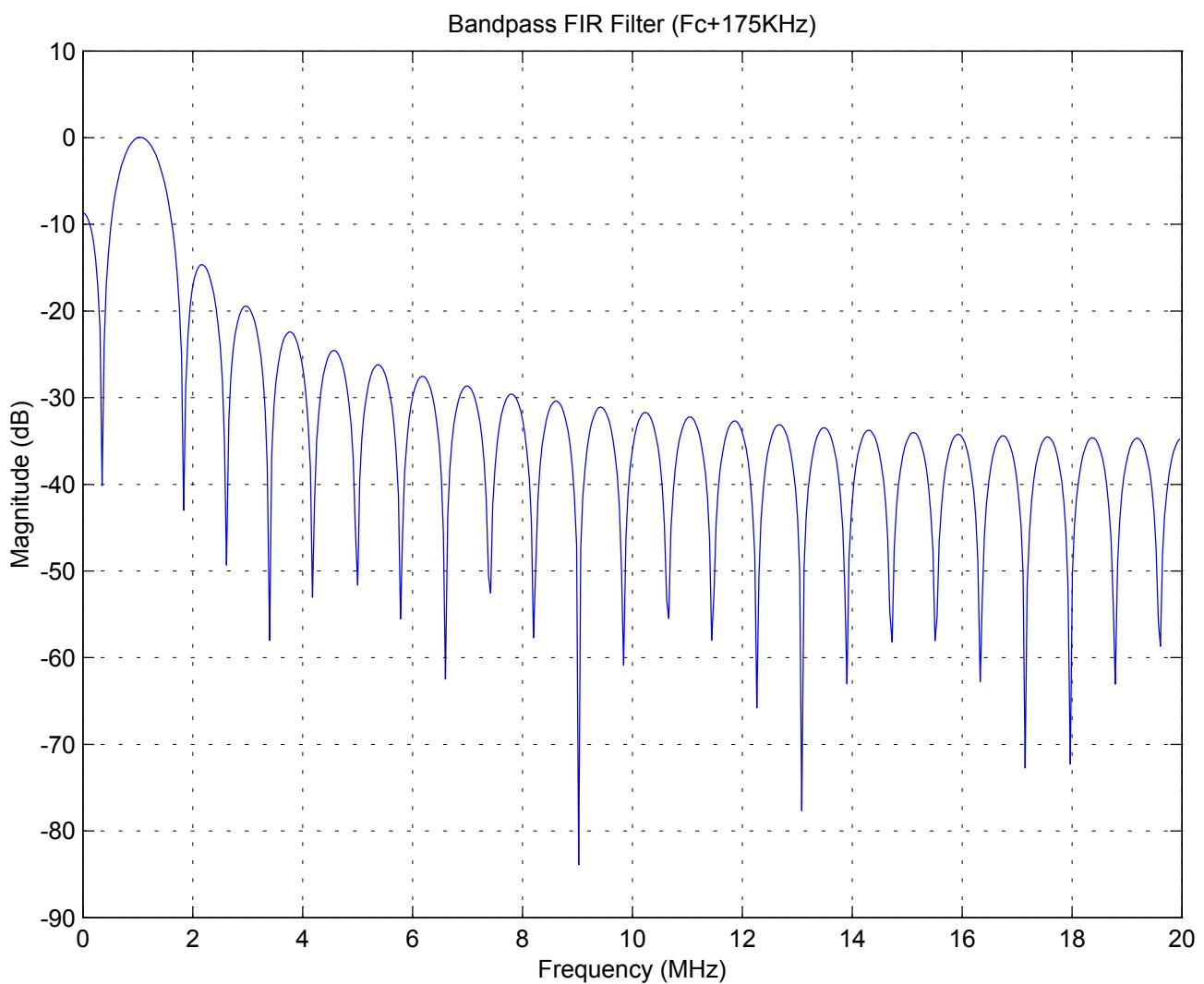
Figure B.22 Output after the Gateway Out Block

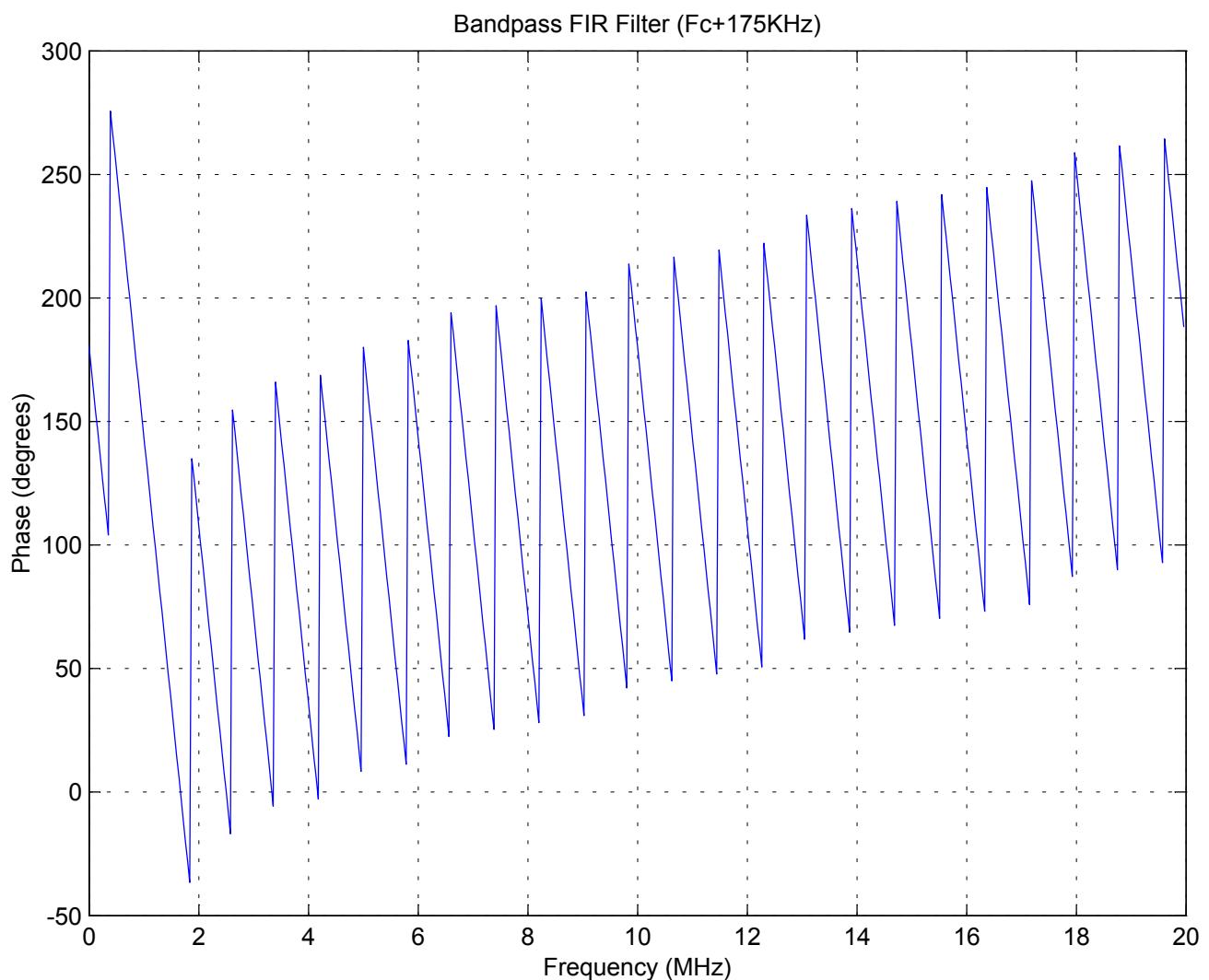
APPENDIX-C

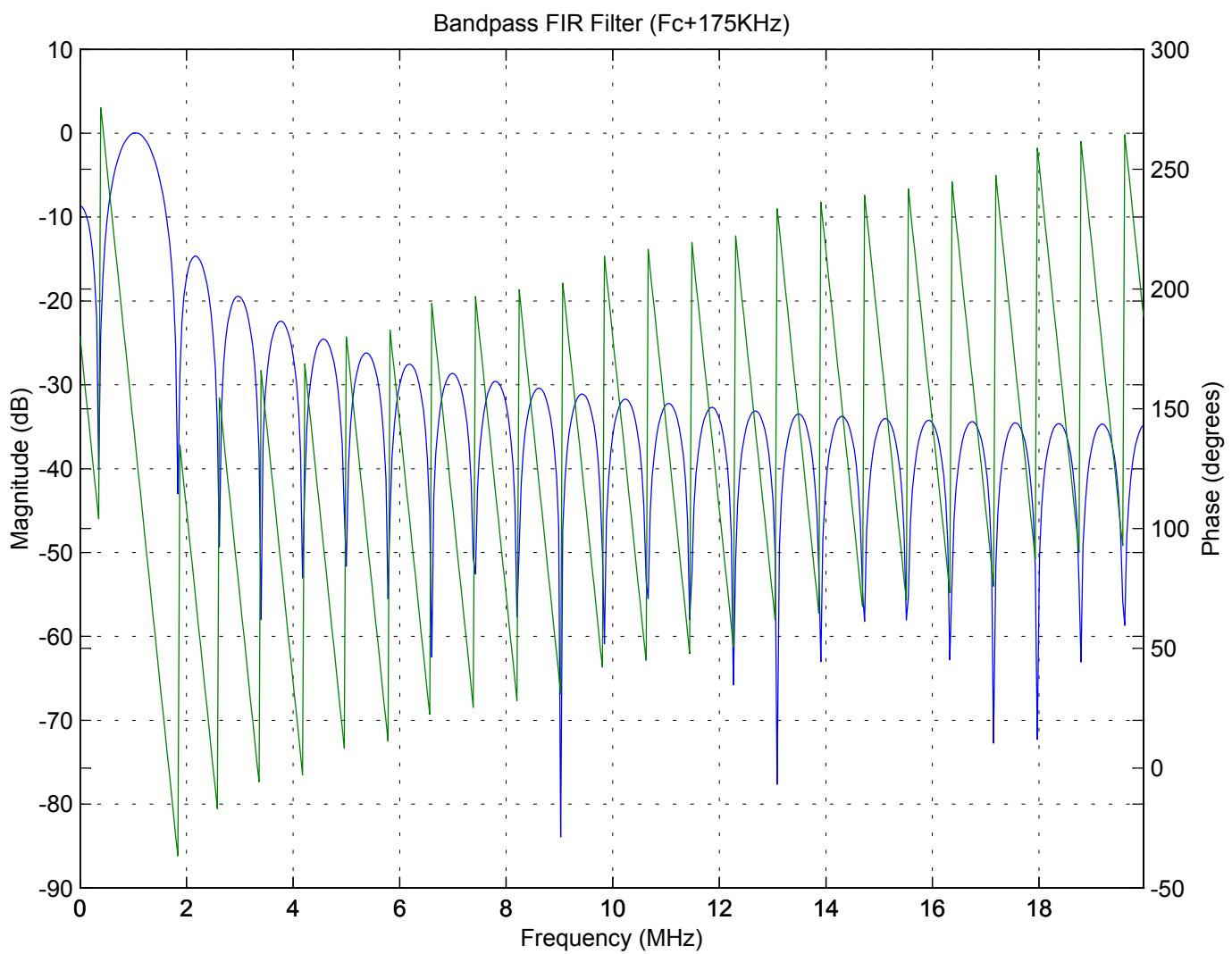
DIGITAL FILTER DESIGNS

Bandpass FIR Filter ($F_c+175\text{KHz}$)

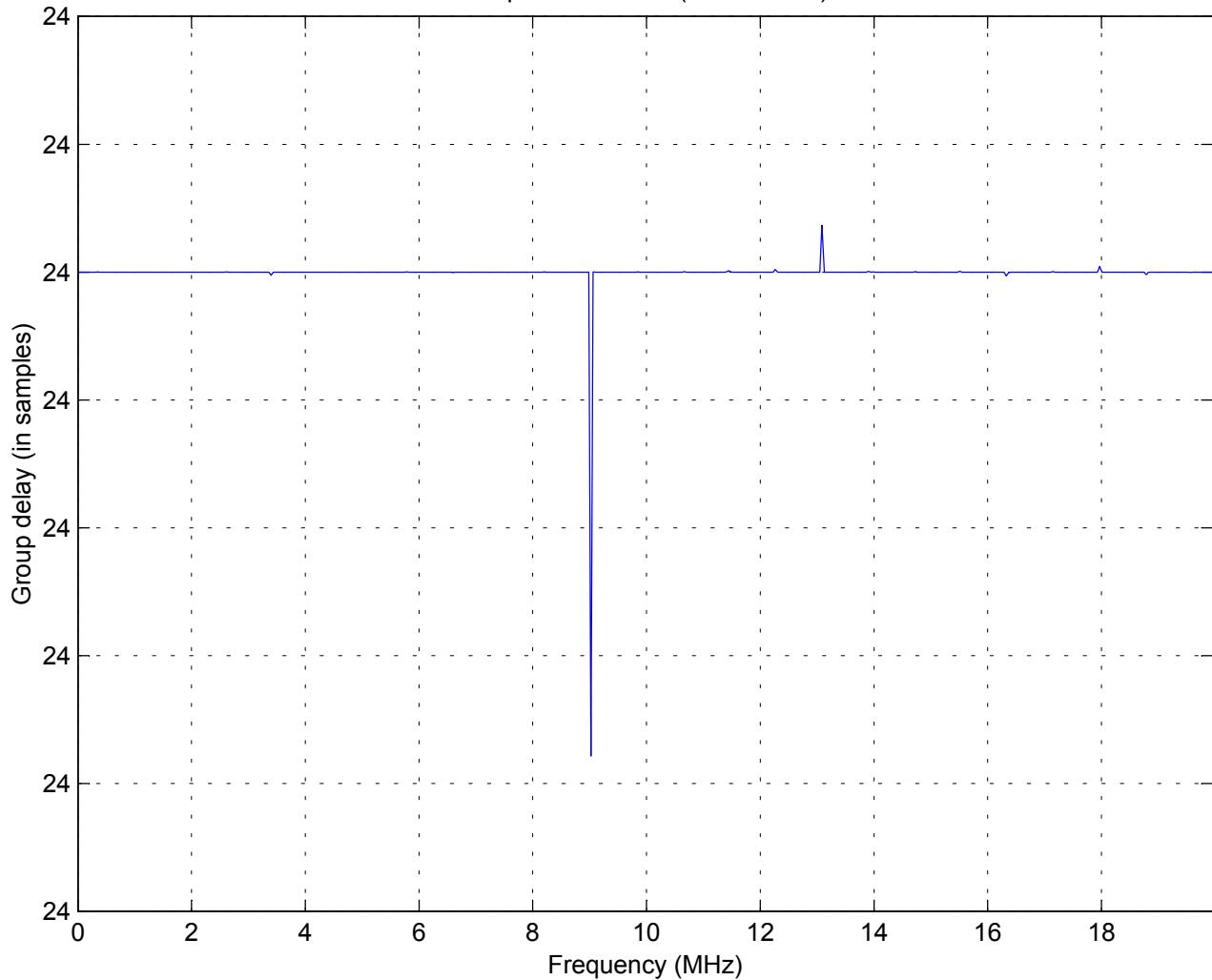


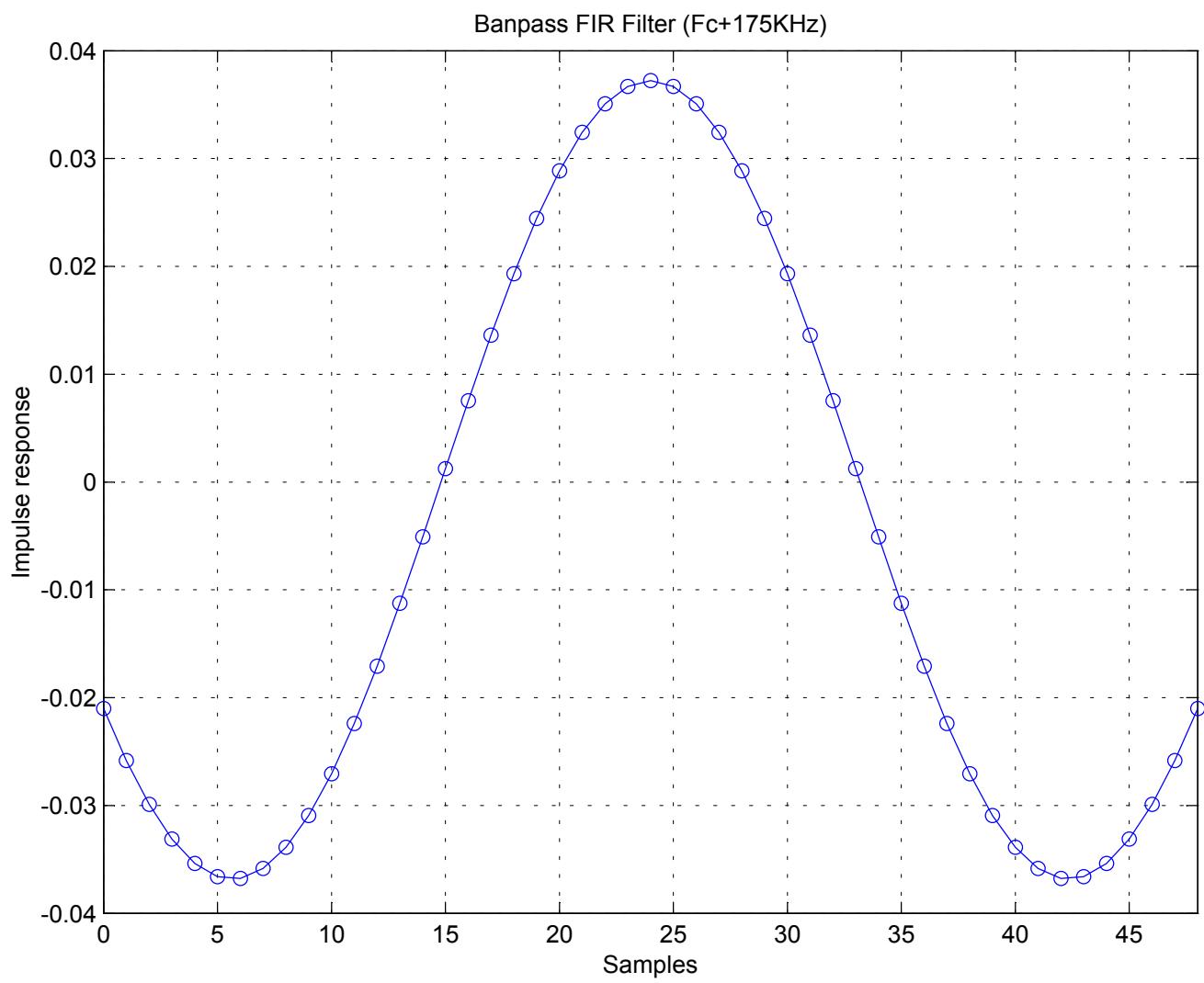




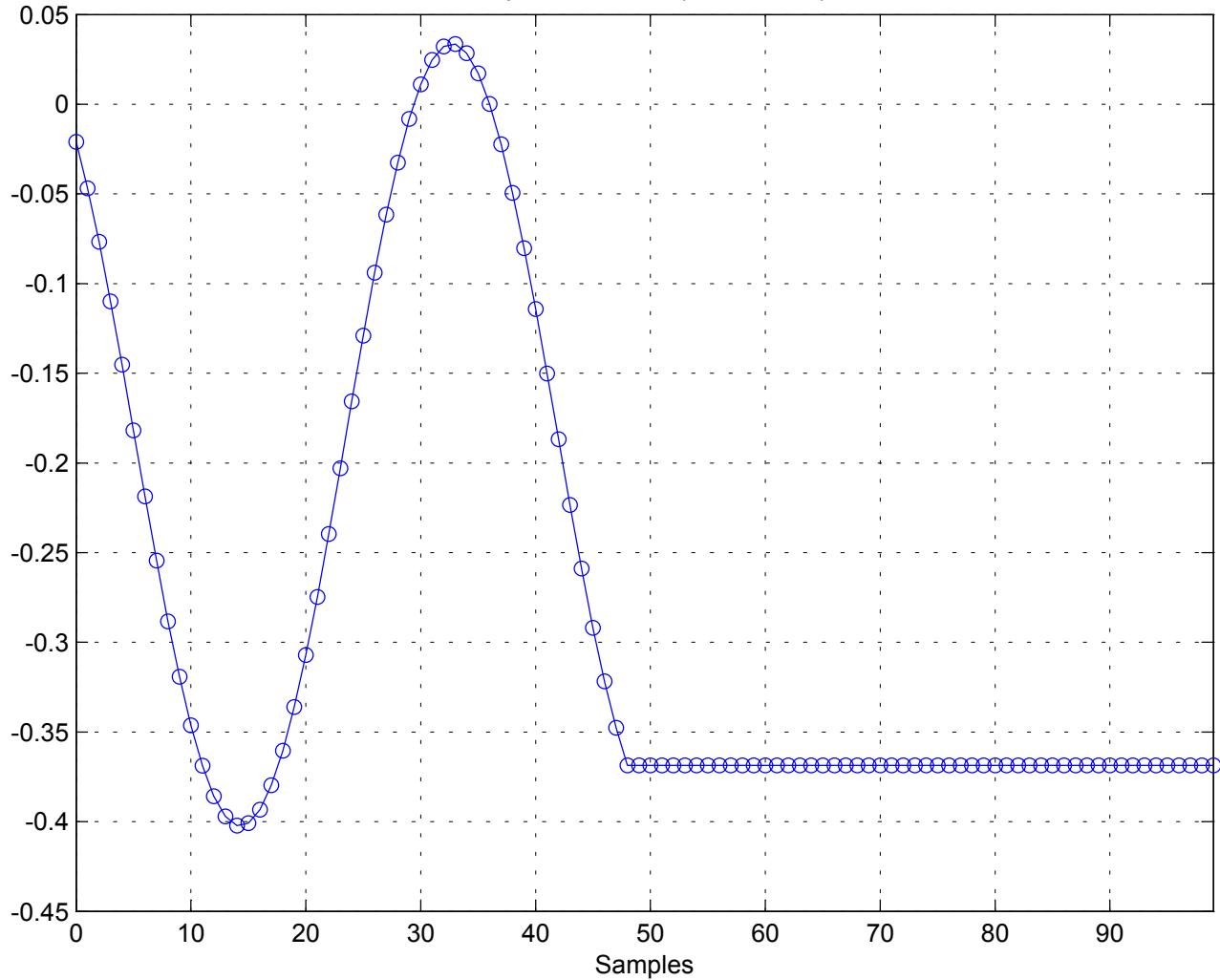


Banpass FIR Filter ($F_c+175\text{KHz}$)

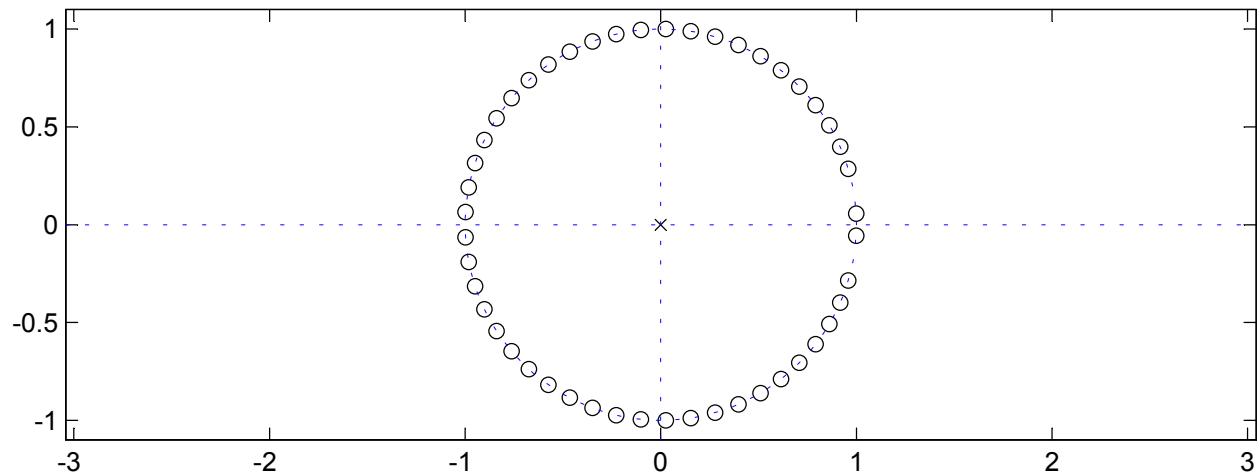




Banpass FIR Filter (Fc+175KHz)



Banpass FIR Filter ($F_c+175\text{KHz}$)



Banpass FIR Filter (Fc+175KHz) Coefficients

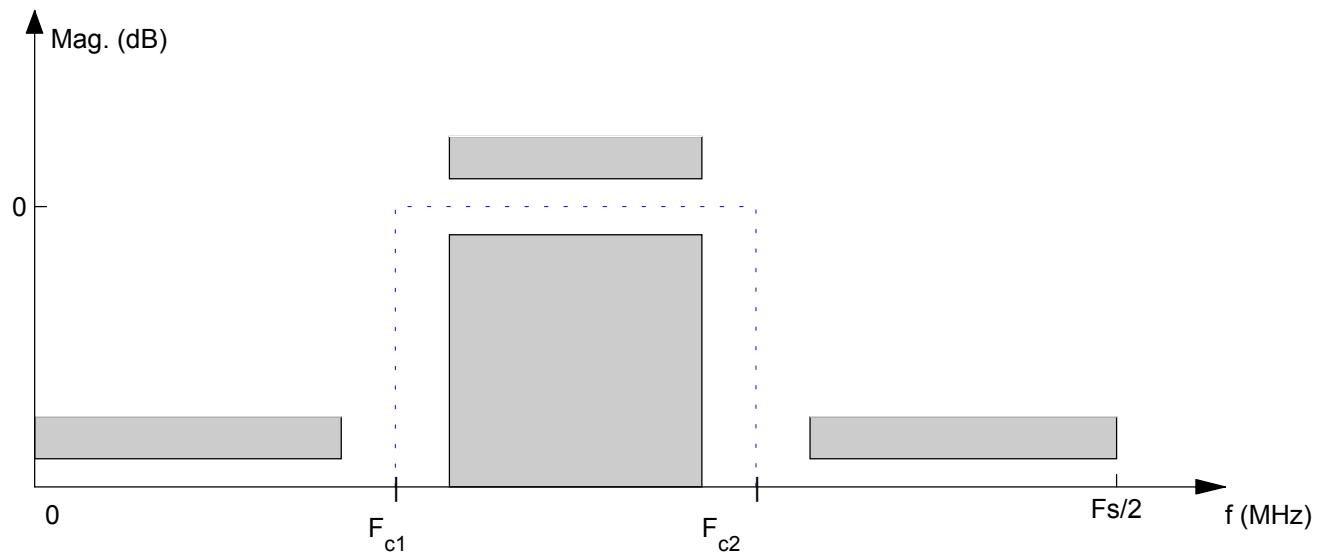
Numerator:

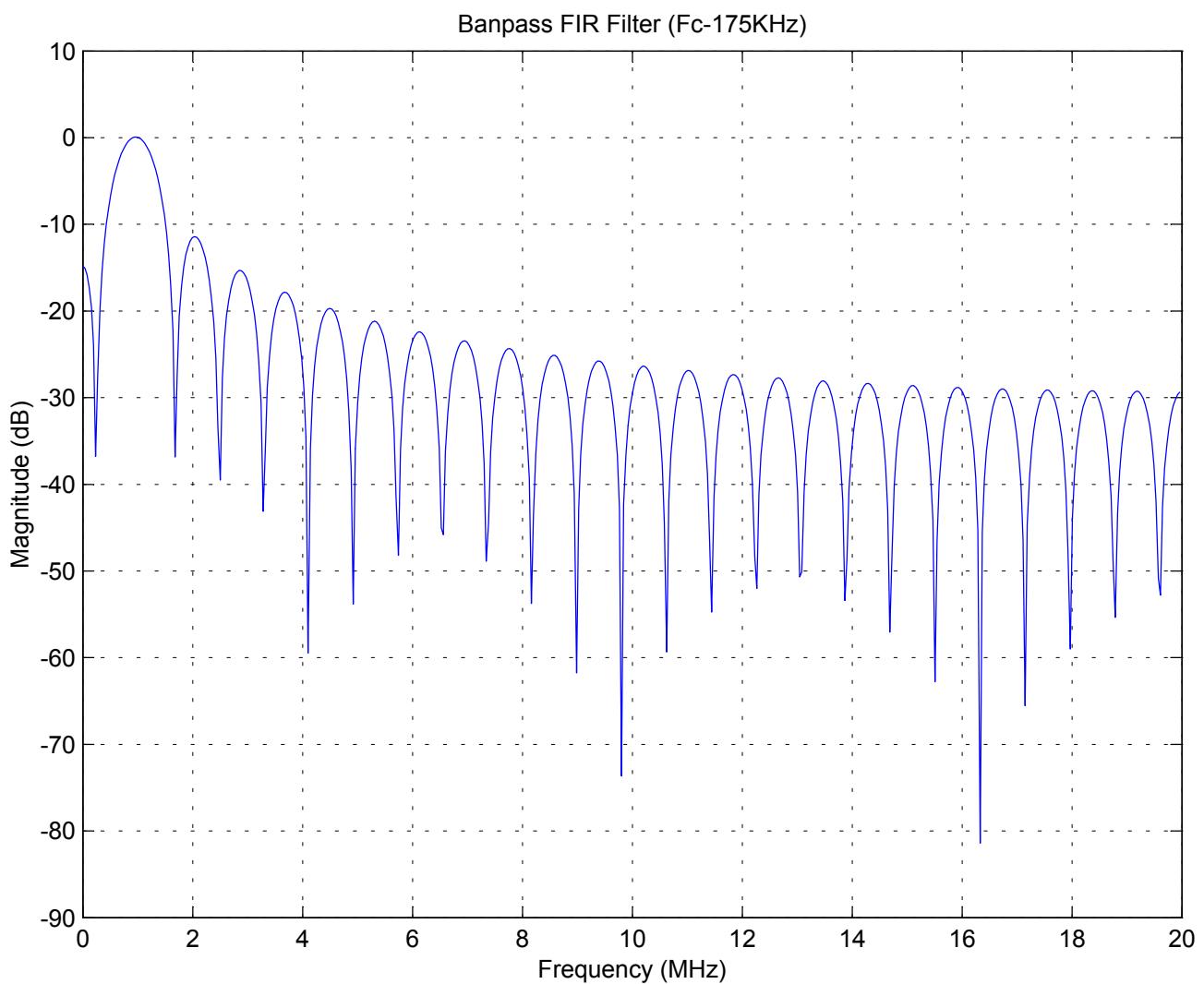
-0.021010546482473
-0.025826494386943
-0.029902293829211
-0.033116131744129
-0.035371259235019
-0.036598925828189
-0.036760488957429
-0.035848633556422
-0.033887663161267
-0.030932851674557
-0.027068873080600
-0.022407354078926
-0.017083620982623
-0.011252736509903
-0.005084943543988
+0.001239349104595
+0.007534889975890
+0.013617058859811
+0.019307328190382
+0.024438549002062
+0.028859903609829
+0.032441377020734
+0.035077613389060
+0.036691042161149
+0.037234180368410
+0.036691042161149
+0.035077613389060
+0.032441377020734
+0.028859903609829
+0.024438549002062
+0.019307328190382
+0.013617058859811
+0.007534889975890
+0.001239349104595
-0.005084943543988
-0.011252736509903
-0.017083620982623
-0.022407354078926
-0.027068873080600
-0.030932851674557
-0.033887663161267
-0.035848633556422
-0.036760488957429
-0.036598925828189
-0.035371259235019
-0.033116131744129
-0.029902293829211

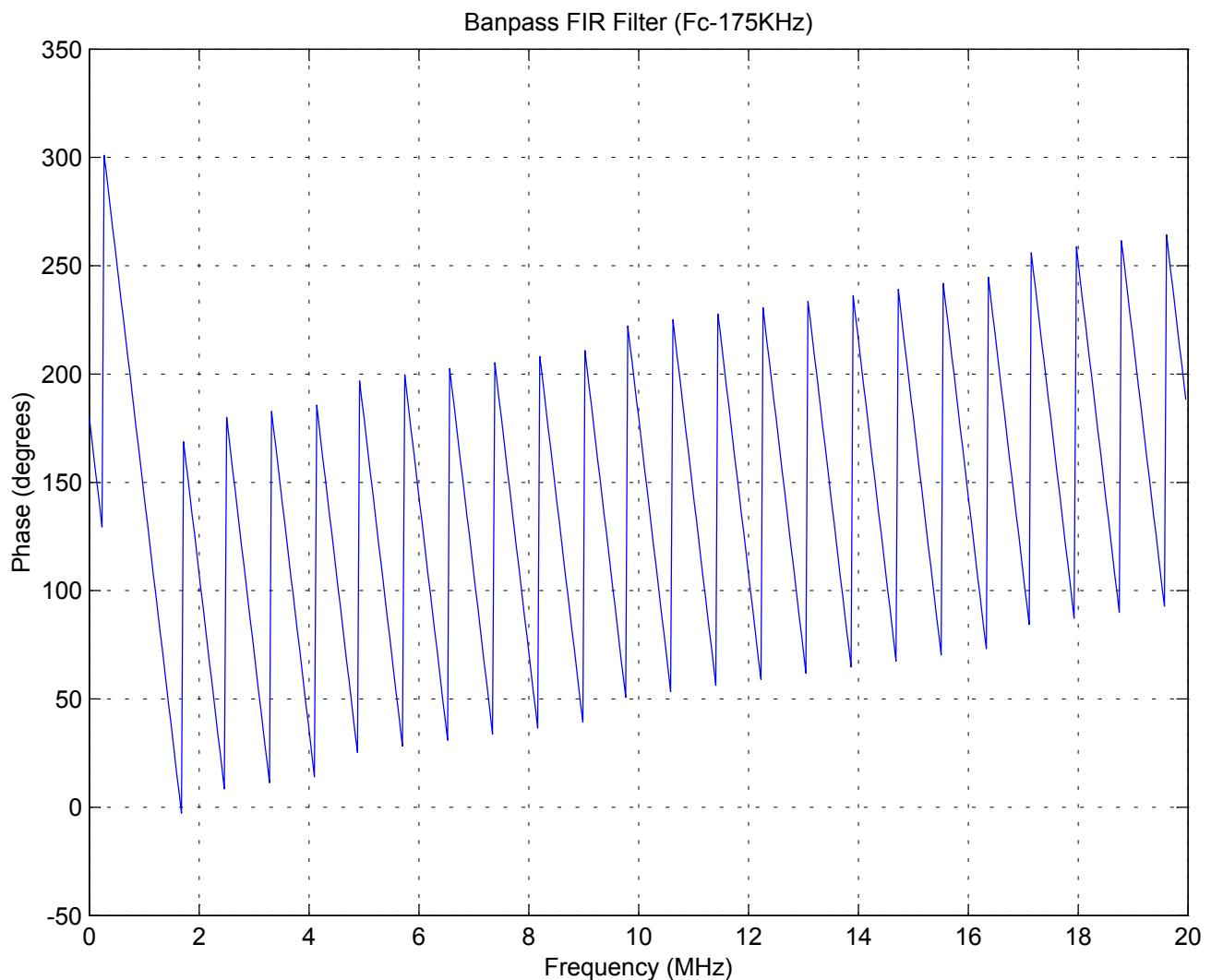
-0.025826494386943
-0.021010546482473

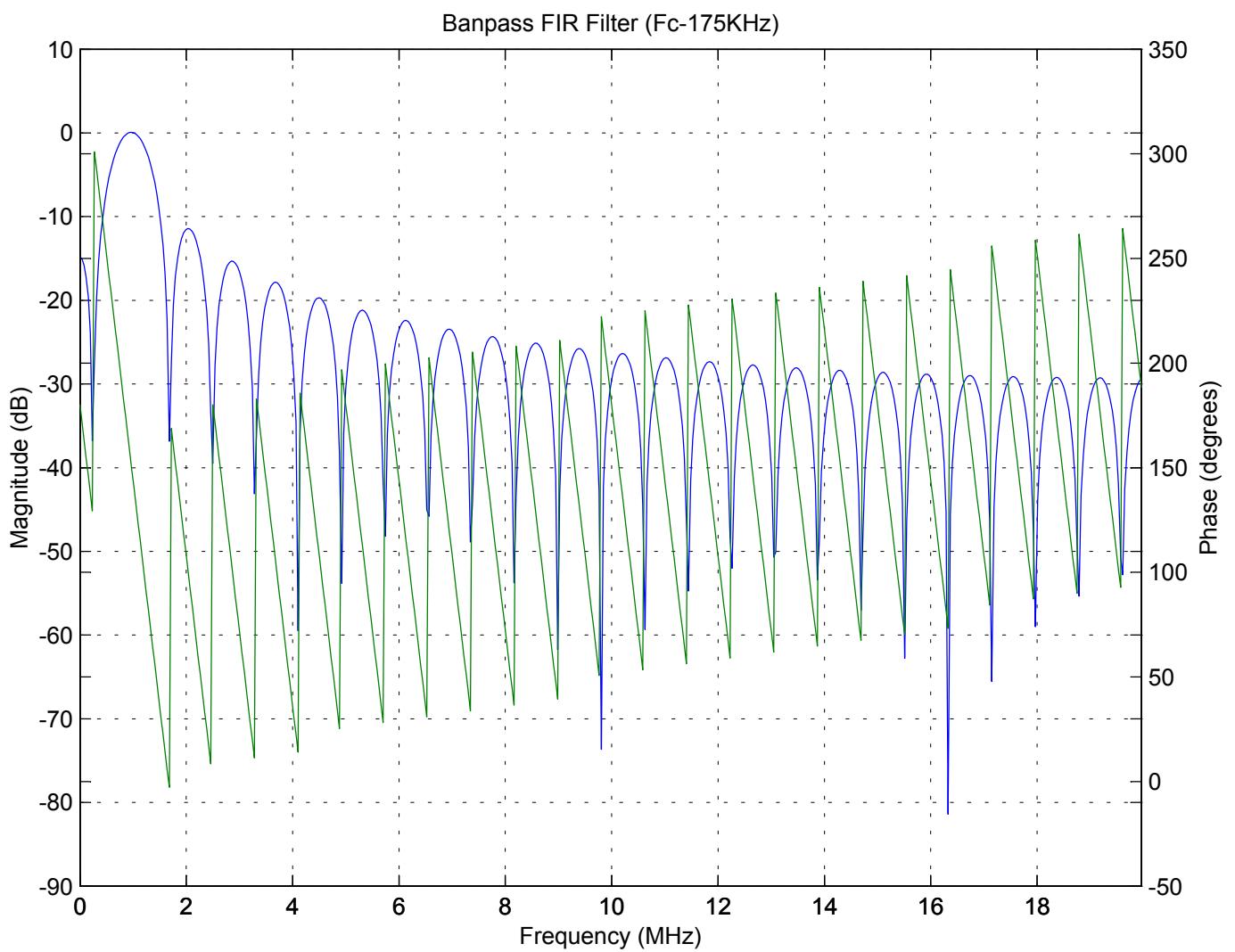
Denominator:
1.000000000000000

Bandpass FIR Filter (F_c -175KHz)

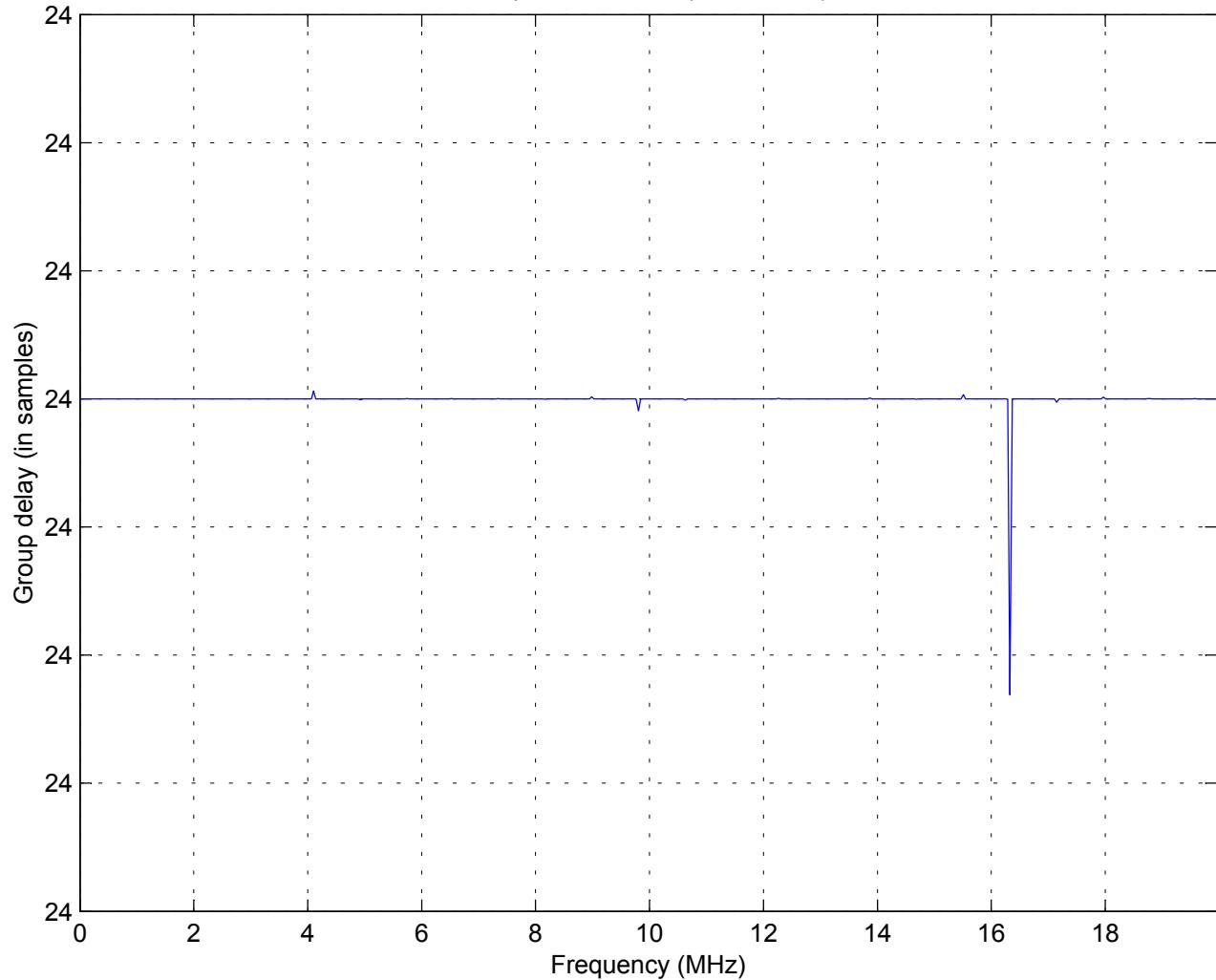


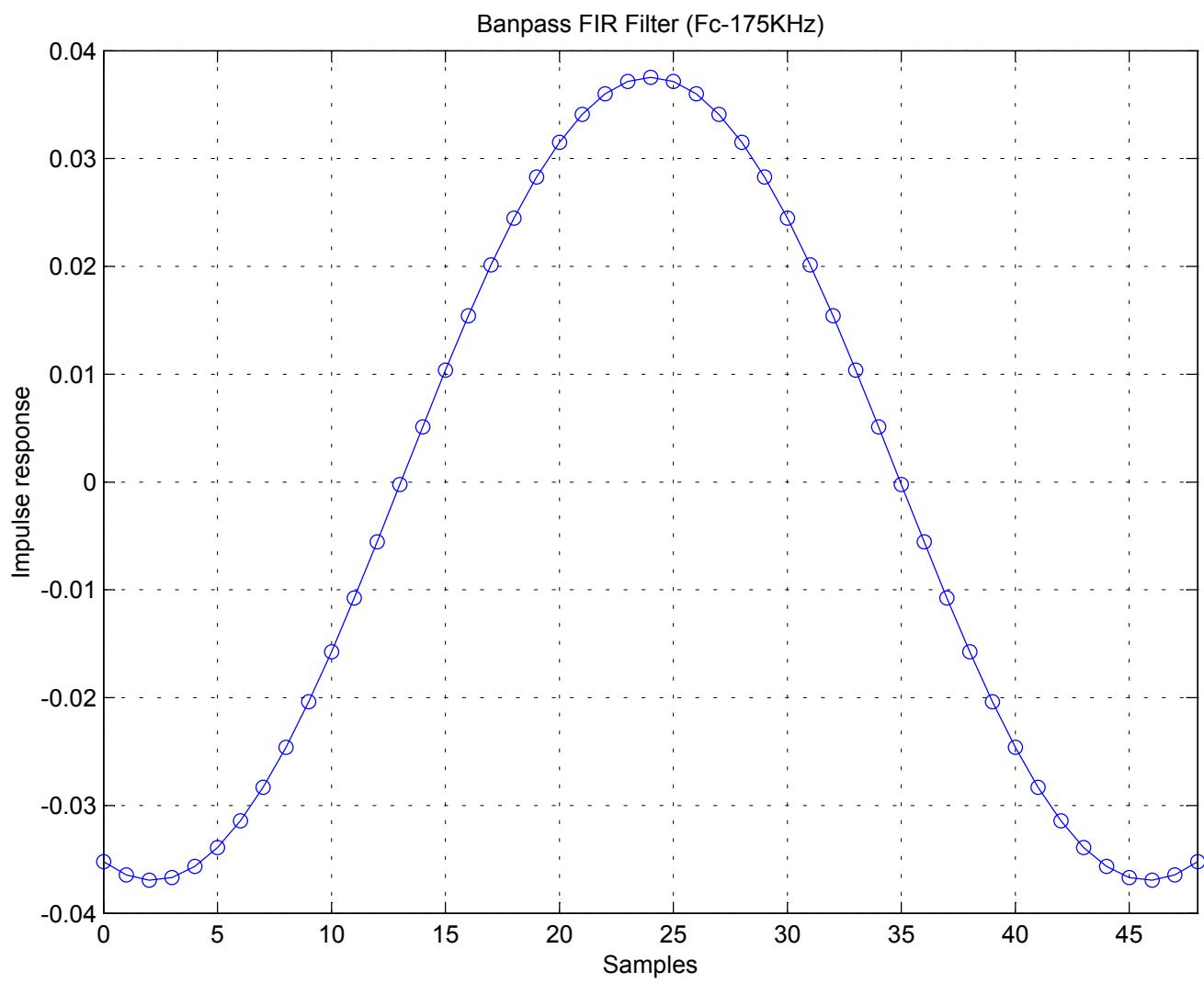




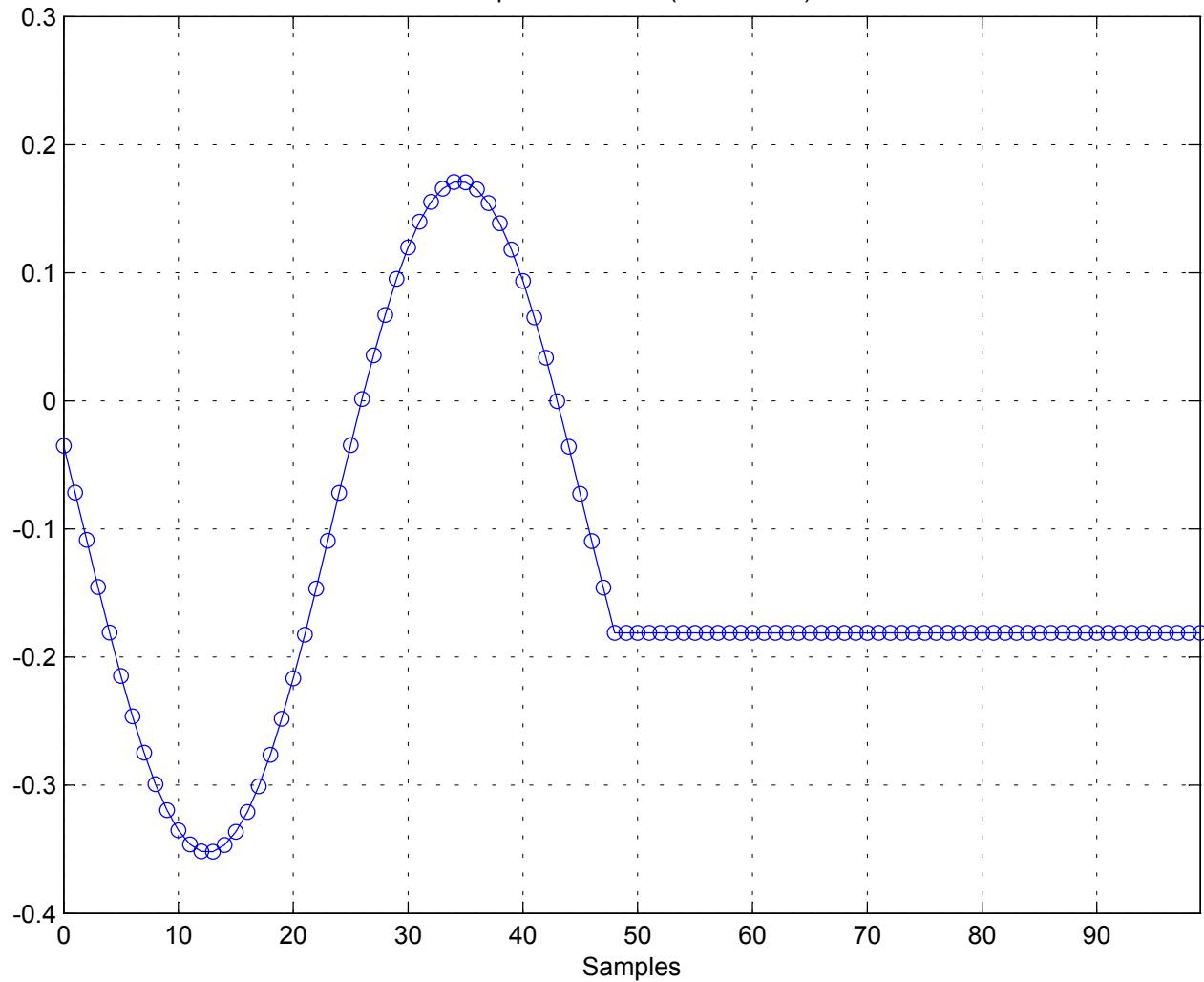


Banpass FIR Filter (Fc-175KHz)

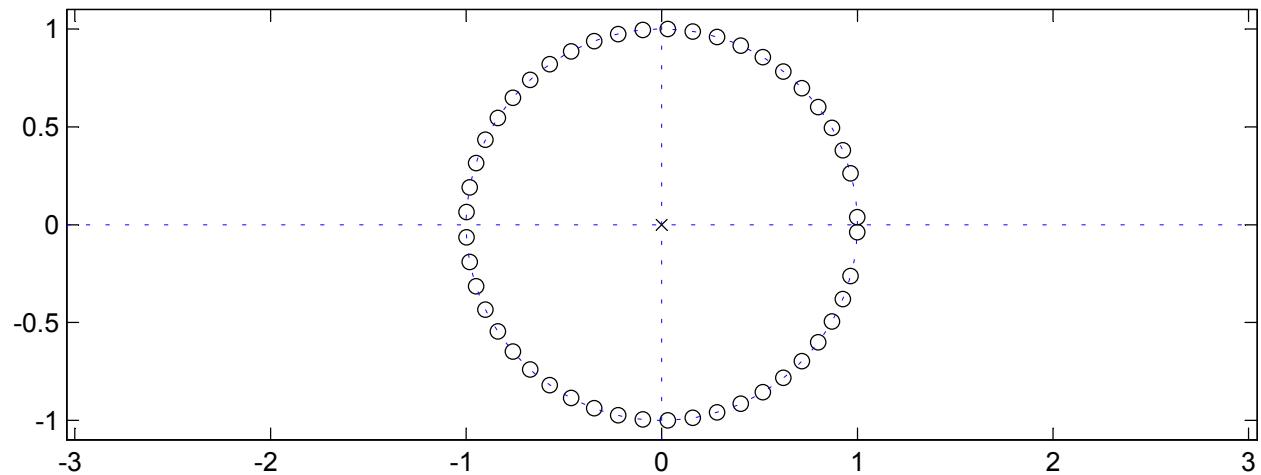




Banpass FIR Filter (Fc-175KHz)



Banpass FIR Filter (Fc-175KHz)



Banpass FIR Filter (Fc-175KHz) Coefficients

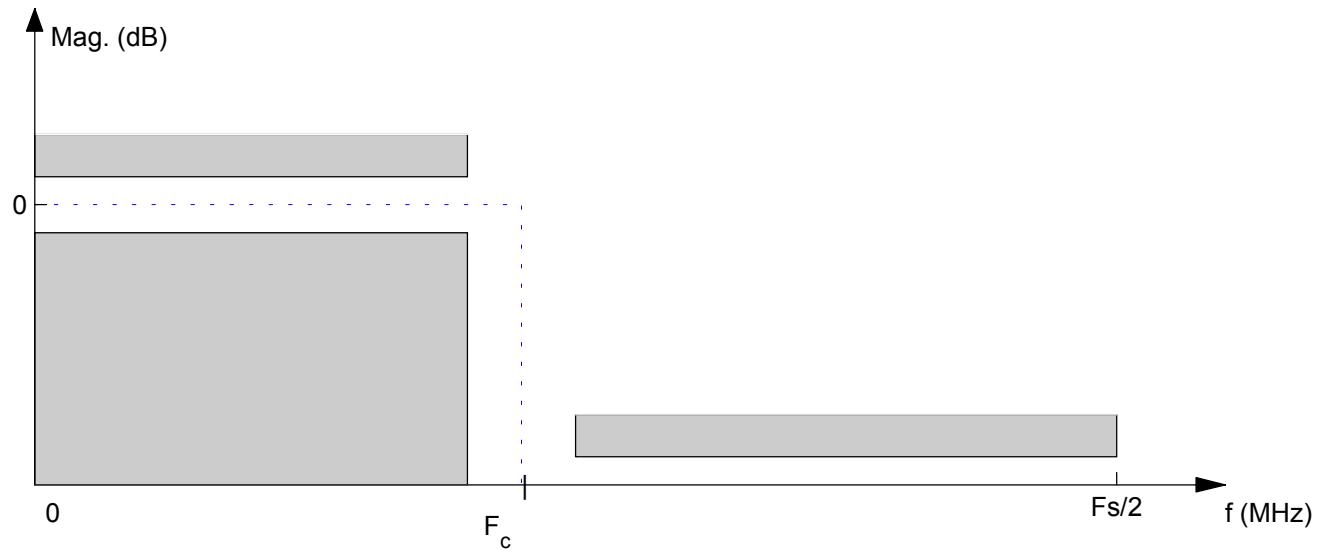
Numerator:

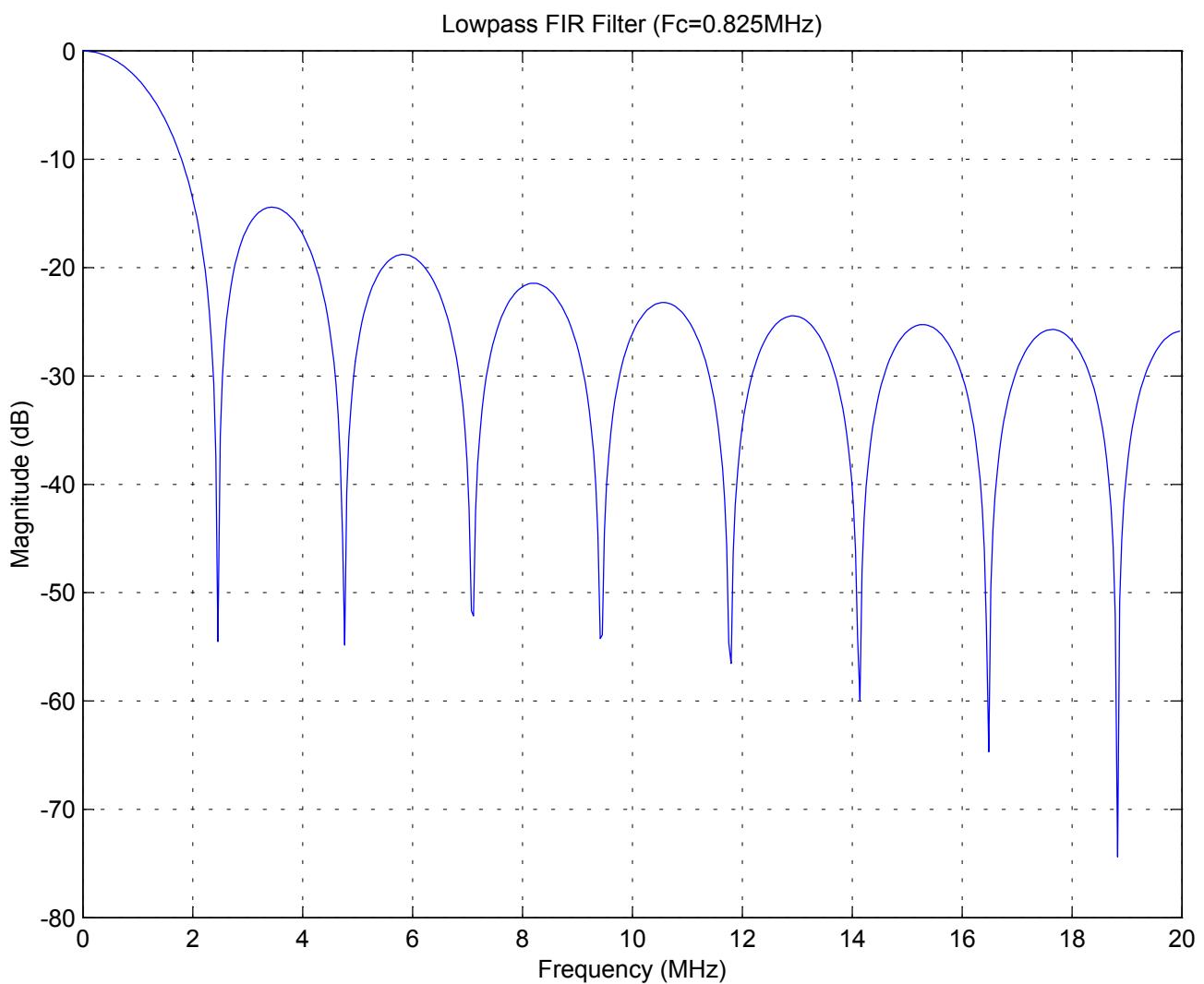
-0.035204205764639
-0.036446359985027
-0.036941147287746
-0.036676109163124
-0.035654609367599
-0.033895806788862
-0.031434294555958
-0.028319412218227
-0.024614244820661
-0.020394329425854
-0.015746095928389
-0.010765074739325
-0.005553908961575
-0.000220212921059
+0.005125677729270
+0.010373016729035
+0.015412969782573
+0.020140899694514
+0.024458565519616
+0.028276188960688
+0.031514344037111
+0.034105629781590
+0.035996090324467
+0.037146352094480
+0.037532453886856
+0.037146352094480
+0.035996090324467
+0.034105629781590
+0.031514344037111
+0.028276188960688
+0.024458565519616
+0.020140899694514
+0.015412969782573
+0.010373016729035
+0.005125677729270
-0.000220212921059
-0.005553908961575
-0.010765074739325
-0.015746095928389
-0.020394329425854
-0.024614244820661
-0.028319412218227
-0.031434294555958
-0.033895806788862
-0.035654609367599
-0.036676109163124
-0.036941147287746

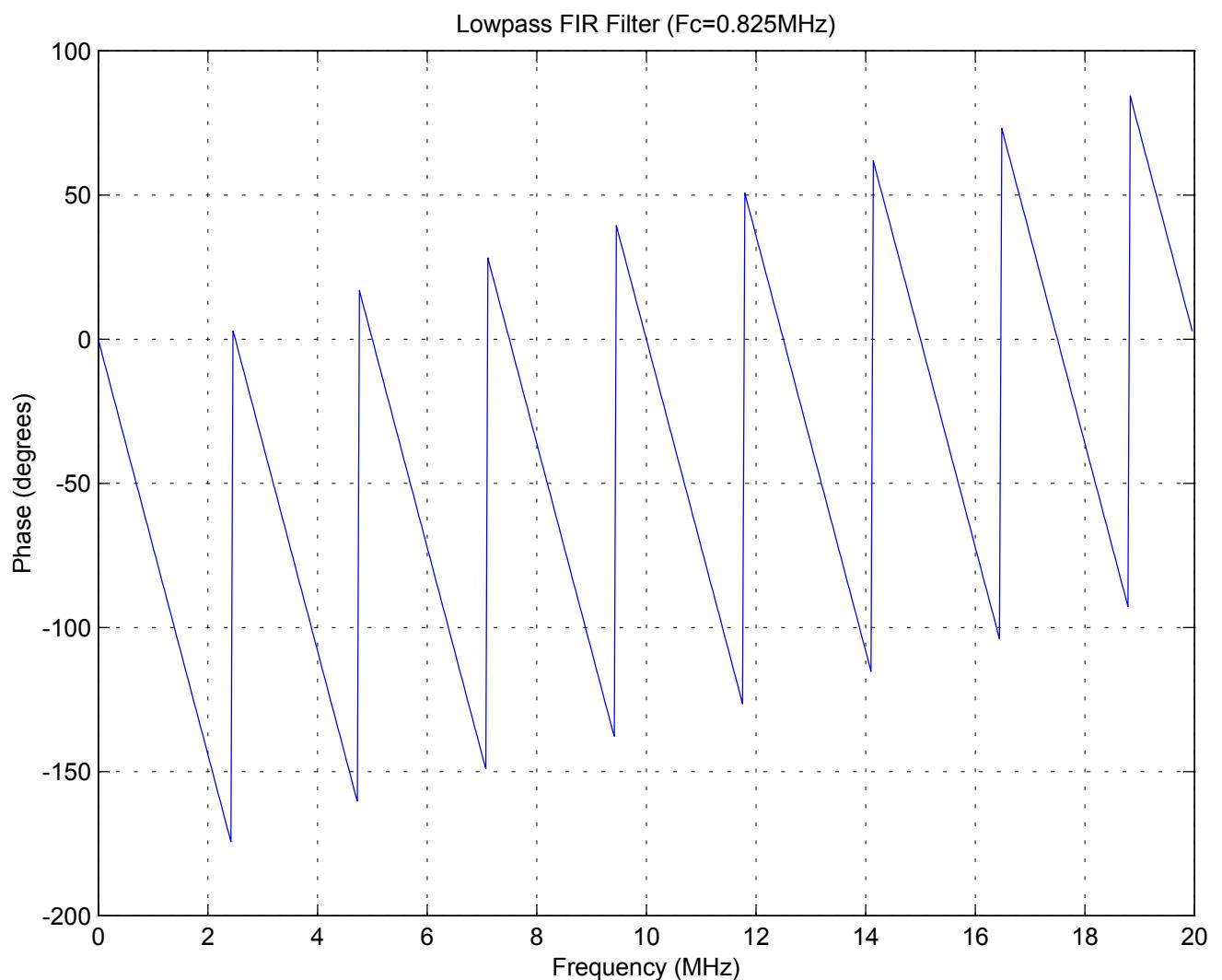
-0.036446359985027
-0.035204205764639

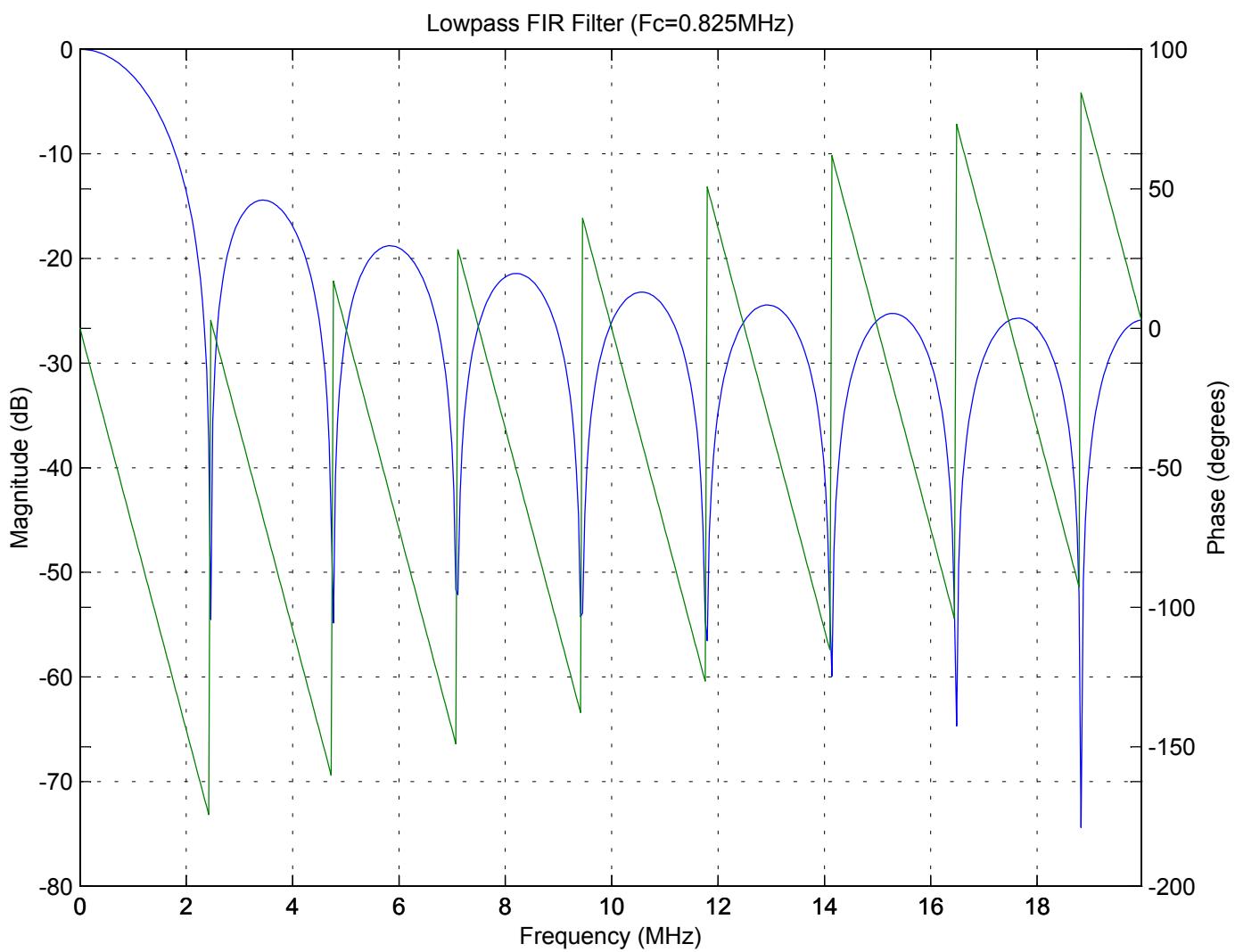
Denominator:
1.000000000000000

Lowpass FIR Filter ($F_c=0.825\text{MHz}$)

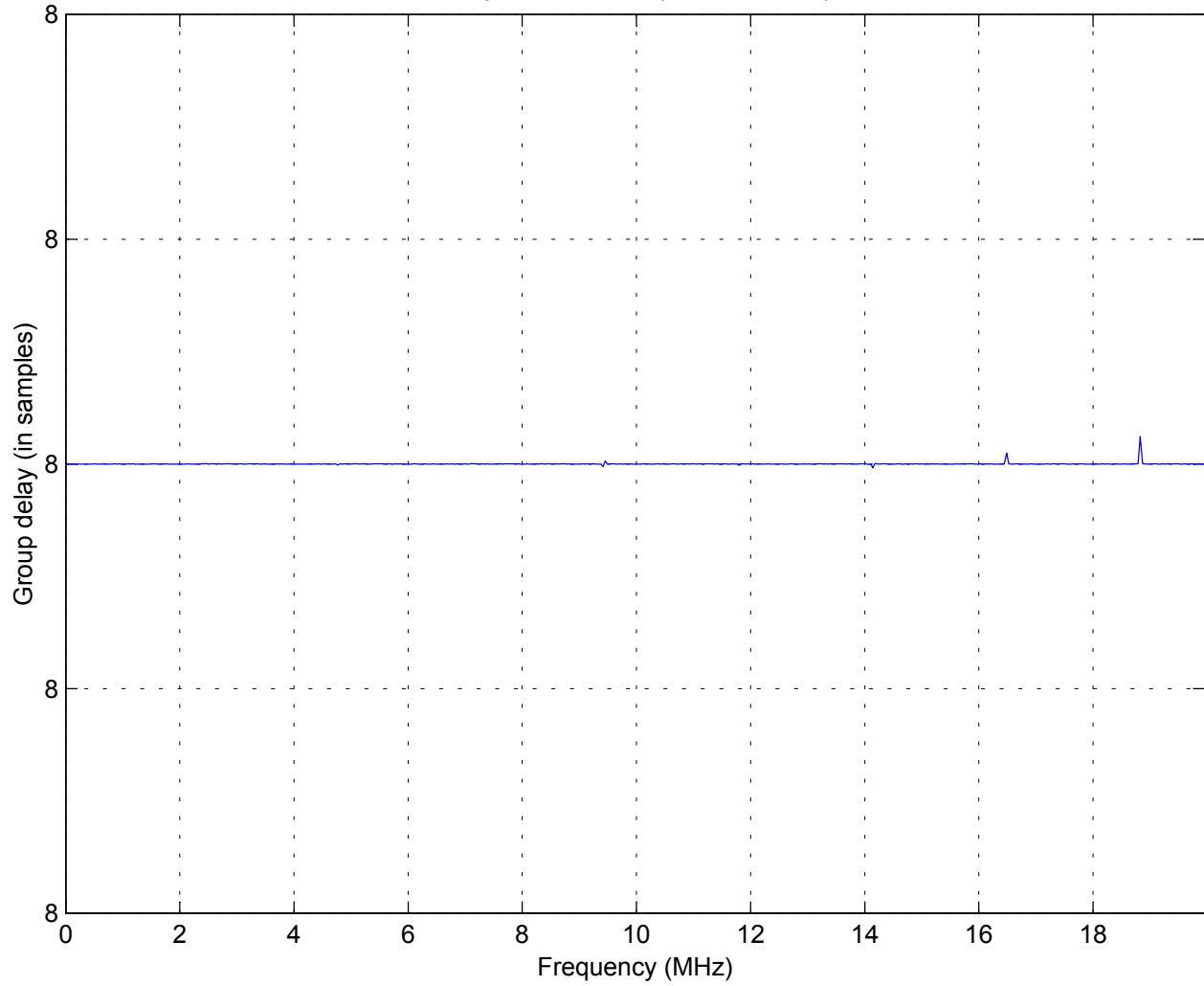


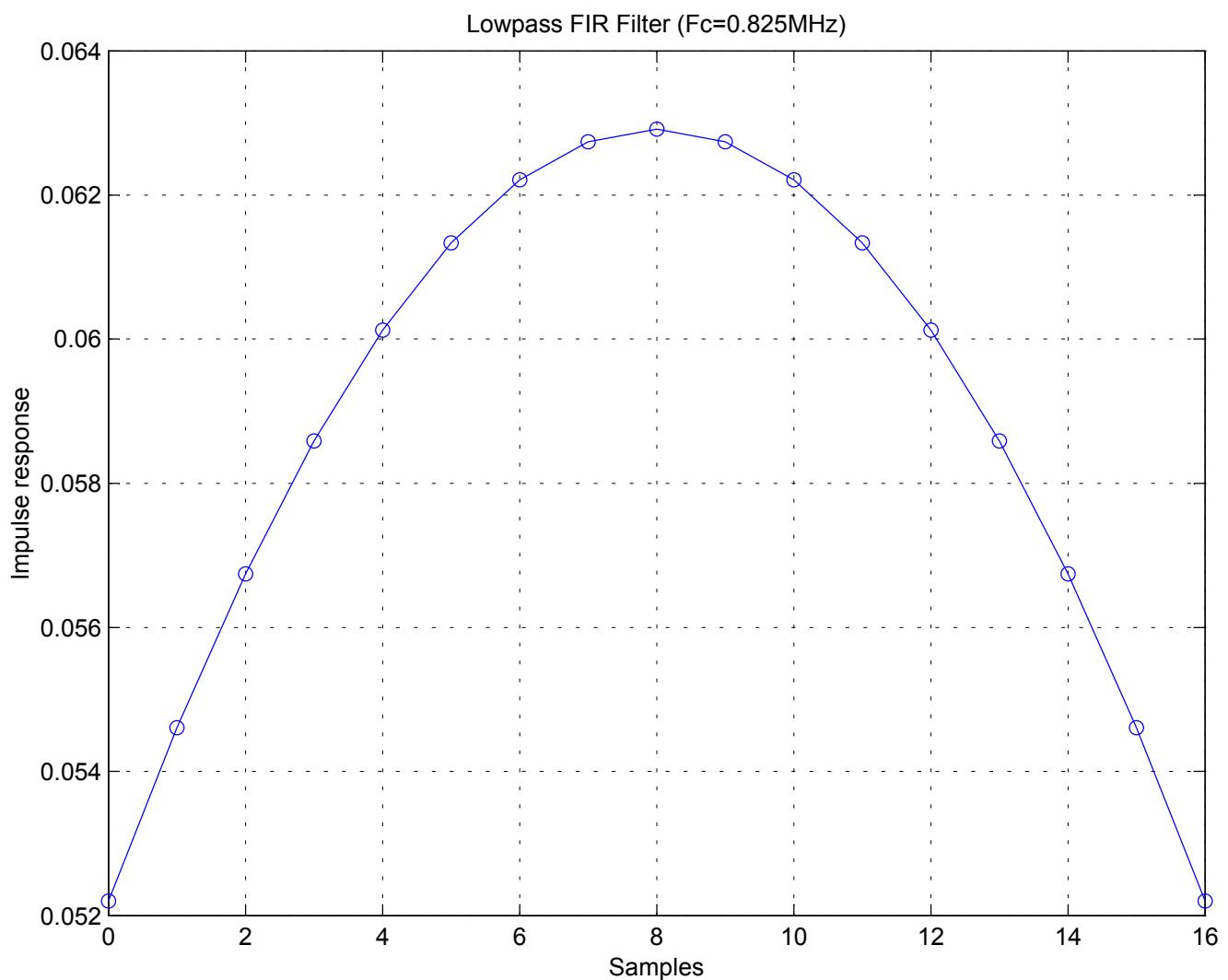




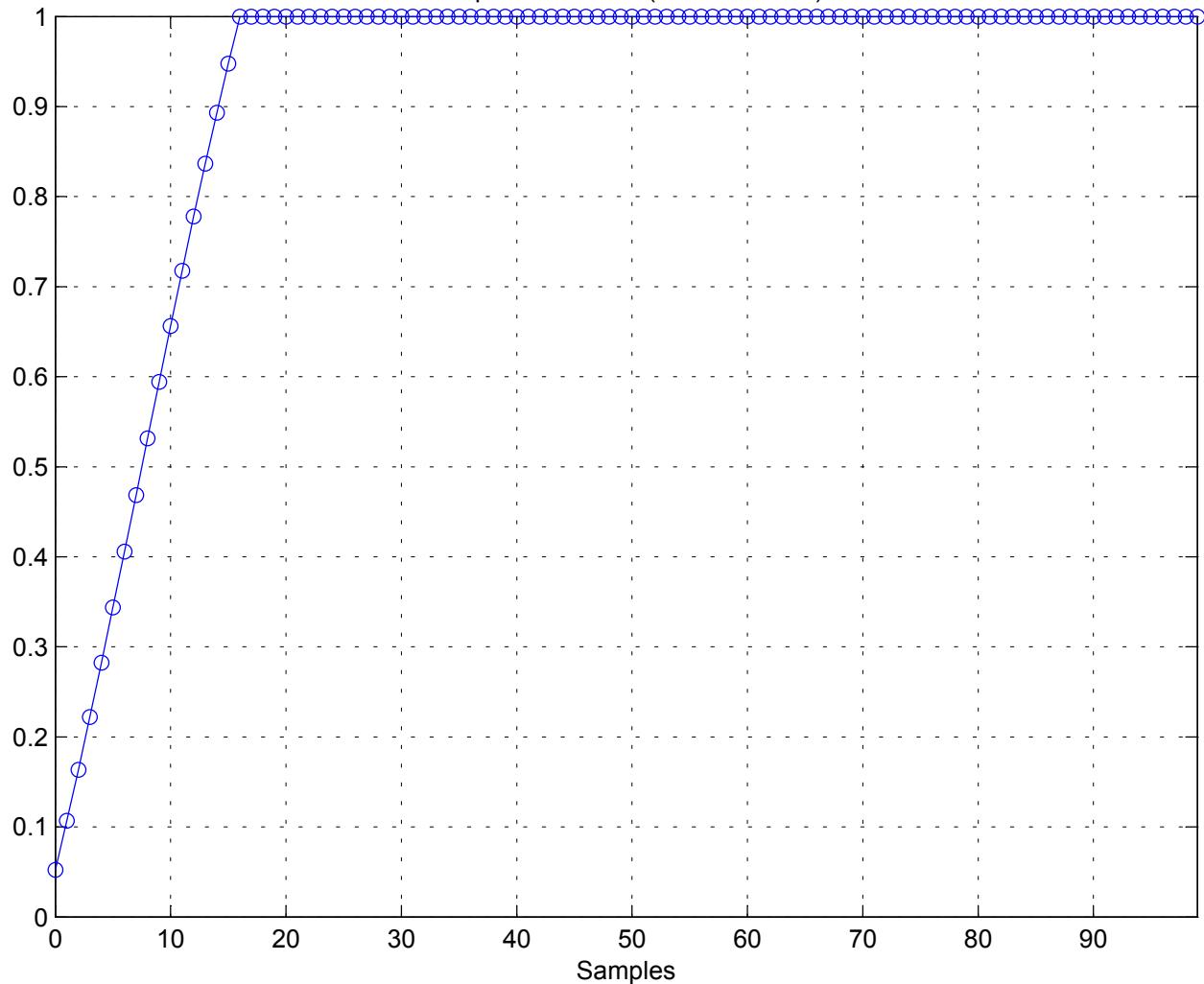


Lowpass FIR Filter ($F_c=0.825\text{MHz}$)

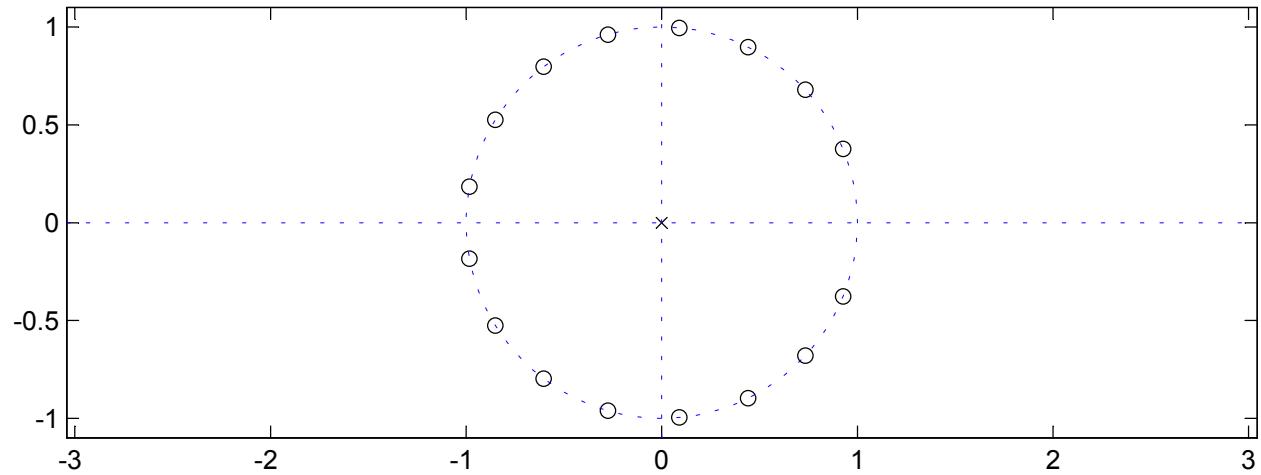




Lowpass FIR Filter ($F_c=0.825\text{MHz}$)



Lowpass FIR Filter ($F_c=0.825\text{MHz}$)



Lowpass FIR Filter (Fc=0.825MHz) Coefficients**Numerator:**

0.052201125324047
0.054606764546536
0.056742892204744
0.058588146486140
0.060123995734790
0.061334961963488
0.062208807908864
0.062736685153841
0.062913241355097
0.062736685153841
0.062208807908864
0.061334961963488
0.060123995734790
0.058588146486140
0.056742892204744
0.054606764546536
0.052201125324047

Denominator:

1.000000000000000

GAUSSIAN LOWPASS PASS FILTER DESIGN

FREQUENCY	SCALE VALUE ON THE GAUSSIAN CURVE	NORMALIZED FREQUENCY
147.0588K	2.5	0.968
294.1176K	5.0	0.884
441.1764K	7.5	0.750
500.0000K	8.5	0.700
588.2350K	10.0	0.600
735.2940K	12.5	0.460
882.3530K	15.0	0.328
1029.4110K	17.0	0.220
1176.4700K	20.0	0.136
1323.5290K	22.5	0.080
1470.5880K	25.0	0.044
1617.6468K	27.5	0.024

APPENDIX-D

**GENERATED VHDL CODE & DATA SHEETS OF THE
XILINX IP CORES**

The file **vhdlFiles** gives a list of the VHDL code that is needed for the System Generator project. This list contains both the generated files, as well as Xilinx Blockset (XBS) library files (stored in the default location **\$MATLAB/toolbox/xilinx/sysgen/vhdl**). The list of files is presented in the order of dependencies. The file in the list is necessary for the other files to simulate or synthesize. When creating a synthesis project file, the VHDL files have to be entered into the project in the order mentioned in the vhdlFiles [48].

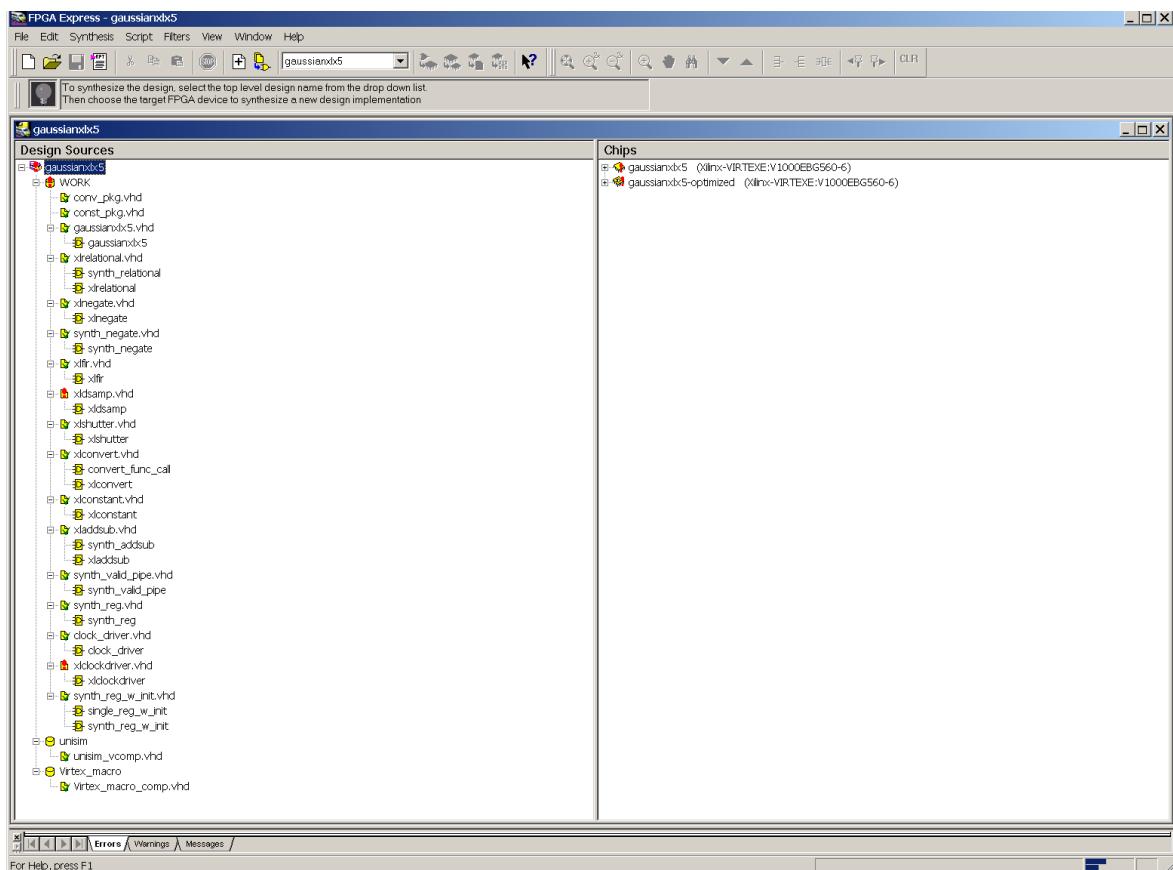
The **vhdlFiles** file for this project is listed below:

```
const_pkg.vhd
C:/matlabR12/toolbox/xilinx/sysgen/vhdl/conv_pkg.vhd
C:/matlabR12/toolbox/xilinx/sysgen/vhdl/synth_reg.vhd
C:/matlabR12/toolbox/xilinx/sysgen/vhdl/synth_reg_w_init.vhd
C:/matlabR12/toolbox/xilinx/sysgen/vhdl/synth_valid_pipe.vhd
C:/matlabR12/toolbox/xilinx/sysgen/vhdl/synth_mult.vhd
C:/matlabR12/toolbox/xilinx/sysgen/vhdl/xlshutter.vhd
C:/matlabR12/toolbox/xilinx/sysgen/vhdl/xldelay.vhd

C:/matlabR12/toolbox/xilinx/sysgen/vhdl/synth_negate.vhd
C:/matlabR12/toolbox/xilinx/sysgen/vhdl/xlconvert.vhd
C:/matlabR12/toolbox/xilinx/sysgen/vhdl/xlclockdriver.vhd
C:/matlabR12/toolbox/xilinx/sysgen/vhdl/xldsamp.vhd
C:/matlabR12/toolbox/xilinx/sysgen/vhdl/xlconstant.vhd
C:/matlabR12/toolbox/xilinx/sysgen/vhdl/xladdsub.vhd

clock_driver.vhd
xlrelational_core1.vhd
xlnegate_core1.vhd
xlfir_core1.vhd
xlfir_core2.vhd
xlfir_core3.vhd
xlfir_core4.vhd
xlfir.vhd
xlnegate.vhd
xlrelational.vhd
gaussianxlx5.vhd
```

A listing of the top file in the design hierarchy follows as it was modified to include a DLL (Delay Locked Loop) core into the design to minimize clock skew. The DLL core was generated by the Xilinx CORE Generator and inserted in the System Generator-generated file **gaussianxlx4.vhd** and saved as **gaussianxlx5.vhd**. The beginning and end of the modified sections of code are marked by: **kmv**. Figure 8.3 is reproduced again for comparison.



gaussianlx5.vhd

```

library IEEE;
use IEEE.std_logic_1164.all;
use work.conv_pkg.all;

entity gaussianlx5 is
  generic (
    gaussianlx5_Gateway_In_arith: integer := xlSigned;
    gaussianlx5_Gateway_In_bin_pt: integer := 15;
    gaussianlx5_Gateway_In_width: integer := 16;
    gaussianlx5_Gateway_Out1_arith: integer := xlUnsigned;
    gaussianlx5_Gateway_Out1_bin_pt: integer := 0;
    gaussianlx5_Gateway_Out1_width: integer := 1
  );
  port (
    ce: in std_logic;
    CLKIN_P: in std_logic;
    clr: in std_logic;
    LOCKED: out std_logic; --kmv
    gaussianlx5_Gateway_In: in std_logic_vector (gaussianlx5_Gateway_In_width - 1 downto 0);
    gaussianlx5_Gateway_In_valid: in std_logic;
    gaussianlx5_Gateway_Out1: inout std_logic_vector (gaussianlx5_Gateway_Out1_width - 1 downto 0);
    gaussianlx5_Gateway_Out1_valid: inout std_logic
  );
end gaussianlx5;

architecture structural of gaussianlx5 is
-----kmv-----
component CLKDLL
  port (CLKIN, CLKFB, RST : in STD_LOGIC;
        CLK0, CLK90, CLK180, CLK270, CLK2X, CLKDV, LOCKED : out std_logic);
end component;

component IBUFG
  port (I : in STD_LOGIC; O : out std_logic);
end component;

component BUFG
  port (I : in STD_LOGIC; O : out std_logic);
end component;
-----kmv-----
component clock_driver
  port (
    ce: in std_logic;
    ce40: inout std_logic;
    clk: in std_logic;
    clk40: inout std_logic;
    clr: in std_logic;
    clr40: inout std_logic
  );
end component;
component xlconvert
  generic (
    din_arith: integer := xlSigned;
    din_bin_pt: integer := 30;
    din_width: integer := 39;
    dout_arith: integer := xlUnsigned;
    dout_bin_pt: integer := 15;
    dout_width: integer := 16;
    latency: integer := 4;
    overflow: integer := xlSaturate;
    quantization: integer := xlRound
  );
end component;

```

```

);
port (
    ce: in std_logic;
    clk: in std_logic;
    clr: in std_logic;
    din: in std_logic_vector (din_width - 1 downto 0);
    din_valid: in std_logic;
    dout: inout std_logic_vector (dout_width - 1 downto 0);
    dout_valid: inout std_logic
);
end component;
component xlrelational
generic (
    a_arith: integer := xlSigned;
    a_bin_pt: integer := 30;
    a_width: integer := 39;
    b_arith: integer := xlSigned;
    b_bin_pt: integer := 0;
    b_width: integer := 1;
    c_data_type: integer := 0;
    c_has_a_eq_b: integer := 0;
    c_has_a_ge_b: integer := 0;
    c_has_a_gt_b: integer := 0;
    c_has_a_le_b: integer := 0;
    c_has_a_lt_b: integer := 0;
    c_has_a_ne_b: integer := 0;
    c_has_ce: integer := 1;
    c_has_qa_eq_b: integer := 0;
    c_has_qa_ge_b: integer := 1;
    c_has_qa_gt_b: integer := 0;
    c_has_qa_le_b: integer := 0;
    c_has_qa_lt_b: integer := 0;
    c_has_qa_ne_b: integer := 0;
    c_has_sclr: integer := 1;
    c_pipe_stages: integer := 0;
    c_width: integer := 39;
    core: boolean := true;
    dout_arith: integer := xlUnsigned;
    dout_bin_pt: integer := 0;
    dout_width: integer := 1;
    latency: integer := 3
);
port (
    a: in std_logic_vector (a_width - 1 downto 0);
    a_valid: in std_logic;
    b: in std_logic_vector (b_width - 1 downto 0);
    b_valid: in std_logic;
    ce: in std_logic;
    clk: in std_logic;
    clr: in std_logic;
    dout: inout std_logic_vector (dout_width - 1 downto 0);
    dout_valid: inout std_logic
);
end component;
component xldsamp
generic (
    d_arith: integer := xlUnsigned;
    d_bin_pt: integer := 0;
    d_width: integer := 1;
    init_valid: integer := 0;
    passthru: integer := 1;
    q_arith: integer := xlUnsigned;
    q_bin_pt: integer := 0;
    q_width: integer := 1
);
port (
    d: in std_logic_vector (d_width - 1 downto 0);
    d_valid: in std_logic;
    dest_ce: in std_logic;
    dest_clk: in std_logic;

```

```

dest_clr: in std_logic;
q: inout std_logic_vector (q_width - 1 downto 0);
q_valid: inout std_logic;
src_ce: in std_logic;
src_clk: in std_logic;
src_clr: in std_logic
);
end component;
component xlconstant
generic (
  const_arith: integer := xlSigned;
  const_bin_pt: integer := 0;
  const_index: integer := 0;
  const_width: integer := 1;
  dout_arith: integer := xlSigned;
  dout_bin_pt: integer := 0;
  dout_width: integer := 1
);
port (
  dout: inout std_logic_vector (dout_width - 1 downto 0);
  dout_valid: inout std_logic
);
end component;
component xlnegate
generic (
  a_arith: integer := xlSigned;
  a_bin_pt: integer := 30;
  a_width: integer := 38;
  c_has_q: integer := 0;
  c_has_s: integer := 1;
  c_width: integer := 38;
  core: boolean := true;
  latency: integer := 0;
  overflow: integer := xlWrap;
  p_arith: integer := xlSigned;
  p_bin_pt: integer := 30;
  p_width: integer := 39;
  quantization: integer := xlTruncate
);
port (
  a: in std_logic_vector (a_width - 1 downto 0);
  a_valid: in std_logic;
  ce: in std_logic;
  clk: in std_logic;
  clr: in std_logic;
  p: inout std_logic_vector (p_width - 1 downto 0);
  p_valid: inout std_logic
);
end component;
component xladdsub
generic (
  a_arith: integer := xlUnsigned;
  a_bin_pt: integer := 15;
  a_width: integer := 16;
  b_arith: integer := xlUnsigned;
  b_bin_pt: integer := 15;
  b_width: integer := 16;
  full_s_arith: integer := xlUnsigned;
  full_s_width: integer := 17;
  latency: integer := 5;
  mode: integer := 1;
  overflow: integer := xlWrap;
  quantization: integer := xlTruncate;
  s_arith: integer := xlUnsigned;
  s_bin_pt: integer := 15;
  s_width: integer := 17
);
port (
  a: in std_logic_vector (a_width - 1 downto 0);
  a_valid: in std_logic;

```

```

b: in std_logic_vector (b_width - 1 downto 0);
b_valid: in std_logic;
ce: in std_logic;
clk: in std_logic;
clr: in std_logic;
s: inout std_logic_vector (s_width - 1 downto 0);
s_valid: inout std_logic
);
end component;
component xlfir
generic (
-- synopsys translate_off
  c_mem_init_file: string := "gaussianlx5_FIR3.mif";
-- synopsys translate_on
  c_baat: integer := 16;
  c_channels: integer := 1;
  c_coeff_type: integer := 0;
  c_coeff_width: integer := 16;
  c_data_type: integer := 0;
  c_data_width: integer := 16;
  c_filter_type: integer := 0;
  c_has_sel_i: integer := 0;
  c_has_sel_o: integer := 0;
  c_latency: integer := 12;
  c_reg_output: integer := 1;
  c_response: integer := 0;
  c_result_width: integer := 38;
  c_saturate: integer := 0;
  c_taps: integer := 49;
  c_use_model_func: integer := 0;
  c_zpf: integer := 1;
  clks_per_sample: integer := 1;
  din_arith: integer := xlSigned;
  din_bin_pt: integer := 15;
  din_width: integer := 16;
  dout_arith: integer := xlSigned;
  dout_bin_pt: integer := 30;
  dout_width: integer := 38;
  extra_registers: integer := 12;
  latency: integer := 25;
  uid: integer := 2
);
port (
  core_ce: in std_logic;
  core_clk: in std_logic;
  core_clr: in std_logic;
  din: in std_logic_vector (din_width - 1 downto 0);
  din_valid: in std_logic;
  dout: inout std_logic_vector (dout_width - 1 downto 0);
  dout_valid: inout std_logic;
  sample_ce: in std_logic;
  sample_clk: in std_logic;
  sample_clr: in std_logic
);
end component;

signal CLKIN, CLK, CLK0 : std_logic; --kmv--
signal ce40: std_logic;
signal clk40: std_logic;
signal clr40: std_logic;
signal from_fir_1: std_logic_vector (37 downto 0);
signal from_fir_1_valid: std_logic;
signal from_fir_2: std_logic_vector (37 downto 0);
signal from_fir_2_valid: std_logic;
signal gnd: std_logic;
signal net: std_logic_vector (15 downto 0);
signal net1: std_logic_vector (38 downto 0);
signal net10: std_logic_vector (37 downto 0);
signal net10_valid: std_logic;
signal net11: std_logic_vector (0 downto 0);

```

```

signal net11_valid: std_logic;
signal net12: std_logic_vector (0 downto 0);
signal net12_valid: std_logic;
signal net13: std_logic_vector (38 downto 0);
signal net13_valid: std_logic;
signal net14: std_logic_vector (0 downto 0);
signal net14_valid: std_logic;
signal net1_valid: std_logic;
signal net2: std_logic_vector (16 downto 0);
signal net2_valid: std_logic;

signal net3: std_logic_vector (15 downto 0);
signal net3_valid: std_logic;
signal net4: std_logic_vector (15 downto 0);
signal net4_valid: std_logic;
signal net5: std_logic_vector (37 downto 0);
signal net5_valid: std_logic;
signal net6: std_logic_vector (38 downto 0);
signal net6_valid: std_logic;
signal net7: std_logic_vector (16 downto 0);
signal net7_valid: std_logic;
signal net8: std_logic_vector (15 downto 0);
signal net8_valid: std_logic;
signal net9: std_logic_vector (15 downto 0);
signal net9_valid: std_logic;
signal net_valid: std_logic;

begin
-----kmv-----
U1: IBUFG port map (I=>CLKIN_P, O=>CLKIN);

U2: CLKDLL port map (CLKIN=>CLKIN, CLKFB=>CLK, RST=>clr,
                      CLK0=>CLK0, LOCKED=>LOCKED);

U3: BUFG port map (I=>CLK0, O=>CLK);
-----kmv-----

gnd <= '0';

net <= gaussianxlx5_Gateway_In;
gaussianxlx5_Gateway_Out1 <= net14;
gaussianxlx5_Gateway_Out1_valid <= net14_valid;
net_valid <= gaussianxlx5_Gateway_In_valid;

Add1: xladdsub
generic map (
  a_arith => xlUnsigned,
  a_bin_pt => 15,
  a_width => 16,
  b_arith => xlUnsigned,
  b_bin_pt => 15,
  b_width => 16,
  full_s_arith => xlUnsigned,
  full_s_width => 17,
  latency => 5,
  mode => 1,
  overflow => xlWrap,
  quantization => xlTruncate,
  s_arith => xlUnsigned,
  s_bin_pt => 15,
  s_width => 17
)
port map (
  a => net4,
  a_valid => net4_valid,
  b => net3,
  b_valid => net3_valid,
  ce => ce,
  clk => clk,

```

```

    clr => clr,
    s => net2,
    s_valid => net2_valid
) ;

Add2: xladdsub
generic map (
  a_arith => xlUnsigned,
  a_bin_pt => 15,
  a_width => 16,
  b_arith => xlUnsigned,
  b_bin_pt => 15,
  b_width => 16,
  full_s_arith => xlUnsigned,
  full_s_width => 17,
  latency => 5,
  mode => 1,
  overflow => xlWrap,
  quantization => xlTruncate,
  s_arith => xlUnsigned,
  s_bin_pt => 15,
  s_width => 17
)
port map (
  a => net9,
  a_valid => net9_valid,
  b => net8,
  b_valid => net8_valid,
  ce => ce,
  clk => clk,
  clr => clr,
  s => net7,
  s_valid => net7_valid
) ;

Constant1: xlconstant
generic map (
  const_arith => xlSigned,
  const_bin_pt => 0,
  const_index => 0,
  const_width => 1,
  dout_arith => xlSigned,
  dout_bin_pt => 0,
  dout_width => 1
)
port map (
  dout => net11,
  dout_valid => net11_valid
) ;

Convert: xlconvert
generic map (
  din_arith => xlSigned,
  din_bin_pt => 30,
  din_width => 38,
  dout_arith => xlUnsigned,
  dout_bin_pt => 15,
  dout_width => 16,
  latency => 4,
  overflow => xlSaturate,
  quantization => xlRound
)
port map (
  ce => ce,
  clk => clk,
  clr => clr,
  din => net5,
  din_valid => net5_valid,
  dout => net3,
  dout_valid => net3_valid
)

```

```
) ;

Convert1: xlconvert
generic map (
  din_arith => xlSigned,
  din_bin_pt => 30,
  din_width => 39,
  dout_arith => xlUnsigned,
  dout_bin_pt => 15,
  dout_width => 16,
  latency => 4,
  overflow => xlSaturate,
  quantization => xlRound
)
port map (
  ce => ce,
  clk => clk,
  clr => clr,
  din => net1,
  din_valid => net1_valid,
  dout => net4,
  dout_valid => net4_valid
);
```

```
Convert2: xlconvert
generic map (
  din_arith => xlSigned,
  din_bin_pt => 30,
  din_width => 38,
  dout_arith => xlUnsigned,
  dout_bin_pt => 15,
  dout_width => 16,
  latency => 4,
  overflow => xlSaturate,
  quantization => xlRound
)
port map (
  ce => ce,
  clk => clk,
  clr => clr,
  din => net10,
  din_valid => net10_valid,
  dout => net8,
  dout_valid => net8_valid
);
```

```
Convert3: xlconvert
generic map (
  din_arith => xlSigned,
  din_bin_pt => 30,
  din_width => 39,
  dout_arith => xlUnsigned,
  dout_bin_pt => 15,
  dout_width => 16,
  latency => 4,
  overflow => xlSaturate,
  quantization => xlRound
)
port map (
  ce => ce,
  clk => clk,
  clr => clr,
  din => net6,
  din_valid => net6_valid,
  dout => net9,
  dout_valid => net9_valid
);
```

```
Down_Sample: xlldsamp
generic map (
```

```

d_arith => xlUnsigned,
d_bin_pt => 0,
d_width => 1,
init_valid => 0,
passthru => 1,
q_arith => xlUnsigned,
q_bin_pt => 0,
q_width => 1
)
port map (
  d => net12,
  d_valid => net12_valid,
  dest_ce => ce40,
  dest_clk => clk40,
  dest_clr => clr40,
  q => net14,
  q_valid => net14_valid,
  src_ce => ce,
  src_clk => clk,
  src_clr => clr
);

FIR: xlfir
  generic map (
-- synopsys translate_off
    c_mem_init_file => "gaussianlx5_FIR.mif",
-- synopsys translate_on
    c_baat => 17,
    c_channels => 1,
    c_coeff_type => 0,
    c_coeff_width => 16,
    c_data_type => 1,
    c_data_width => 17,
    c_filter_type => 0,
    c_has_sel_i => 0,
    c_has_sel_o => 0,
    c_latency => 11,
    c_reg_output => 1,
    c_response => 0,
    c_result_width => 38,
    c_saturate => 0,
    c_taps => 17,
    c_use_model_func => 0,
    c_zpf => 1,
    clks_per_sample => 1,
    din_arith => xlUnsigned,
    din_bin_pt => 15,
    din_width => 17,
    dout_arith => xlSigned,
    dout_bin_pt => 30,
    dout_width => 38,
    extra_registers => 0,
    latency => 12,
    uid => 1
)
port map (
  core_ce => ce,
  core_clk => clk,
  core_clr => clr,
  din => net2,
  din_valid => net2_valid,
  dout => from_fir_1,
  dout_valid => from_fir_1_valid,
  sample_ce => ce,
  sample_clk => clk,
  sample_clr => clr
);

FIR1: xlfir
  generic map (

```

```

-- synopsys translate_off
  c_mem_init_file => "gaussianlx5_FIR1.mif",
-- synopsys translate_on
  c_baat => 17,
  c_channels => 1,
  c_coeff_type => 0,
  c_coeff_width => 16,
  c_data_type => 1,
  c_data_width => 17,
  c_filter_type => 0,
  c_has_sel_i => 0,
  c_has_sel_o => 0,
  c_latency => 11,
  c_reg_output => 1,
  c_response => 0,
  c_result_width => 38,
  c_saturate => 0,
  c_taps => 17,
  c_use_model_func => 0,
  c_zpf => 1,
  clks_per_sample => 1,
  din_arith => xlUnsigned,
  din_bin_pt => 15,
  din_width => 17,
  dout_arith => xlSigned,
  dout_bin_pt => 30,
  dout_width => 38,
  extra_registers => 0,
  latency => 12,
  uid => 3
)
port map (
  core_ce => ce,
  core_clk => clk,
  core_clr => clr,
  din => net7,
  din_valid => net7_valid,
  dout => from_fir_2,
  dout_valid => from_fir_2_valid,
  sample_ce => ce,
  sample_clk => clk,
  sample_clr => clr
);
FIR2: xlfir
  generic map (
-- synopsys translate_off
    c_mem_init_file => "gaussianlx5_FIR2.mif",
-- synopsys translate_on
    c_baat => 16,
    c_channels => 1,
    c_coeff_type => 0,
    c_coeff_width => 16,
    c_data_type => 0,
    c_data_width => 16,
    c_filter_type => 0,
    c_has_sel_i => 0,
    c_has_sel_o => 0,
    c_latency => 12,
    c_reg_output => 1,
    c_response => 0,
    c_result_width => 38,
    c_saturate => 0,
    c_taps => 49,
    c_use_model_func => 0,
    c_zpf => 1,
    clks_per_sample => 1,
    din_arith => xlSigned,
    din_bin_pt => 15,
    din_width => 16,

```

```

dout_arith => xlSigned,
dout_bin_pt => 30,
dout_width => 38,
extra_registers => 12,
latency => 25,
uid => 0
)
port map (
  core_ce => ce,
  core_clk => clk,
  core_clr => clr,
  din => net,
  din_valid => net_valid,
  dout => net5,
  dout_valid => net5_valid,
  sample_ce => ce,
  sample_clk => clk,
  sample_clr => clr
);

FIR3: xlfir
generic map (
-- synopsys translate_off
  c_mem_init_file => "gaussianxlx5_FIR3.mif",
-- synopsys translate_on
  c_baat => 16,
  c_channels => 1,
  c_coeff_type => 0,
  c_coeff_width => 16,
  c_data_type => 0,
  c_data_width => 16,
  c_filter_type => 0,
  c_has_sel_i => 0,
  c_has_sel_o => 0,
  c_latency => 12,
  c_reg_output => 1,
  c_response => 0,
  c_result_width => 38,
  c_saturate => 0,
  c_taps => 49,
  c_use_model_func => 0,
  c_zpf => 1,
  clks_per_sample => 1,
  din_arith => xlSigned,
  din_bin_pt => 15,
  din_width => 16,
  dout_arith => xlSigned,
  dout_bin_pt => 30,
  dout_width => 38,
  extra_registers => 12,
  latency => 25,
  uid => 2
)
port map (
  core_ce => ce,
  core_clk => clk,
  core_clr => clr,
  din => net,
  din_valid => net_valid,
  dout => net10,
  dout_valid => net10_valid,
  sample_ce => ce,
  sample_clk => clk,
  sample_clr => clr
);

Negate: xlnegate
generic map (
  a_arith => xlSigned,
  a_bin_pt => 30,

```

```

a_width => 38,
c_has_q => 0,
c_has_s => 1,
c_width => 38,
core => true,
latency => 0,
overflow => xlWrap,
p_arith => xlSigned,
p_bin_pt => 30,
p_width => 39,
quantization => xlTruncate
)
port map (
  a => net5,
  a_valid => net5_valid,
  ce => gnd,
  clk => gnd,
  clr => gnd,
  p => net1,
  p_valid => net1_valid
);

Negate1: xlnegate
generic map (
  a_arith => xlSigned,
  a_bin_pt => 30,
  a_width => 38,
  c_has_q => 0,
  c_has_s => 1,
  c_width => 38,
  core => true,
  latency => 0,
  overflow => xlWrap,
  p_arith => xlSigned,
  p_bin_pt => 30,
  p_width => 39,
  quantization => xlTruncate
)
port map (
  a => net10,
  a_valid => net10_valid,
  ce => gnd,
  clk => gnd,
  clr => gnd,
  p => net6,
  p_valid => net6_valid
);

Relational: xlrelational
generic map (
  a_arith => xlSigned,
  a_bin_pt => 30,
  a_width => 39,
  b_arith => xlSigned,
  b_bin_pt => 0,
  b_width => 1,
  c_data_type => 0,
  c_has_a_eq_b => 0,
  c_has_a_ge_b => 0,
  c_has_a_gt_b => 0,
  c_has_a_le_b => 0,
  c_has_a_lt_b => 0,
  c_has_a_ne_b => 0,
  c_has_ce => 1,
  c_has_qa_eq_b => 0,
  c_has_qa_ge_b => 1,
  c_has_qa_gt_b => 0,
  c_has_qa_le_b => 0,
  c_has_qa_lt_b => 0,
  c_has_qa_ne_b => 0,

```

```

c_has_sclr => 1,
c_pipe_stages => 0,
c_width => 39,
core => true,
dout_arith => xlUnsigned,
dout_bin_pt => 0,
dout_width => 1,
latency => 3
)
port map (
  a => net13,
  a_valid => net13_valid,
  b => net11,
  b_valid => net11_valid,
  ce => ce,
  clk => clk,
  clr => clr,
  dout => net12,
  dout_valid => net12_valid
);

addsub: xladdsub
generic map (
  a_arith => xlSigned,
  a_bin_pt => 30,
  a_width => 38,
  b_arith => xlSigned,
  b_bin_pt => 30,
  b_width => 38,
  full_s_arith => xlSigned,
  full_s_width => 39,
  latency => 5,
  mode => 2,
  overflow => xlWrap,
  quantization => xlTruncate,
  s_arith => xlSigned,
  s_bin_pt => 30,
  s_width => 39
)
port map (
  a => from_fir_1,
  a_valid => from_fir_1_valid,
  b => from_fir_2,
  b_valid => from_fir_2_valid,
  ce => ce,
  clk => clk,
  clr => clr,
  s => net13,
  s_valid => net13_valid
);

clock_driver_1: clock_driver
port map (
  ce => ce,
  ce40 => ce40,
  clk => clk,
  clk40 => clk40,
  clr => clr,
  clr40 => clr40
);
end structural;

```



Distributed Arithmetic FIR Filter

V5.0.0

March 9 2001

Product Specification



Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
Phone: +1 408-559-7778
FAX: +1 408-559-7114
URL: www.xilinx.com/ipcenter
Support: www.support.xilinx.com

1 Features

- Drop-in module for VirtexTM, VirtexTM-E, SpartanTM-II and VirtexTM-II FPGAs
- High-performance finite impulse response (FIR), half-band, Hilbert transform, interpolated filters, polyphase decimator, polyphase interpolator, half-band decimator and half-band interpolator implementations
- Highly parameterizable
- 2-to-1024 taps
- 1-to-32 bit input data precision
- Signed or unsigned input data
- Signed or unsigned filter coefficients
- 1-to-32 bit coefficient precision
- 1-to-8 channels
- Support for interpolation and decimation factors of between 1 and 8 inclusive
- Coefficient symmetry exploited (symmetric/negative-symmetric) to produce compact implementations
- Serial and parallel filters supported. The user may specify the degree of parallelism and tradeoff FPGA logic resources for sample rate in order to generate an optimal design
- Data-flow style core interface and control
- On-line coefficient reload capability
- Incorporates Xilinx Smart-IP technology for maximum performance
- To be used with version 3.1i or later of the Xilinx CORE Generator System

2 General Description

The Xilinx filter Core is a highly parameterizable, area efficient high-performance FIR filter. Several highly optimized filters can be realized with the filter Core Generator: single-rate, half-band, Hilbert transform and interpolated filters, in addition to polyphase decimators and interpolators and half-band decimators and interpolators. Structure in the coefficient set is exploited to produce area-efficient FPGA implementations. Sufficient arithmetic precision is employed in the internal data-path to avoid the possibility of overflow. The filter always presents a full-precision result at its output port.

The conventional single-rate FIR version of the core computes the convolution sum defined in Eq. (1)

$$y(k) = \sum_{n=0}^{N-1} a(n)x(k-n) \quad k = 0,1,\dots \quad (1)$$

where N is the number of filter coefficients. The conventional tapped delay line realization of this inner-product calculation is shown in Figure 1.

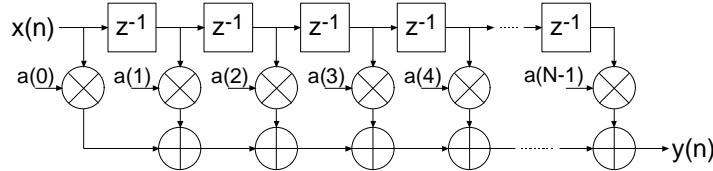


Figure 1: Conventional tapped-delay line FIR filter mechanization.

Even though the figure is a useful conceptualization of the computation performed by the core, the actual FPGA realization is quite different. A distributed arithmetic (DA) realization [1] [2] is employed. With this approach there are no explicit multipliers employed in the design, only look-up tables (LUTs), shift registers and a scaling accumulator.

2.1 Filter Realization – *Distributed Arithmetic*

A simplified view of a DA FIR is shown in Figure 2. In its most obvious and direct form, DA based computations are bit-serial in nature – serial distributed arithmetic (SDA) FIR. Extensions to the basic algorithm remove this potential throughput limitation [2]. The advantage of a distributed arithmetic approach is its efficiency of mechanization. The basic operations required are a sequence of table look-ups, additions, subtractions and shifts of the input data sequence. All of these functions efficiently map to FPGAs. Input samples are presented to the input parallel-to-serial shift register (PSC) at the input signal sample rate. As the new sample is serialized, the bit-wide output is presented to a bit-serial shift register or *time-skew buffer* (TSB). The TSB stores the input sample history in a bit-serial format and is used in forming the required inner-product computation. The TSB is itself constructed using a cascade of shorter bit-serial shift registers. The nodes in the cascade connection of TSB's are used as address inputs to a look-up table. This LUT stores all possible partial products [2] over the filter coefficient space.

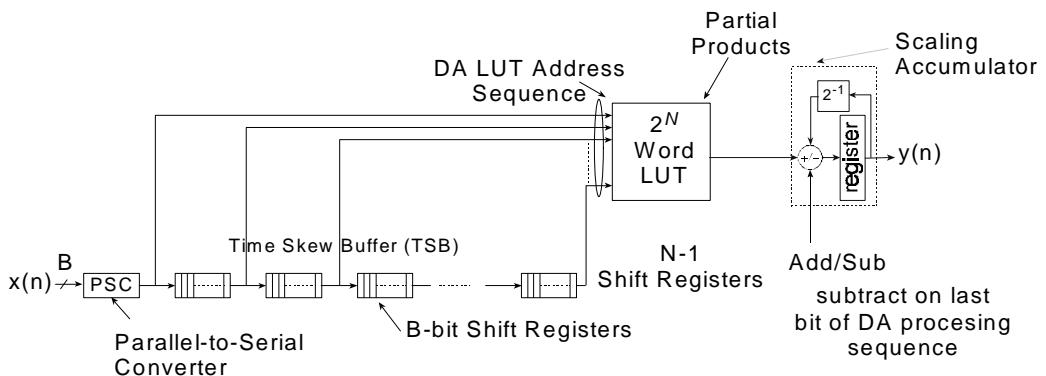


Figure 2: Serial distributed arithmetic FIR filter.

Several observations provide valuable insight into the operation of a DA FIR filter. In a conventional multiply-accumulate (MAC) based FIR realization, the sample throughput is coupled to the filter length. With a DA architecture the system sample rate is related to the bit precision of the input data samples. Each bit of an input sample must be indexed and processed in turn before a new output sample is available. For B -bit precision input samples, B clock cycles are required to form a new output sample for a non-symmetrical filter, and $B+1$ clock cycles are needed for a symmetrical filter. The rate at which data bits are indexed occurs at the *bit-clock* rate. The bit-clock frequency is greater than the filter sample rate (f_s) and is equal to Bf_s for a non-symmetrical filter and $(B+1)f_s$ for a symmetrical filter. In a conventional instruction-set (processor) approach to the problem, the required number of multiply-accumulate operations are implemented using a time-shared or *scheduled* MAC unit. The filter sample throughput is inversely proportional to the number of filter taps. As the filter length is increased the system sample rate is proportionately decreased. This is not the case with DA based architectures. The filter sample rate is de-coupled from the filter length. The trade off introduced here is one of silicon area (FPGA logic resources) for time. As the filter length is increased in a DA FIR filter, more logic resources are consumed, but throughput is maintained.

Figure 3 provides a comparison between a DA FIR architecture and a conventional scheduled MAC-based approach. The clock rate is assumed to be 120 MHz for both filter architectures. Several values of input sample precision for the DA FIR are presented. The dependency of the DA filter throughput on the sample precision is apparent from the plots. For 8-bit precision input samples, the DA FIR maintains a higher throughput for filter lengths greater than 8 taps. When the sample precision is increased to 16 bits, the crossover point is 16 taps.

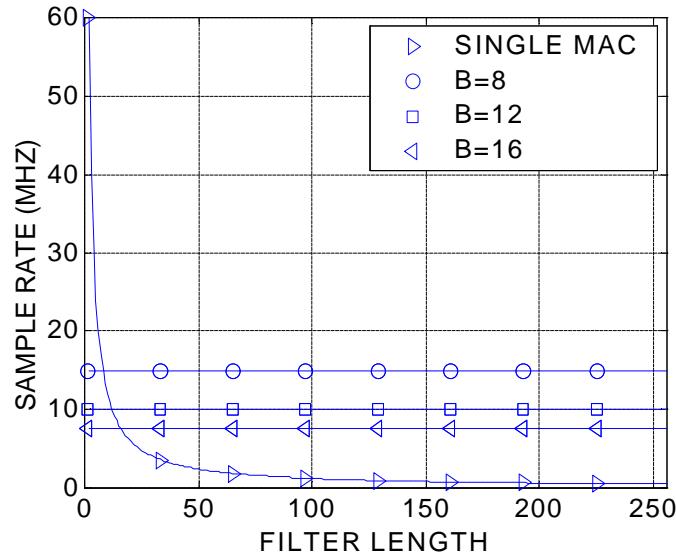


Figure 3: Throughput (sample rate) comparison of single-MAC based FIR and DA FIR as a function of filter length. B is the DA FIR input sample precision. The clock rate is 120 MHz.

Figure 4 provides a similar comparison but for a dual-MAC architecture.

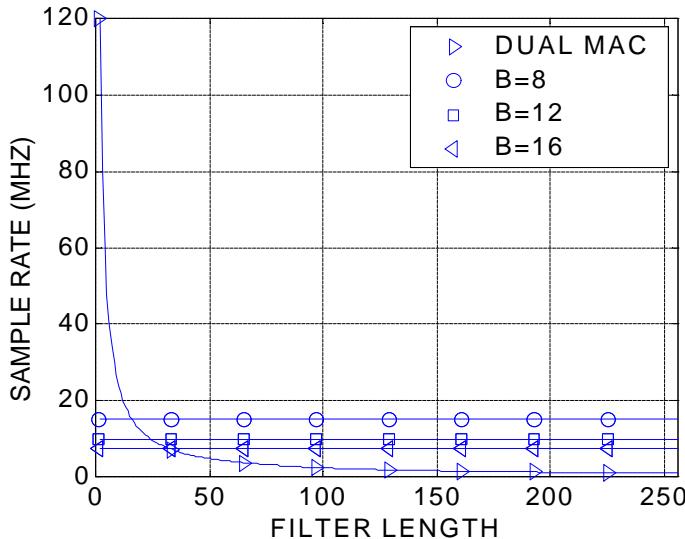


Figure 4: Throughput (sample rate) comparison of dual-MAC based FIR and DA FIR as a function of filter length. B is the DA FIR input sample precision. The clock rate is 120 MHz.

2.2 Increasing the Speed of Multiplication - Parallel Distributed Arithmetic

In its most obvious and direct form, DA based computations are bit-serial in nature – each bit of the samples must be indexed in turn before a new output sample becomes available (SDA FIR). When the input samples are represented with B bits of precision, B clock cycles are required to complete an inner-product calculation (for a non-symmetrical impulse response). Additional speed may be obtained in several ways. One approach is to partition the input words into M subwords and process these subwords in parallel. This method requires M -times as many memory look-up tables and so comes at a cost of increased storage requirements. Maximum speed is achieved by factoring the input variables into single bit subwords. The resulting structure is a fully parallel DA (PDA) FIR filter. With this factoring a new output sample is computed on each clock cycle. PDA FIR filters provide exceptionally high-performance. The Xilinx filter Core provides support for parallel DA FIR implementations. Filters may be designed that process several bits in a clock period, through to a completely parallel architecture that processes all the bits of the input data during a single clock period. For example, consider a non-symmetrical filter with 12-bit precision input samples. Using a serial DA filter, new output samples are available every 12 clock periods. If the data samples are processed 2-bits-at-a-time (2-BAAT), a new output sample is ready every $12/2 = 6$ clock cycles. With 3-, 4-, 6- and 12-BAAT implementations, a new result is available every 4, 3, 2 and 1 clock cycles respectively.

Another way to view the problem is in terms of the number of clock cycles L needed to produce a filter output sample. And indeed, this is how the degree of computation parallelism is presented to the user on the filter design GUI. So for example, let's consider a filter core with a master system clock (and this is not necessarily the filter sample rate) equal to 150 MHz. Also assume that the input sample precision is 12 bits and that the impulse response is not symmetrical. For this set of parameters the valid values of L (and these are presented on the core GUI) are 12, 6, 4, 3, 2 and 1. The corresponding filter sample rate (or throughput) for each value of L is $150/12=12.5$, $150/6=25$, $150/4=37.5$, $150/3=50$, $150/2=75$ and $150/1=150$ MHz respectively. If the filter employs a symmetrical impulse response the valid values of L are different – and this is associated with the hardware architecture that is employed to exploit the coefficient symmetry in order to produce

the most compact (in terms of FPGA logic resources) realization. So for a filter with 12-bit precision input samples and a symmetrical impulse response, the valid values of L are 13, 7, 5, 4, 3, 2 and 1. Again, using a filter core master clock frequency of 150 MHz, the sample rate for each value of L is 11.539, 21.429, 30, 37.5, 50, 75 and 150 MHz respectively.

The higher the degree of filter parallelism (fewer number of clock cycles per output sample or smaller L), the greater the FPGA logic resources required to implement the design.

Specifying the number of clock cycles per output sample is an extremely powerful mechanism that allows the designer to tradeoff silicon area with filter throughput.

2.3 Exploiting Filter Symmetry

The impulse response for many filters possess significant symmetry. This symmetry can be exploited to minimize arithmetic requirements and produce area efficient filter realizations.

Figure 5 shows the impulse response for a 9-tap symmetric FIR filter.

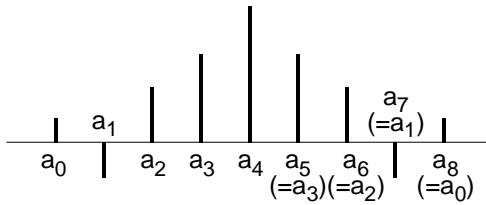


Figure 5: Symmetric FIR – odd number of terms.

Instead of implementing this filter using the architecture shown in Figure 1, the more efficient signal flow-graph in Figure 6 can be used. In general the former approach requires N multiplications and $(N-1)$ additions. In contrast, the architecture in Figure 6 requires only $\lceil N/2 \rceil$ multiplications and approximately N additions. This significant reduction in the computation workload can be exploited to generate efficient filter hardware implementations.

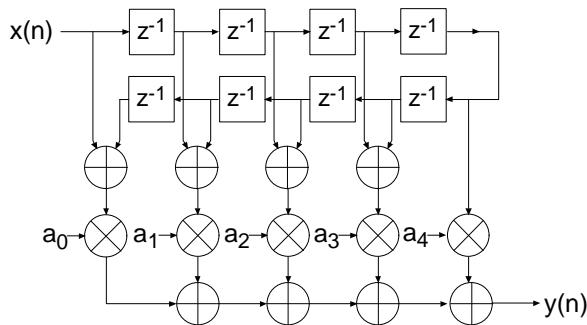


Figure 6: Exploiting coefficient symmetry – odd number of filter taps.

Coefficient symmetry for an even number of terms can be exploited as shown in Figure 7.

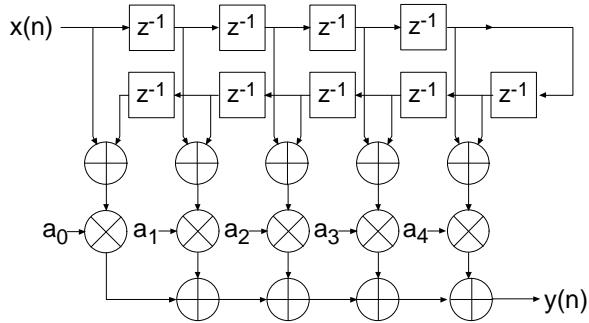


Figure 7: Exploiting coefficient symmetry – even number of filter taps.

The impulse response for a negative, or odd, symmetric filter is shown in Figure 8.

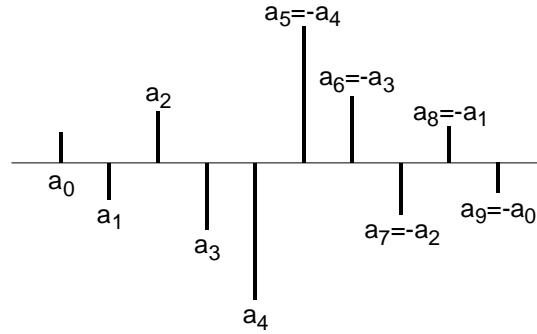


Figure 8: Negative Symmetric impulse response.

This symmetry is easily exploited in a manner similar to that shown in Figure 6 and Figure 7. In this case the middle layer of adders are replaced by subtractors as illustrated in Figure 9.

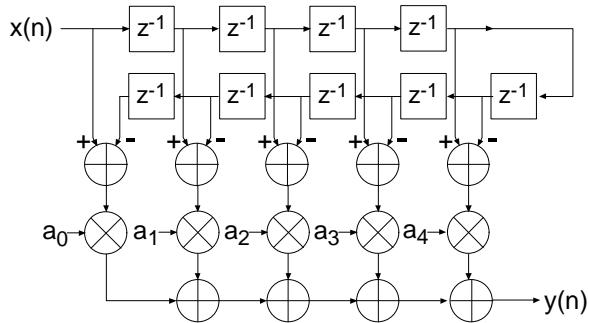


Figure 9: FIR architecture exploiting negative symmetry.

Again, as highlighted above, the symmetry properties can be utilized to produce an efficient hardware realization.

The example considered here illustrates a filter with an even number of terms, the filter structure for an odd number of terms is a simple extension of the same principle.

Even though none of the filter classes supported by the filter core use explicit multipliers, the various symmetries can still be exploited using a distributed arithmetic implementation to produce efficient FPGA realizations.

The filter compiler interface allows the filter symmetry to be specified. When the impulse response does exhibit symmetry, the filter logic requirements can be significantly reduced in comparison to an implementation that does not exploit the impulse response structure. For example a 100 tap non-symmetric filter with 12-bit data samples and 12-bit coefficients consumes 519 Virtex logic slices [3]. In contrast, a 100 tap symmetric filter is realized with 354 slices. This represents approximately a 30% savings in area.

3 Filter Throughput

The signal sample rate for a filter is a function of the core bit clock frequency, f_{clk} Hz, the input data sample precision B , the number of channels, the number of clock cycles (L) per output sample and the coefficient symmetry. For a single channel non-symmetrical FIR filter using $L=B$ clock cycles per output sample, the filter sample frequency, or sample throughput, is f_{clk}/B Hz. If the filter is symmetrical, then the sample rate is $f_{clk}/(B+1)$ Hz. If the number of clock cycles per output sample is changed to $L=1$, the sample throughput is simply f_{clk} Hz. For $L=2$, the throughput is $f_{clk}/2$ Hz.

As a specific example, consider a filter with a core clock frequency equal to 100 MHz, 10-bit input samples, $L=10$ and a non-symmetrical coefficient set. The filter sample rate is $100/10 = 10$ MHz. Observe that this figure is independent of the number of filter taps. If a symmetrical realization had been generated, the sample throughput would be $100/11 = 9.0909$ MHz. For $L=1$ the sample rate would be 100 MHz (non-symmetrical FIR).

If the input sample precision is changed to 8 bits, with $L=8$, the filter sample rate for a non-symmetrical filter would be $100/8 = 12.5$ MHz.

4 Processing Multiple Channels

In many applications the same filter must be applied to several data streams. A common example is the simple digital down converter shown in Figure 10. Here a complex base-band signal $x(n) = x_I(n) + jx_Q(n)$ is applied to a matched filter $M(z)$. The in-phase and quadrature components are each processed by the same filter.

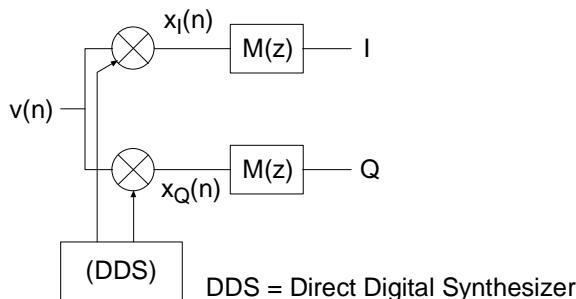


Figure 10: Digital down converter.

One candidate solution to this problem is to simply employ two separate filters. However, this can be wasteful of logic resources. A more efficient design can be realized using a filter architecture that shares logic resources between multiple sample streams. Several of the filter classes

supported by the filter core provide in-built support for multi-channel processing and can accommodate up to 8 independent data streams. As more channels are processed by a filter core, the sample throughput is commensurately reduced. For example, if the sample rate (not the core bit clock *CLK*) for a single channel filter is f_s , a two-channel version of the same filter will process two sample streams, each with a sample rate of $f_s/2$. A three channel version of the filter will process three data streams, and support a sample rate of $f_s/3$ for each of the streams.

A multi-channel filter implementation is very efficient in terms of the amount of logic resources utilized. A filter with two or more channels can be realized using virtually the same amount of logic resources as a single channel version of the same filter. The tradeoff that needs to be addressed when employing multi-channel filters is one of sample rate versus logic requirements. As the number of channels is increased, the logic area remains approximately constant, but the sample rate for an individual input stream will decrease.

The number of channels to be supported by a filter core is specified through the filter customization GUI.

The multirate filters (polyphase decimator, polyphase interpolator, half-band decimator and half-band interpolator) provide support for single channel operation only.

5 Filter Configurations

Eight classes of filters are supported by the filter compiler: 1. conventional single-rate FIR, 2. half-band FIR, 3. Hilbert transform [5], 4. interpolated FIR [4] [6], 5. Polyphase decimator, 6. Polyphase interpolator, 7. half-band decimator and 8. half-band interpolator. The *interpolated* FIR should not be confused with an *interpolation* filter. Interpolated filters are single rate systems that can be employed to produce efficient realizations of narrow-band filters, and with some minor enhancements, wide-band filters can also be accommodated.

Each of the filter categories supported by the DA FIR core are described in separate sections below.

5.1 Single Rate FIR

The basic FIR filter core is a single-rate (input sample rate = output sample rate) finite impulse response filter. Figure 11 shows the schematic symbol for a single channel instance of this module. Filter input data is supplied on the *DIN* port and filter output samples are presented on the *DOUT* port. The *CLK* signal is the bit-rate clock for the core, and is recognized as being different (higher frequency) to the input signal sample frequency. The *ND*, *RDY* and *RFD* signals are filter interface/control signals that permit a simple and efficient data-flow style interface for supplying input samples and reading output samples from the filter. The core interface signals are discussed in detail in the *Interface and Control* section of the product guide.

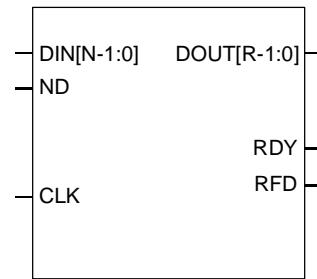


Figure 11: Single channel FIR symbol.

A P -channel filter core is shown in Figure 12. The output ports SEL_I and SEL_O are provided to indicate the active input and output data stream respectively. The SEL_I signal can be used to multiplex several input sources on to the time-shared input bus DIN . SEL_I is employed as the multiplexer select signal in this example. In a similar manner, the SEL_O signal may be used to de-multiplex the time-division multiplexed filter output bus $DOUT$. This is useful for generating P separate filter output samples to present to down-stream processes.

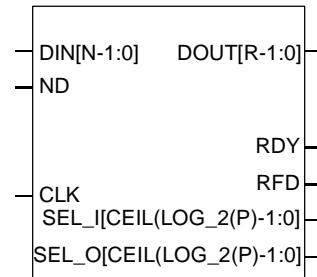


Figure 12: Multi-channel FIR symbol.

Table 1 lists the FIR filter port names and port functional definitions.

Table 1: FIR core signal pinout.

Signal Name	Direction	Description
CLK	Input	BIT CLOCK (active rising edge)
ND	Input	NEW DATA (active high) – When this signal is asserted the data sample presented on the DIN port is loaded into the PSC shown in Figure 2 and an inner-product computation is started. ND should not be asserted while RFD is low. Doing so will corrupt the calculation.
DIN[N-1:0]	Input	FILTER INPUT DATA SAMPLE – N-bit wide filter input sample.
RDY	Output	FILTER OUTPUT SAMPLE READY (active high) Indicates that a new filter output sample is available on the $DOUT$ port.
RFD	Output	READY FOR DATA – (active high) Indicates

		when the final bit of the current data sample is about to be processed and new data may be supplied to the filter.
SEL_I[ceil(log_2(P))-1:0]	Output	INPUT CHANNEL SELECT This is a standard binary count generated by the core that indicates the current filter input channel number.
SEL_O[ceil(log_2(P))-1:0]	Output	OUTPUT CHANNEL SELECT This is a standard binary count generated by the core that indicates the current filter output channel number.
DOUT[R-1:0]	Output	FILTER OUTPUT SAMPLE R-bit wide output sample bus for the FIR, half-band and interpolated filters. R depends on the filter parameters (data precision, coefficient precision, number of taps and coefficient optimization selection) and is always supplied as a full-precision output port to avoid any potential for overflow.
DOUT_I[N-1:0]	Output	FILTER OUTPUT SAMPLE, Hilbert transform – In-phase (I) component. A Hilbert transform accepts real valued input data and produces a complex result. This port is the real or in-phase component of the result. Since this output port is simply an access point to the center of the filter memory buffer, it carries the same precision as the input sample data stream, i.e., N bits.
DOUT_Q[R-1:0]	Output	FILTER OUTPUT SAMPLE, Hilbert transform – quadrature (Q) component. A Hilbert transform accepts real valued input data and produces a complex result. This port is the imaginary or quadrature component of the result.

5.2 Half-Band FIR

The frequency response for a half-band filter is shown in Figure 13.

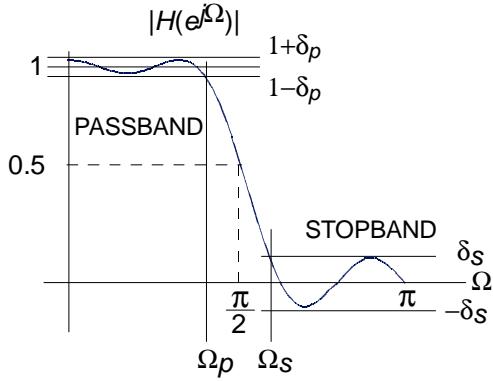


Figure 13: Half-band filter – magnitude frequency response.

Observe from the figure that the magnitude frequency response is symmetrical about quarter sample frequency $\pi/2$ radians. The sample rate is normalized to 2π radians/sec. The passband and stopband frequencies are positioned such that

$$\Omega_p = p - \Omega_s$$

The passband and stopband ripple, d_p and d_s respectively, are equal $d_p = d_s$. These properties are reflected in the filter impulse response. It can be shown [5] that approximately half of the filter coefficients will be zero for an odd number of taps. This is illustrated in Figure 14 for an 11-tap half-band filter.

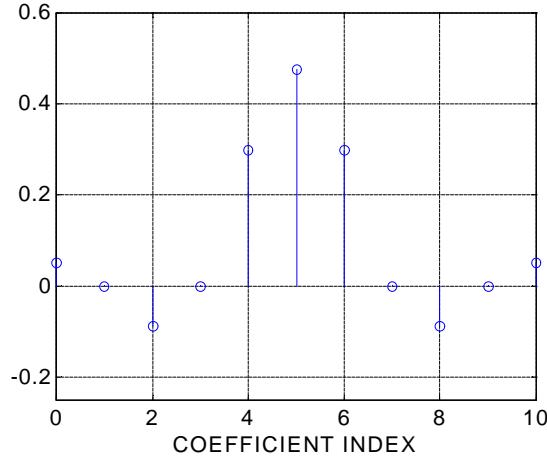


Figure 14: Half-band filter impulse response.

The interleaved zero values in the coefficient data can be exploited to realize an efficient realization like that shown in Figure 15.

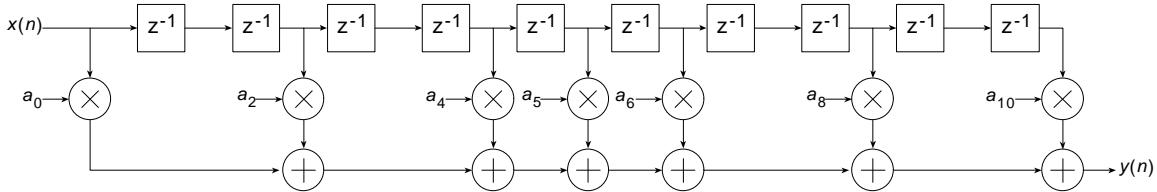


Figure 15: Half-band filter architecture.

This same structure, can of course, be utilized to generate an efficient DA FPGA implementation. The Half-Band filter selection in the compiler is intended for this purpose. This filter is available in the *Filter Type* field of the user interface. The user must supply the complete list of filter coefficients, including the 0 value samples, when using the half-band filter. The filter coefficient file format is discussed in greater detail in the *Filter Coefficient Data* section.

The half-band filter core has the same port definitions as the single-rate FIR filter.

5.3 Hilbert Transform

Hilbert transformers [5] are used in a variety of ways in digital communication systems.

An ideal Hilbert transform provides a phase shift of 90 degrees for positive frequencies and -90 degrees for negative frequencies. It can be shown [5] that the impulse response corresponding to this frequency domain characteristic is odd-symmetric and has interleaved zero's as shown in Figure 16.

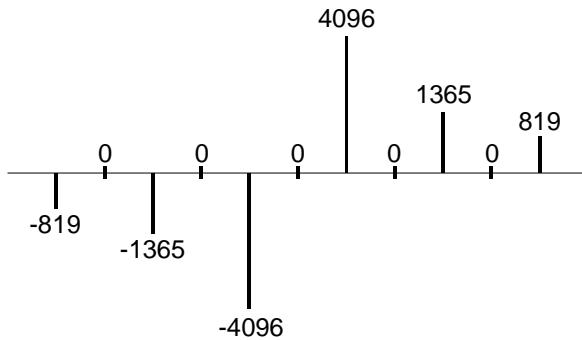


Figure 16: Impulse response of a Hilbert transformer.

Both the alternating zero-valued coefficients and the negative symmetry can be utilized to produce an efficient hardware realization. A Hilbert transformer accepts a real-valued signal and produces a complex (I,Q) output signal. The quadrature (Q) component of the output signal is produced by a FIR filter with an impulse response like that shown in Figure 16. The in-phase (I) component is simply the input signal delayed by an appropriate amount to compensate for the phase delay of the FIR process employed for generating the Q output. This is easily and efficiently achieved by accessing the center tap of the sample history delay of the Q channel FIR filter as shown in Figure 17. In this figure $x(n)$ is the real-valued input signal and $y_I(n)$ and $y_Q(n)$ are the in-phase and quadrature outputs respectively.

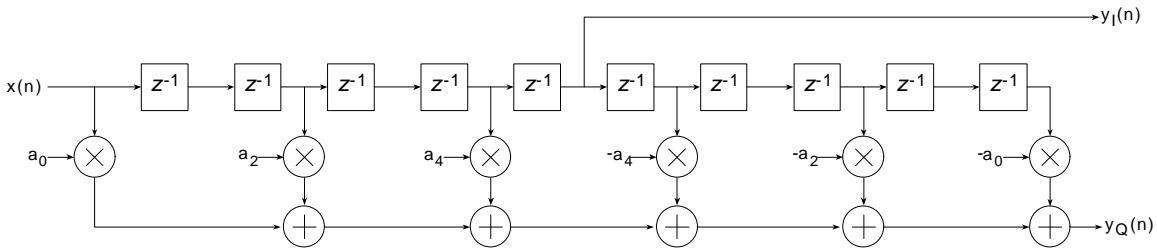


Figure 17: FIR filter realization of a Hilbert transformer.

Figure 18 shows the architecture for a Hilbert transformer that exploits both the zero-valued and the negative symmetry characteristics of the impulse response.

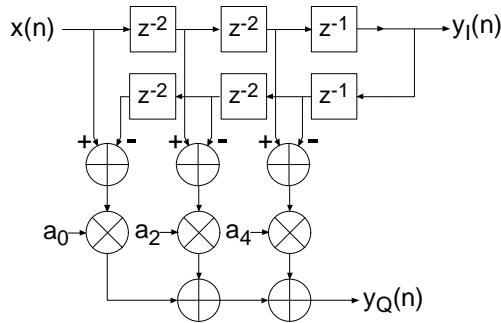


Figure 18: Hilbert transformer exploiting zero-valued filter coefficients and negative symmetry.

The DA equivalent of this architecture is used for realizing the Xilinx Hilbert transformer.

Figure 19 shows the symbol for the Hilbert transform core. The *DIN* port is the filter input signal, and the ports *DOUT_I* and *DOUT_Q* are the I and Q outputs respectively.

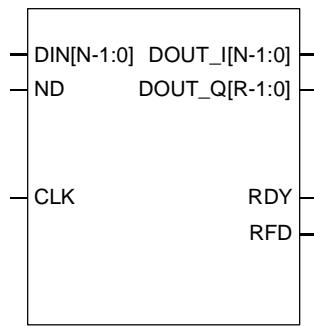


Figure 19: Hilbert transform symbol.

The Hilbert transform core has the same data-flow interface and control signals (*ND*, *RDY*, *RFD*) as the single-rate FIR filter core.

The Hilbert transform core also supports multiple channels as shown in Figure 20.

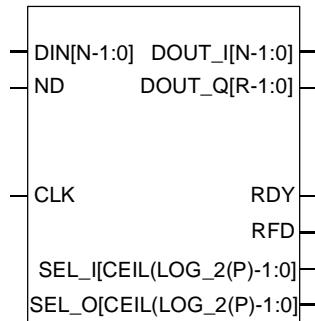
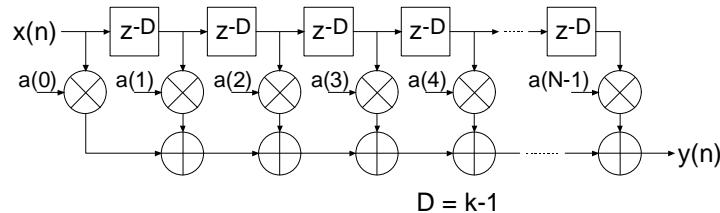


Figure 20: Multi-channel Hilbert transform core.

5.4 Interpolated FIR

An *interpolated FIR* (IFIR) [4] [6] has a similar architecture to a conventional FIR filter, but with the unit delay operator replaced by $k-1$ units of delay. k is referred to as the *zero-packing factor*. An N -tap IFIR filter is shown in Figure 21.

Figure 21: Interpolated FIR (IFIR). The zero-packing factor is k .

This architecture is functionally equivalent to inserting $k-1$ zeros between the coefficients of a prototype filter coefficient set.

Interpolated filters are useful for realizing efficient implementations of both narrow-band and wide-band filters. A filter system based on an IFIR approach requires not only the IFIR but also an image rejection filter. References [4] and [6] provide the details of how these systems are realized, and how to design the IFIR and the image rejection filters.

The IFIR filter core takes advantage of the $k-1$ zeros in the impulse response to realize an area efficient FPGA implementation. The FPGA area required by an IFIR filter is not a strong function of the zero-packing factor.

THE IFIR FILTER IS A SINGLE-RATE STRUCTURE. IT DOES NOT PROVIDE AN EMBEDDED SAMPLE RATE CHANGE – THE INPUT SAMPLE RATE IS THE SAME AS THE OUTPUT SAMPLE RATE.

5.5 Polyphase Decimator

The *polyphase decimation filter* option implements the computationally efficient M -to-1 polyphase decimating filter shown in Figure 22.

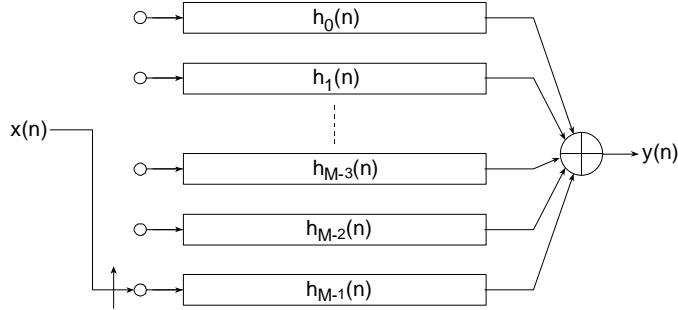


Figure 22: M -to-1 polyphase decimator.

A set of N prototype filter coefficients a_0, a_1, \dots, a_{N-1} are mapped to the M polyphase sub-filters $h_0(n), h_1(n), \dots, h_{M-1}(n)$ according to Eq. (2).

$$h_i(n) = a(i + Mr) \quad i = 0, 1, \dots, M - 1 \quad r = 0, 1, \dots, N - M + i \quad (2)$$

The polyphase segments are accessed by delivering the input samples $x(n)$ to their inputs via an input commutator which starts at the segment index $i = M-1$ and decrements to index 0. After the commutator has executed one cycle and delivered M input samples to the filter, a single output is taken as the summation of the outputs from the polyphase segments. The output sample f'_s rate is

$$f'_s = \frac{f_s}{M}$$

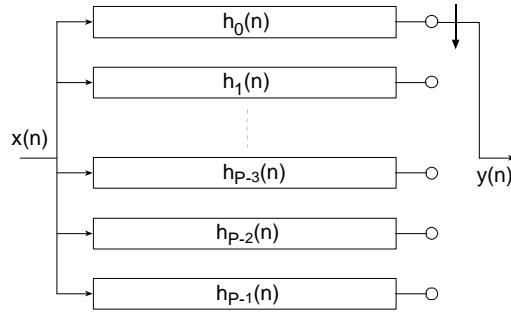
where f_s is sample rate of the input data stream $x(n), n = 0, 1, 2, \dots$. We observe that each of the polyphase segments is operating at the low output sample rate f'_s (compared to the high input sample rate f_s) and a total of N operations are performed per output point.

In the Xilinx decimator, the polyphase segments are realized using distributed arithmetic techniques. M sub-filters, all operating in parallel, are employed in the filter architecture.

The polyphase decimator provides support for single-channel operation only.

5.6 Polyphase Interpolator

The *polyphase interpolation filter* option implements the computationally efficient 1-to- P interpolation filter shown in Figure 23.

Figure 23: 1-to- P polyphase interpolator.

A set of N prototype filter coefficients a_0, a_1, \dots, a_{N-1} are mapped to the P polyphase sub-filters $h_0(n), h_1(n), \dots, h_{P-1}(n)$ according to Eq. (3).

$$h_i(n) = a(i + Pr) \quad i = 0, 1, \dots, P-1 \quad r = 0, 1, \dots, N - P + i \quad (3)$$

Each new input sample $x(n)$ engages all of the polyphase segments in parallel. For each input sample delivered to the filter, P output samples, one from each segment, are delivered to the filter output port as indicated by the commutator in Figure 23.

The output sample f'_s rate is

$$f'_s = f_s P$$

where f_s is sample rate of the input data stream $x(n), n = 0, 1, 2, \dots$. We observe that each of the polyphase segments is operating at the low input sample rate f_s (compared to the high output sample rate f'_s) and a total of N operations are performed per output point.

Like the polyphase decimator, each filter segment in the interpolator is constructed using distributed arithmetic techniques. P concurrently operating segments are employed in the filter realization.

The polyphase interpolator provides support for single-channel operation only.

5.7 Half-Band Decimator

The half-band decimator is a polyphase filter with an embedded 2-to-1 downsampling of the input signal. The structure is shown in Figure 24.

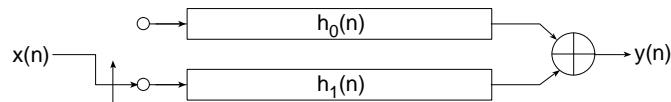


Figure 24: Half-band decimation filter.

The filter is very similar in nature to the polyphase decimator described in 5.5 with the decimation factor set to $M=2$. However, there is a subtle difference in the implementation that makes the half-band decimator a more area efficient 2-to-1 down-sampling filter when the frequency response reflects a true half-band characteristic.

The frequency and time response of a half-band filter are shown in Figure 13 and Figure 14 respectively. Observe the alternating zero-valued coefficients in the impulse response. Figure 25 exposes the details of a 7-tap half-band polyphase filter when the coefficients are allocated to the two polyphase segments $h_0(n)$ and $h_1(n)$ in Figure 24. Figure 25(a) is the filter impulse response, note that $a_1 = 0 = a_5$. Figure 25(b) provides a detailed illustration of the polyphase sub-filters and shows how the filter coefficients are allocated to the two polyphase arms. In the bottom arm, $h_1(n)$, the only non-zero coefficient is the center value of the impulse response a_3 . Figure 25(c) shows the optimized architecture when the redundant multipliers and adders are removed. The final structure has a reduced computation workload in contrast to a more general 2:1 down-sampling filter. The number of multiply-accumulate (MAC) operations required to compute an output sample has been lowered by a factor of approximately two.

The arithmetic optimizations described above are exploited in the Xilinx half-band decimating filter to minimize the logic requirements of the FPGA implementation.

Even though the previous description and associated figures have represented and described the half-band filter in terms of MAC operations, and the signal flow-graphs indicate explicit multiply operations, as with all of the filters discussed in this document, the underlying implementation is done using distributed arithmetic techniques.

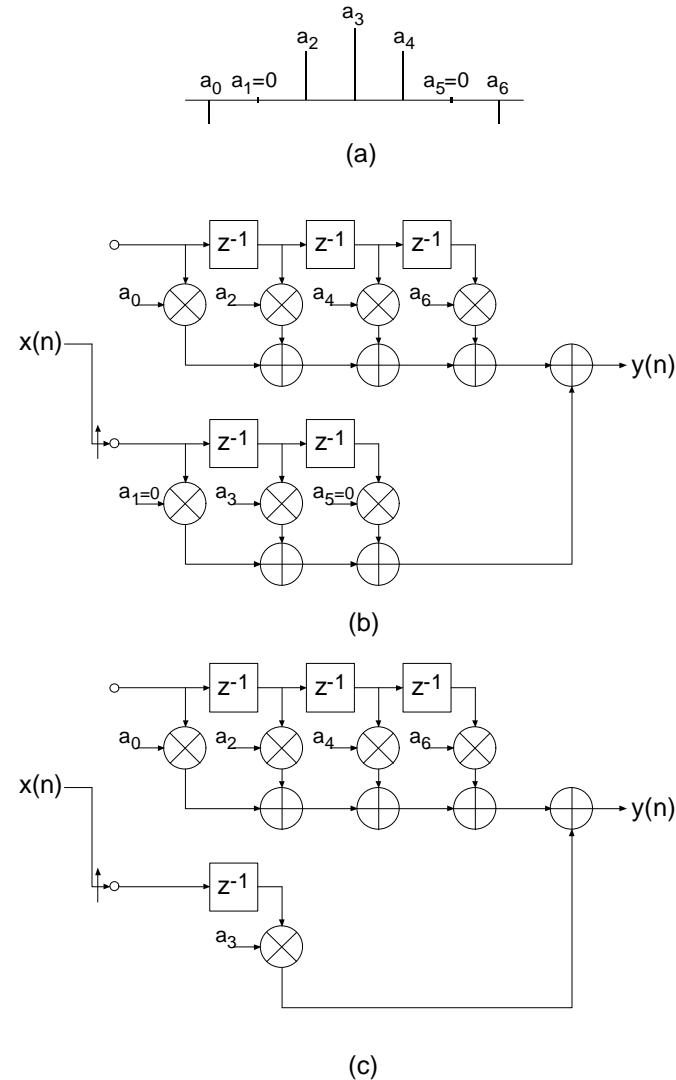


Figure 25: 7-tap half-band decimation filter. (a) Impulse response. (b) Polyphase partition. (c) Reduced complexity (hardware optimized) realization. The high density of zero-valued filter coefficients are exploited in the FPGA realization to produce a minimal area implementation.

5.8 Half-Band Interpolator

Just as the half-band decimator is an optimized version of the more general polyphase decimation filter, the half-band interpolator is a special case of a polyphase interpolator. The half-band interpolator is shown in Figure 26.

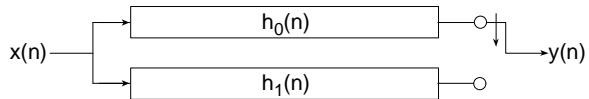


Figure 26: Half-band interpolation filter.

The coefficient set for a true half-band interpolator is identical to that of a half-band decimator with the same specifications. The large number of zero entries in the impulse response is exploited in exactly the same manner as with the half-band decimator to produce hardware optimized half-band interpolators. The process is presented in Figure 27. Figure 27(a) is the impulse response, Figure 27(b) shows the polyphase partition and Figure 27(c) is the optimized architecture that has taken full advantage of the 0 entries in the coefficient data.

Like the polyphase decimator and interpolator, the half-band interpolator only supports single channel input data streams.

5.9 Small Non-Zero Even Terms in a Half-Band Filter Impulse Response

Certain filter design software may result in small non-zero values for the odd terms in the half-band filter impulse response. In this situation it may be useful to force these values to 0 and re-evaluate the frequency response to assess if it is still acceptable for the intended application. If the odd terms are not identically zero the hardware optimizations described above are not possible. If the small non-zero value terms cannot be ignored, the general polyphase decimator or interpolator described in Sections 5.5 and 5.6 respectively using a rate change of two are more appropriate.

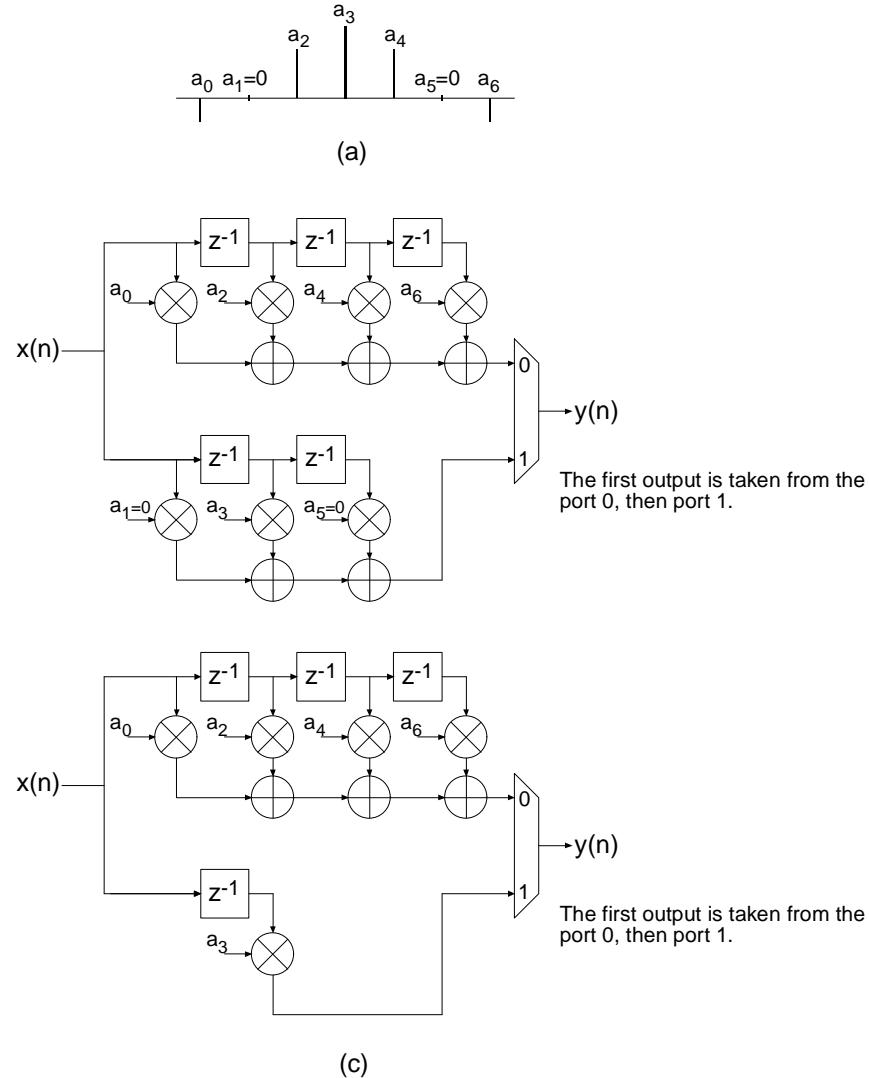


Figure 27: 7-tap half-band interpolation filter. (a) Impulse response. (b) Polyphase partition. (c) Reduced complexity (hardware optimized) realization. The high density of zero-valued filter coefficients are exploited in the FPGA realization to produce a minimal area implementation.

6 On-Line Coefficient Reload

All of the filters provide an interface for loading new coefficient data. While the new coefficient values are being loaded, and some internal data structures are subsequently initialized, the filter ceases to process input samples. The coefficient reload time is a function of the filter length and type.

A high-level view of the reloadable DA FIR architecture is shown in Figure 28. Observe that the DA LUT build engine in addition to resources to store the new coefficient vector (*coefficient buffer*) are integrated with the FIR filter engine.

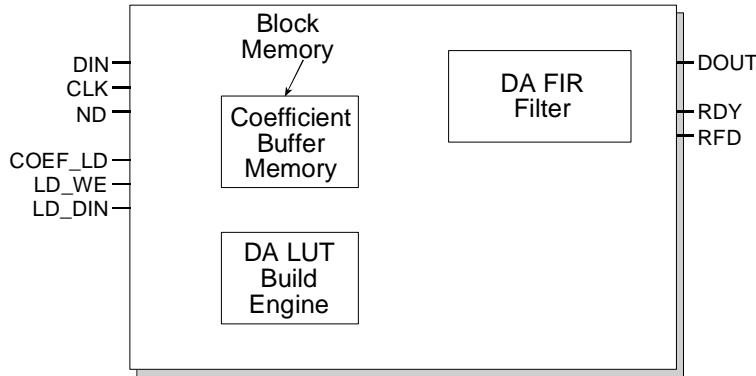


Figure 28: High-level view of DA FIR with reloadable coefficients.

Figure 29 is the symbol for a single-rate FIR supporting coefficient reload. The signals that support the reload operation are *LD_DIN*, *COEF_LD* and *LD_WE*. The *LD_DIN* port is used to supply the new vector of coefficients to the core. *COEF_LD* is asserted to initiate a load operation and *LD_WE* is a write enable signal for the internal coefficient buffer.

When a coefficient load operation is initiated the new vector of coefficients are first written to an internal buffer – the coefficient buffer. Once the load operation has completed, the DA LUT build-engine is automatically started. The build-engine uses the values in the coefficient buffer to re-initialize the DA LUT.

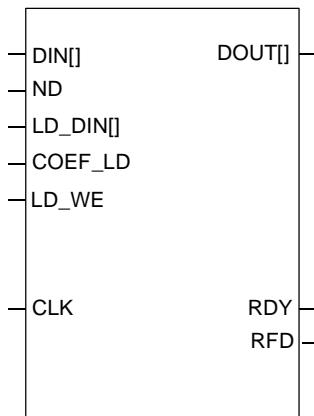


Figure 29: Single-rate FIR filter with coefficient reload functionality.

Figure 30 shows the timing for a coefficient reload operation. *COEF_LD* is asserted to start the procedure. The new vector of coefficients are then written to the internal memory buffer synchronously with the core master clock *CLK*. *LD_WE* may be used to control the flow of coefficient data from the external coefficient source, for example a microprocessor, to the core. *LD_WE* performs a clock-enable function for the load process.

Asserting *COEF_LD* forces *RFD* to the inactive state (low) indicating that the core cannot accept any new input samples. Note, during the reload operation the filter inner-product engine is suspended. Once the new coefficients have been loaded and the DA LUT build engine has constructed the new partial-product lookup tables, *RFD* will be asserted indicating that the core is

ready to accept new input samples and resume normal operation. The filter sample history buffer (regressor vector) is cleared when a new coefficient vector is loaded.

Asserting *COEF_LD* will also force *RDY* to the inactive state (low).

COEF_LD may be re-asserted again at any point during an update procedure (even once the DA LUT build-engine is running) to start a new coefficient configuration.

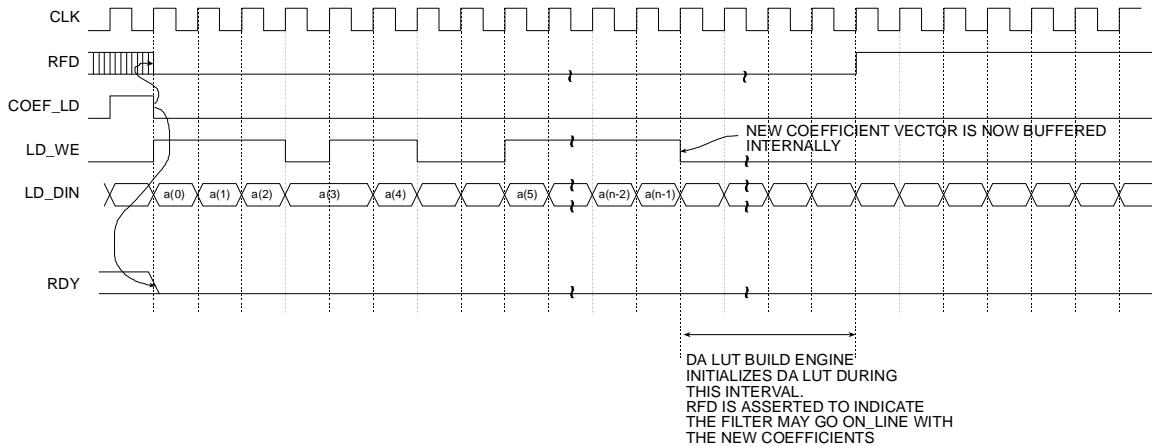


Figure 30: Coefficient reload timing.

Coefficient Reload – Typical Use Model

The typical sequence of events that would occur to engage the coefficient reload would be

1. Pulse *COEF_LD* for a single clock cycle to initiate a coefficient load operation.
2. Supply a length N vector new coefficient data on the *LD_DIN* port. The coefficients can be written to the internal buffer at a rate of one value per clock cycle. The coefficient source may use *LD_WE* to control the rate at which coefficients are delivered. This is useful for systems in which the coefficient source may not be able to accommodate the core bit clock – remember the coefficients are written to the internal buffer synchronously with the core master clock signal *CLK*.
3. Wait until *RFD* is asserted, indicating the filter may now be put back on-line and process input samples with the new coefficient vector.

7 CORE Generator Parameters

A filter core is customized using a configuration wizard. The wizard screens are shown in Figures 31 through 33.

DISTRIBUTED ARITHMETIC FIR FILTER

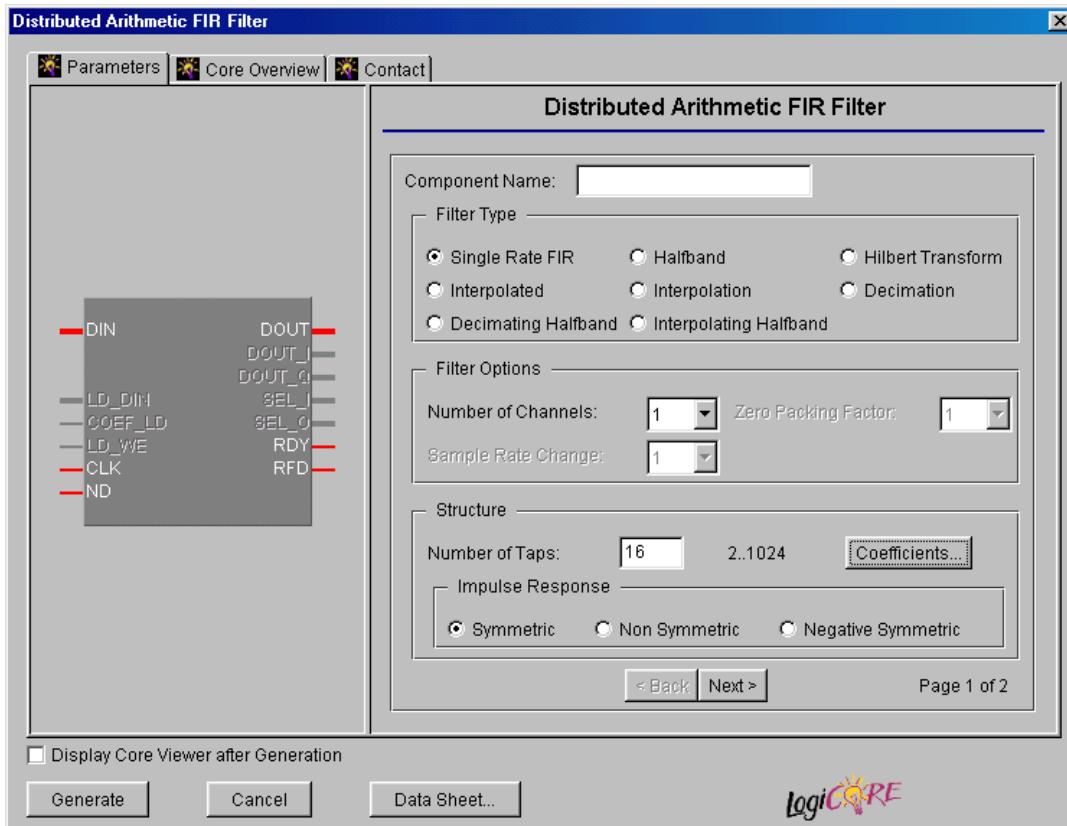


Figure 31: Filter parameterization screen – field 1.

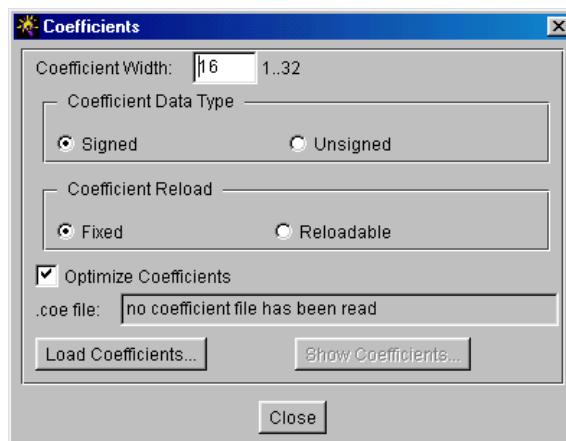


Figure 32: Parameterization screen – field2 or *Coefficients* panel. The coefficient parameterization screen is accessed using the *Coefficients* tab on the primary GUI shown in Figure 31.

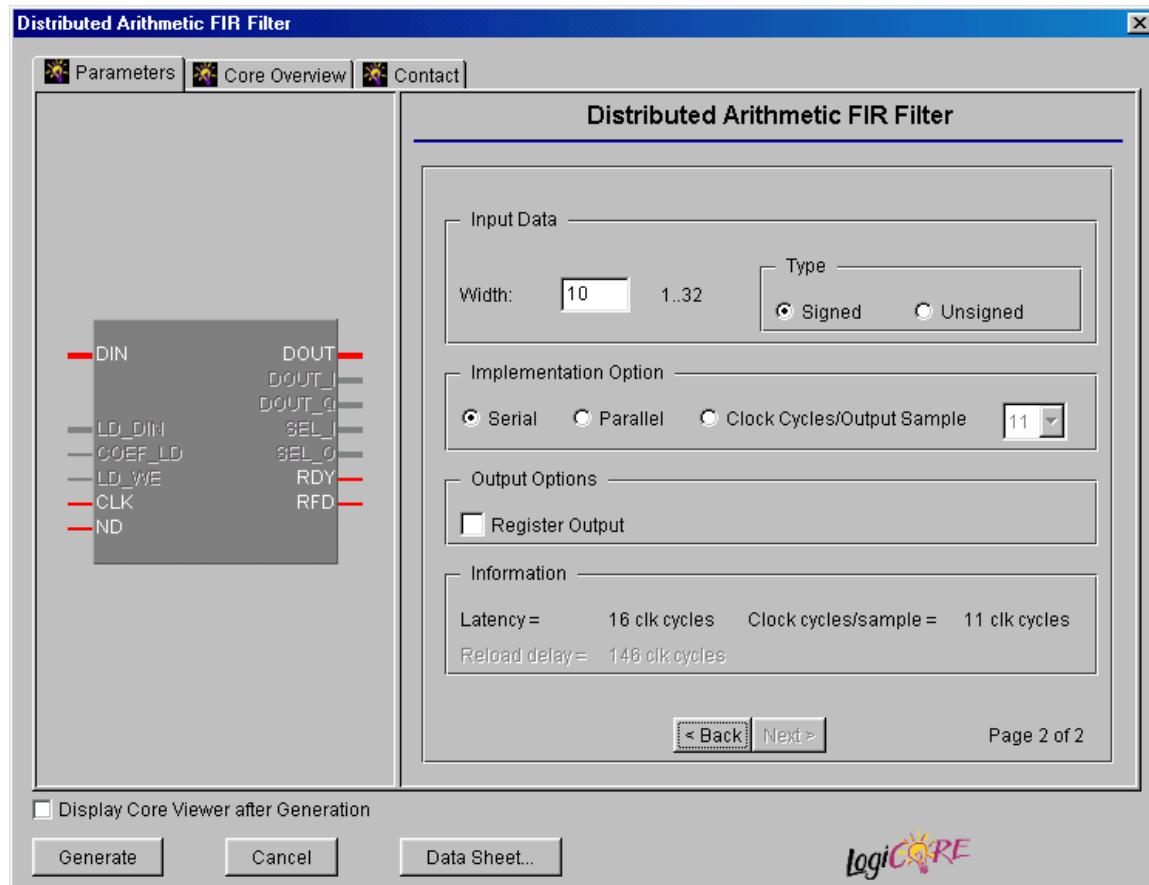


Figure 33: Filter parameterization screen – field 3.

The user supplied parameters are:

- **Component Name:** The user defined filter component name.
- **Filter Type:** Eight filter types are supported 1. Single rate FIR, 2. Half-band FIR, 3. Hilbert transform, 4. Interpolated FIR, 5. Polyphase decimator, 6. Polyphase interpolator, 7. Half-band decimator, and 8. Half-band interpolator.
- **Number of Channels:** The number of channels processed by the filter. One to a maximum of 8 channels can be accommodated by a single filter core. The polyphase decimator and polyphase interpolator provide single channel support only.
- **Zero Packing Factor:** This field is applicable to the *interpolated* filter only. The zero packing factor specifies the number of 0's inserted between the coefficient data supplied by the user in the .coe (filter coefficient file). This is an integer value between 2 and 8 inclusive. A zero packing factor of k will insert $k-1$ 0's between the supplied coefficient values.
- **Sample Rate Change:** This field is applicable to the polyphase decimator and interpolator structures. When the decimator is selected, the *Sample Rate Change* value defines the

decimation factor. For the interpolation filter it defines the up-sampling factor. Sample rate changes of between 1 to 8 inclusive are supported for both up-sampling and down-sampling.

- **Number of Taps:** The number of filter taps. For a symmetric impulse response (either even or odd symmetric) the number of filters taps is between 2 and 1024 inclusive. For a non-symmetrical coefficient set the range is 2 to 1024 inclusive.
- **Coefficient Width:** The bit precision of the coefficient data. This is an integer value between 1 and 32 inclusive. The **Coefficient Width** parameter is accessed using the *Coefficients* user interface (UI) shown in Figure 32. This sub-panel is enabled using the *Coefficients* tab on the primary GUI.
- **Coefficient Data Type:** The coefficient data can be specified as either signed or unsigned. When the signed option is selected conventional two's complement representation is assumed. The **Coefficient Data Type** parameter is accessed using the *Coefficients* user UI shown in Figure 32. This sub-panel is accessed using the *Coefficients* tab on the primary GUI.
- **Coefficient Reload:** When the *Fixed* radio button on the *Coefficient Reload* panel is selected the filter Core will be generated without a coefficient reload interface. When the *Reloadable* button is selected a coefficient reload interface is provided on the Core.
- **Optimize Coefficients:** The look-up tables employed in the filter mechanization can be optimized to minimize the amount of FPGA logic fabric employed by the core. The optimization is data (filter coefficient set) dependent.
- **Load Coefficients:** The filter coefficients are supplied in a coefficient or coe file. This is an ASCII file with a ".coe" extension. The file format is described in detail below. Activating this tab presents a browser window that lets the user select a coefficient file.
- **Show Coefficients:** Selecting this tab on the *Coefficients* panel displays the filter coefficient data.
- **Input Data Width:** The precision (in bits) of the filter input data samples. The input sample precision is an integer value between 1 and 32 inclusive.
- **Input Data Type:** The filter input data can be specified as either signed or unsigned. The signed option employs conventional two's complement arithmetic.
- **Implementation Option:** Selecting the *Serial* option generates an SDA FIR filter. This is a fully serial DA FIR filter. In this case, if a non-symmetric impulse response is specified, B (B is the bit precision of the input data) clock cycles are required to generate a new output sample (B clock cycles per output point). If a symmetric impulse response is employed, $B+1$ clock cycles are required per output point.

If the *Parallel* filter is specified a fully parallel PDA filter is produced. The fully parallel filter produces a new output sample on every clock edge. Choosing the *Clock Cycles/Sample* option allows the degree of filter parallelism to be specified using the associated pull-down menu. The menu presents the valid set of values (L) that can be selected to specify the number of cycles per output sample of the *internal* polyphase filter segments. For example, selecting $L=3$ for a polyphase decimator will result in a filter where each internal DA sub-filter generates a new output sample every 3 clock cycles.

For all of the polyphase filters, including the half-band decimation and half-band interpolation filters, the *Clock Cycles/Sample* value refers to the individual filters that are employed to construct all of the multirate architectures.

- **Impulse Response:** Indicates structure present in the coefficient set. The user may specify a symmetric, negative (odd)-symmetric or non-symmetric impulse response.
- **Output Options:** The filter output bus can be registered or unregistered. When the registered output option is selected, the filter output bus *DOUT* is maintained at the core output between successive assertions of *RDY*. In the unregistered mode the output sample is only valid when *RDY* is active. At other times the port will change on successive clock cycles.
- **Information:** This field reports the filter latency (the number of clock cycles between presenting an input data sample and the corresponding filter output sample) and the number of clock cycles per sample. The filter latency is also available in the component instantiation file. This file has a base-name that is the same as the filter component name, with a *.vho* extension for a VHDL design flow or a *.veo* extension for a Verilog flow. For example, if the filter component name is *my_filt*, and a VHDL flow has been selected, the instantiation file will be named *my_fir.vho*. If a Verilog flow had been selected this file would be called *my_fir.veo*.

8 XCO File Parameters

The parameters supplied via the filter customization wizard are captured and logged to the *.xco* file. The full name of this file is simply the *Component Name* with a *.xco* file extension. Table 2 defines the *.xco* file parameter names and range specifications.

Table 2: XCO file parameter names, definitions and range specifications.

Parameter Name	Definition	Range
BusFormat	Controls the notation employed for identifying buses in the output edif netlist file.	{BusFormatAngleBracket BusFormatParen}
SimulationOutputProducts	Core HDL simulation selection – either VHDL or Verilog.	{VHDL VERILOG}
ViewlogicLibraryAlias	Pathname to Viewlogic directory	Valid path name for the user's operating system.
XilinxFamily	The FPGA target device family.	{Virtex Spartan2}
DesignFlow	HDL flow specifier.	{VHDL VERILOG}
FlowVendor	Design flow vendor information.	{Other Synplicity Exemplar Synopsis Foundation}
coefficient_file	File of filter coefficient values. The file format is defined in Section 10.	Any valid file name for the user's operating system consisting of the letters a...z, 0...9 and '_'.
coefficient_data_type	The filter coefficient data type. When the type <i>signed</i> is selected conventional 2's complement arithmetic is employed.	{signed unsigned}

number_of_taps	The number of filter taps.	[2,...,1024]
register_output	When <i>true</i> , an output register is inserted at the output of the filter datapath. In this case the filter output will remain valid during successive transitions of the filter output port(s). When this parameter is <i>false</i> , the filter output is not registered and the output sample is valid only during the clock cycle demarcated by the <i>RDY</i> control signal.	{true false}
optimize_coefficients	When <i>true</i> , logic optimization is performed on the filter look-up tables. Selecting optimization will result in the most compact (minimum FPGA logic resources) implementation. If this parameter is <i>false</i> , no logic optimization is performed.	{true false}
component_name	Textbox that defines the filter component name.	Any valid file name for the user's operating system consisting of the letters a...z, 0...9 and '_'.
zero_packing_factor	This field is only applicable to <i>interpolated filters</i> and controls the number of 0's inserted between the user supplied coefficient values. A value of 2 results in a single 0 valued entry between the user coefficients, a value of 3 inserts 2 0's between the user coefficients and so on. For all filters other than the <i>interpolated</i> filter this parameter should be 1.	[1,...,8]
impulse_response	This parameter allows the user to identify structure in the filter coefficient data. Coefficient vectors that are identified (explicitly by the user) as being structured (symmetric or negative symmetric) result in minimal size hardware implementations.	{non_symmetric symmetric negative_symmetric}
sample_rate_change	This parameter specifies the sample rate change embedded in the filter. For all single-rate filters the rate change is considered to be 1.	[1,...,8]
number_of_channels	The number of channels supported by the filter. All	[1,...,8]

	multirate filters support only a single channel.	
clock_cycles_per_sample	Number of clock cycles required to generate a filter output sample. In the context of any of the multirate filters, this value is associated with the sub-filters (polyphase segments) and not the final output result.	The valid set of values for this parameter is a function of several other parameters, including <i>input_data_width</i> and <i>impulse_response</i> . This value will be a minimum of 1, corresponding to a full parallel implementation in which a new output sample is available on every clock edge, to a maximum of <i>input_data_width</i> +1.
filter_type	The filter type specifier.	{ single_rate_fir half-band hilbert_transform interpolated interpolation decimation decimating_half-band half-band_interpolating }
coefficient_width	Number of bits used to represent the filter coefficient values.	[1,2,...,32]
input_data_width	Number of bits used to represent the filter input samples.	[1,2,...,32]
implementation_option	This field defines the degree of filter parallelism and is further discussed in Section 2.2.	{clock_cycles_per_output_sample parallel serial}
input_data_type	The filter input sample data type. When the type <i>signed</i> is selected conventional 2's complement arithmetic is employed.	{signed unsigned}
coefficient_reload	This parameter controls the presence (or not) of a coefficient reload interface. When defined as <i>stop_during_reload</i> the interface is included. When defined as <i>fixed_coefficients</i> no coefficient reload feature is present.	{stop_during_reload fixed_coefficients}

Figure 34 is an example .xco file. The '#' characters at the start of the first five lines in the example identify in-line comments.

```
# Xilinx CORE Generator 3.1.01i
# Username = chrisd
# COREGenPath = c:\xilinx\coregen
# ProjectPath = C:\xilinx_projects\projects\FIR\coregen\reload
# ExpandedProjectPath = C:\xilinx_projects\projects\FIR\coregen\reload
SET BusFormat = BusFormatParen
SET SimulationOutputProducts = VHDL
```

```
SET ViewlogicLibraryAlias = ""
SET XilinxFamily = Virtex2
SET DesignFlow = VHDL
SET FlowVendor = Synplicity
SELECT Distributed_Arithmetic_FIR_Filter Virtex2 Xilinx,_Inc. 5.0
CSET coefficient_file = C:\xilinx_projects\projects\FIR\coregen\reload\h.coe
CSET coefficient_data_type = signed
CSET number_of_taps = 16
CSET register_output = false
CSET optimize_coefficients = false
CSET component_name = f
CSET zero_packing_factor = 1
CSET impulse_response = non_symmetric
CSET sample_rate_change = 4
CSET number_of_channels = 1
CSET clock_cycles_per_sample = 10
CSET filter_type = interpolation
CSET coefficient_width = 16
CSET input_data_width = 10
CSET input_data_type = signed
CSET coefficient_reload = stop_during_reload
GENERATE
```

Figure 34: Example .xco File.

9 Interface, Control and Timing

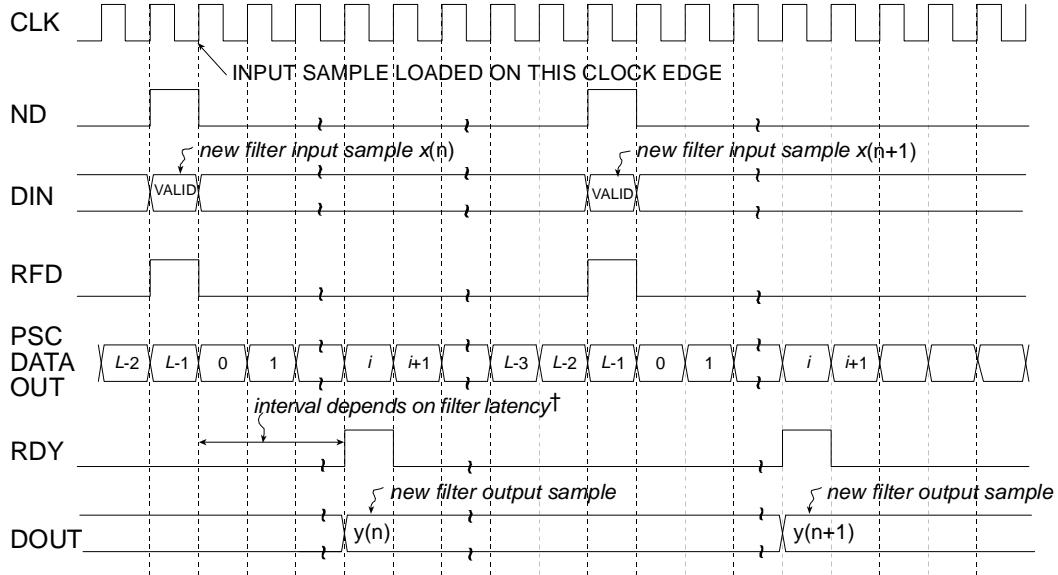
All of the filter classes employ a data-flow style interface for supplying input samples to the core and for reading the filter output port. *ND* (*New Data*), *RFD* (*Read For Data*) and *RDY* (*Ready*) are used to co-ordinate I/O operations. In addition, for multi-channel filters, *SEL_I* and *SEL_O* are supplied to indicate the active input and output stream respectively.

9.1 Nomenclature

In the timing diagrams supplied in this section the notation $x(n)$ and $y(n)$ are used to denote the filter input and output samples respectively. In some diagrams, for space reasons, the variable name (*x* or *y*) has been omitted and the diagram is only annotated with the index value n .

9.2 Timing: Single Rate and Multi-Channel Filters

The timing for a single channel filter, with L clock cycles per output sample and a registered output port is shown in Figure 35. The *ND* input signal is used for loading a new input sample into the filter. It is effectively used internally as a clock enable, and the actual sample load operation occurs on the rising of the clock (*CLK*). When the core is ready to accept a new input sample the *RFD* signal is asserted. When a new output sample is available *RDY* is asserted for a single clock period. When the registered output option is selected the output sample will remain valid between successive assertions of *RDY*.



† The latency is reported on the filter GUI

Figure 35: Single channel FIR filter timing. L clock cycles per output sample, registered output.

Figure 36 shows the timing for a single channel filter with an unregistered output port. The input timing is the same as for the registered output example, but now the filter result is valid for only a single clock period and is framed by RDY.

For the two cases described so far, the host system is supplying input samples at the highest frequency possible, that is, every L clock ticks. This does not have to be the case, data samples can be supplied at a lower rate without disturbing the operation of the filter as shown in Figure 37. In this example, even though the filter has been designed specifying L clock cycles per output sample, new data is supplied to the filter every $L+2$ clock periods. Observe that *RFD* is still asserted on the L th clock cycle of a data sample epoch, but the host system only supplies a new input sample 2 clock cycles later. *RFD* remains active until the new input sample has been accepted by the filter core. This occurs synchronously with the positive going edge of the clock and with *ND* effectively acting as an active high clock enable.

DISTRIBUTED ARITHMETIC FIR FILTER

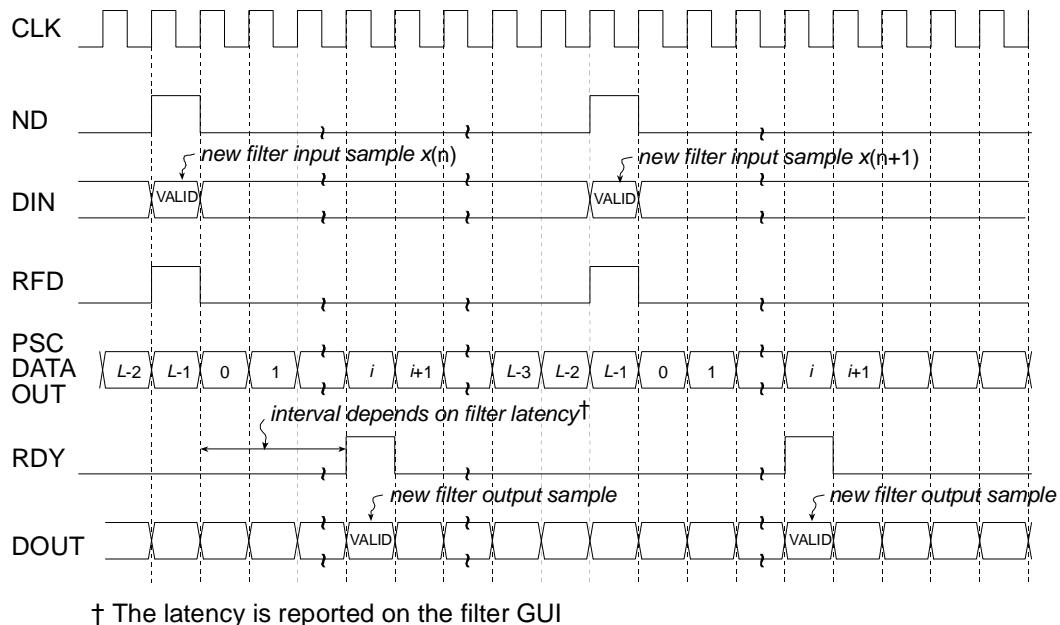


Figure 36: Single channel FIR filter timing. L clock cycles per output sample, unregistered output.

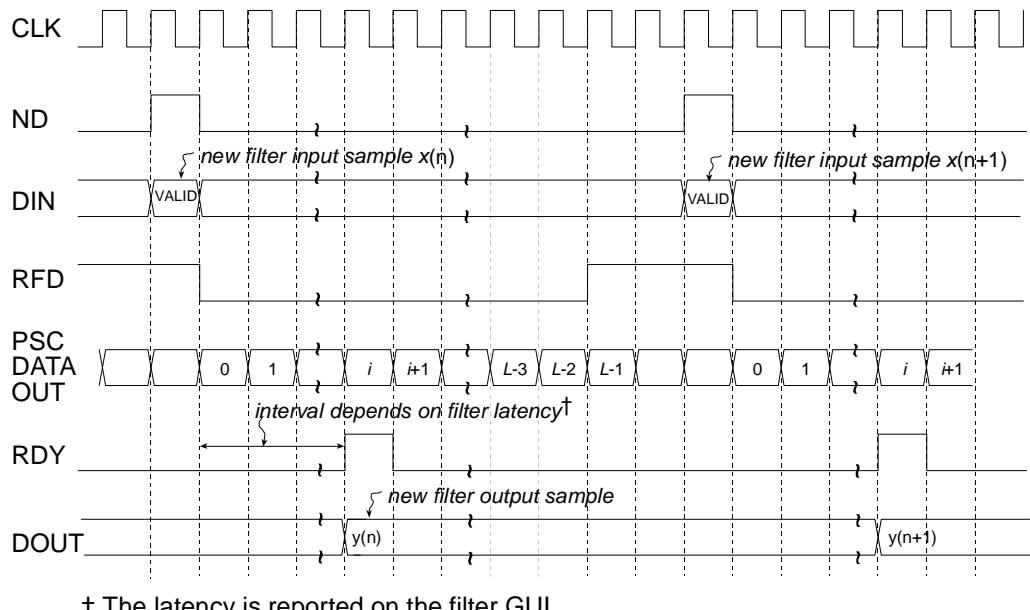


Figure 37: Single channel FIR filter timing. L clock cycles per output sample, registered output. Input samples supplied every $L+2$ clock periods.

As a specific example of the filter interface timing, consider a non-symmetric single-channel FIR filter with 10-bit precision input samples and a full serial realization ($L=10$). The timing diagram is shown in Figure 38. Ten clock cycles are needed to process each new input sample.

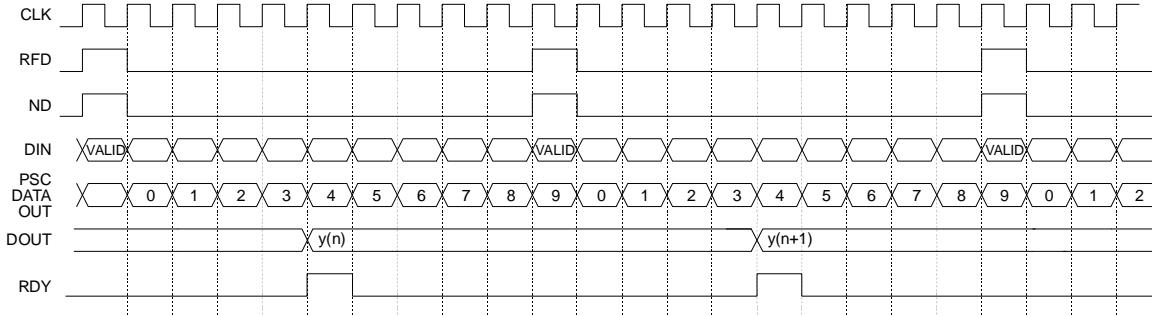


Figure 38: Single channel FIR filter timing. Full serial implementation, 10-bit input samples, registered output. For $L=10$, there are 10 clock periods between successive output samples.

A symmetrical filter with B -bit precision input samples requires in general $B+1$ clock periods for a full serial (SDA) implementation. Figure 39 shows the timing for a single channel symmetrical FIR employing 10-bit input samples. In this case, eleven clock cycles ($L=11$) are required to process each new piece of data.

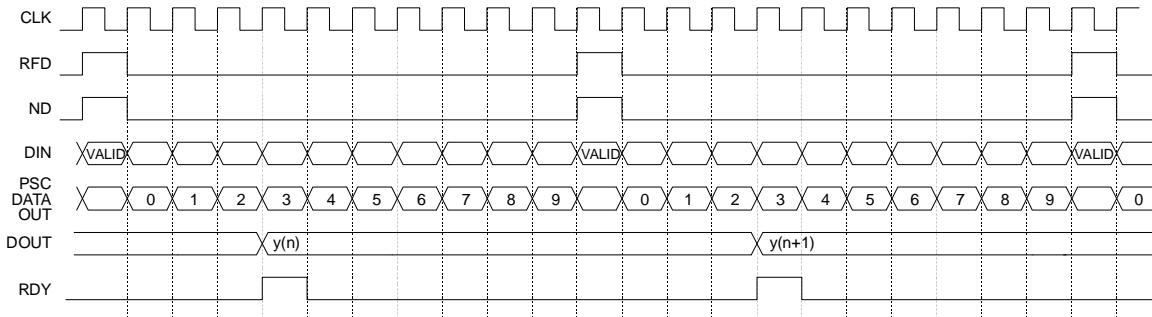


Figure 39: Single channel FIR filter timing. Full serial implementation, 10-bit input samples, symmetrical impulse response, registered output. 11 clock periods are required to process each new input sample.

The previous two figures illustrated the timing for full serial, or SDA filter implementations with symmetrical and non-symmetrical coefficient data. The Core Generator filter core supports various types of parallel filter realizations. The greater the degree of filter parallelism employed, the higher the filter sample rate. Filter parallelism is specified in terms of the number of clock cycles (L) required to compute an output sample. This value is accessed via the filter core GUI when the *Multi clock cycles per output sample* is selected in the *Implementation Option* field. The associated drop-down menu indicates valid options for L . The valid options for L depend on the filter parameters – symmetrical/non-symmetrical coefficient data and precision of the input samples. For example, for an input sample precision $B=10$ and using a non-symmetrical impulse response, the valid values for L are $\{1, 2, 3, 4, 5, 10\}$. For $B=10$ and a symmetrical impulse response $L=\{1, 2, 3, 4, 6, 11\}$.

Figure 40, Figure 41 and Figure 42 illustrate the timing diagrams for a filter with $B=10$ bit precision input samples, with $L=2, 4$ and 6 respectively.

DISTRIBUTED ARITHMETIC FIR FILTER

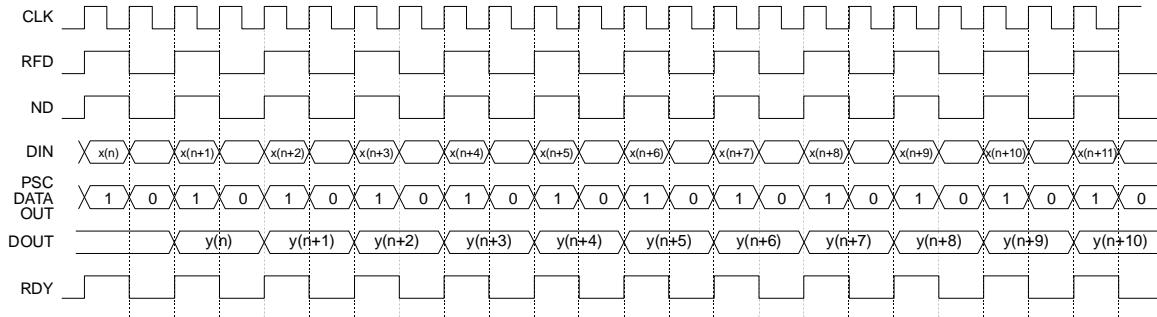


Figure 40: Single channel FIR filter timing. PDA FIR with $B=10$ -bit input samples, $L=2$ clock cycles per output sample, registered output. There are 2 clock periods between successive output samples.

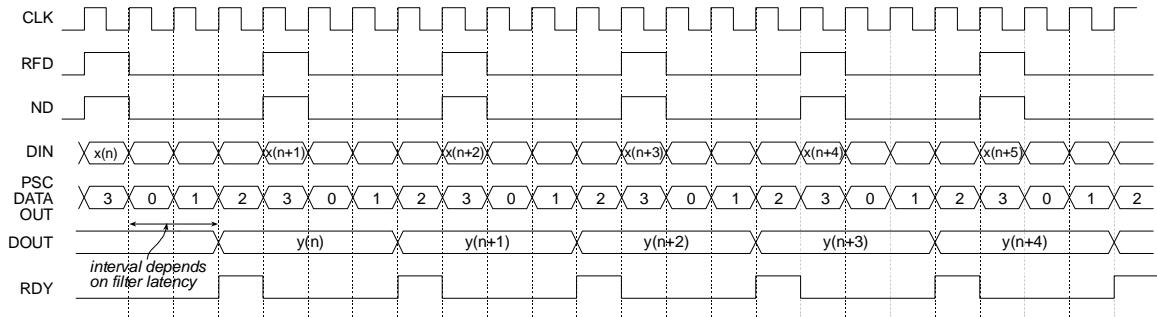


Figure 41: Single channel FIR filter timing. PDA FIR with $B=10$ -bit input samples, $L=4$ clock cycles per output sample, registered output. There are 4 clock periods between successive output samples.

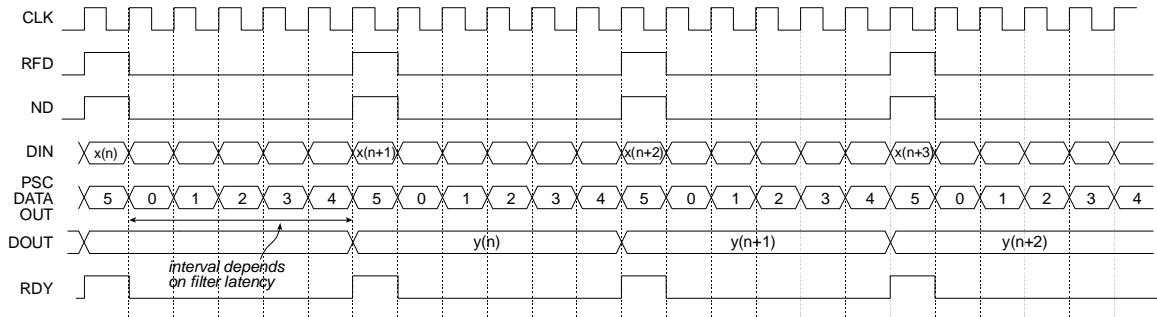


Figure 42: Single channel FIR filter timing. Symmetrical PDA FIR with $B=10$ -bit input samples, $L=6$ clock cycles per output sample, registered output. There are 6 clock periods between successive output samples.

Figure 43 illustrates the filter timing for a fully parallel DA (PDA) FIR filter. Observe that after the initial start-up latency a new output sample is available on every clock edge. The number of clock cycles in the start-up latency period is a function of the filter parameters. This value is reported in the filter design GUI in addition to the associated .vho (or .veo, refer to Section 8) file.

The figure shows *ND* valid on every clock edge – so a new input sample is delivered to the filter on each clock edge. Of course, *ND* may be removed for an arbitrary number of clock cycles in order to temporarily suspend the filter operation. No internal state information is lost when this is done, and the filter will resume normal operation when *ND* is re-applied (placed in the active again).

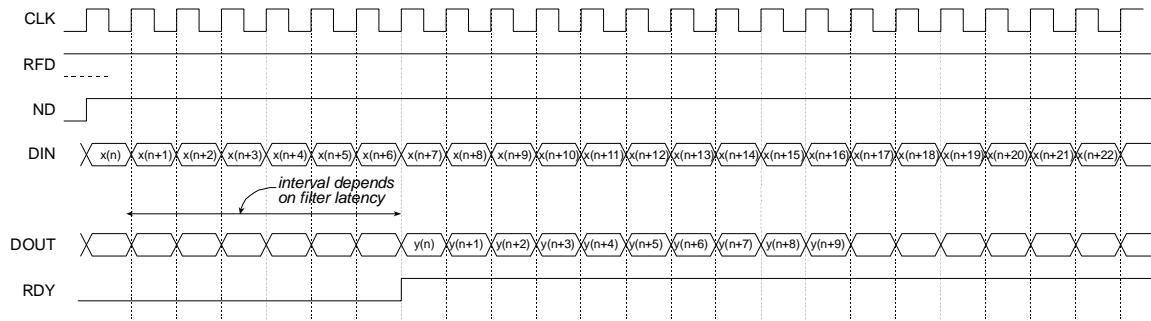


Figure 43: Fully parallel implementation. Single channel filter. With a fully parallel implementation a new output sample is available on each clock edge (after the start-up latency) independent of the filter length or the bit precision of the input data samples.

Figure 44 and Figure 45 demonstrate the timing for a multi-channel filter. Multi-channel filters provide two additional output ports, *SEL_I* and *SEL_O*, that indicate the active input and output channel respectively. Figure 44 illustrates a filter with an unregistered output while Figure 45 shows the timing for registered output samples.

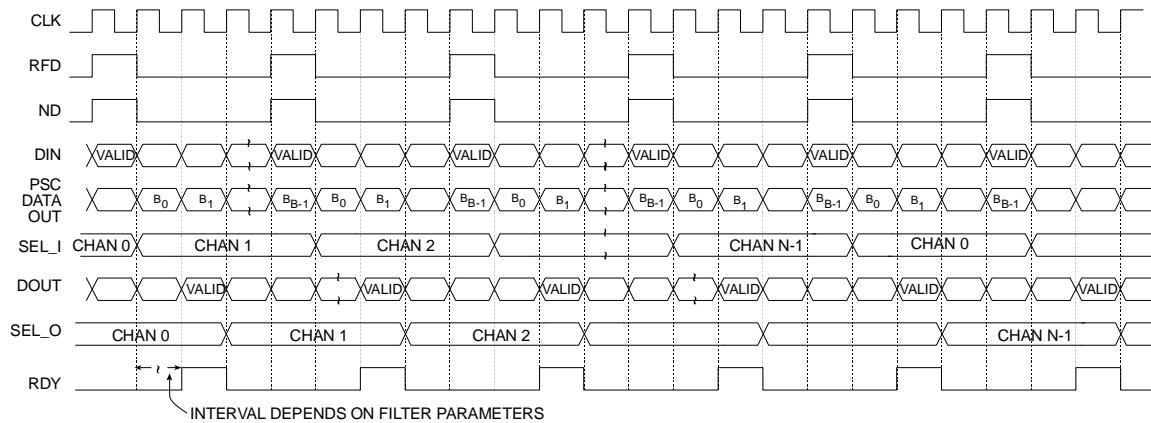


Figure 44: Multi-channel FIR filter timing. Non-symmetrical impulse response, *B*-bit input samples, unregistered output.

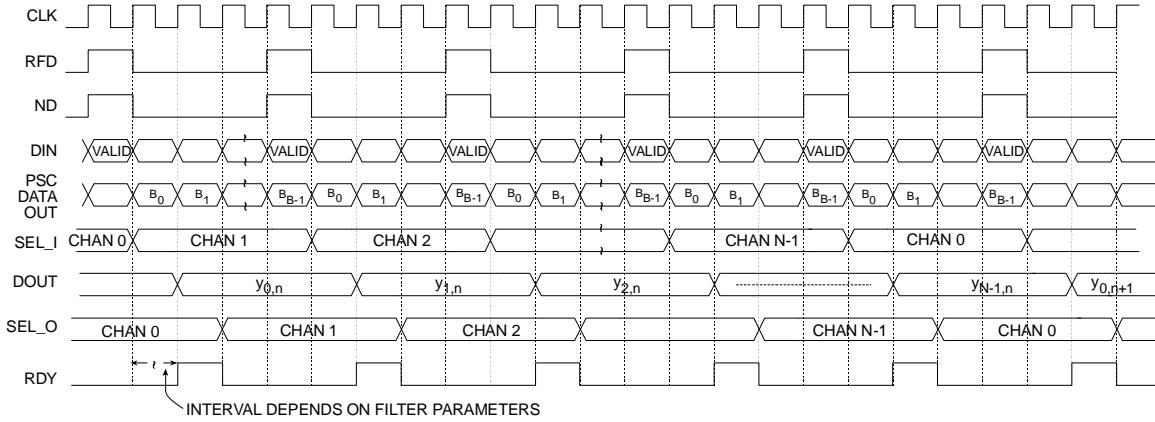


Figure 45: Multi-channel FIR filter timing. Non-symmetrical impulse response, B -bit input samples, registered output.

9.3 Polyphase Decimator Timing

Figure 46 demonstrates the timing for a polyphase decimator with $M = 4, B = 8$ and 8 clock cycles per output point (*Clock Cycles/Output Sample=8*). Remember, for all of the multirate filter structures, the number of clock cycles per output point specification (*Clock Cycles/Output Sample*) refers to the individual filter segments that comprise the filter, and is not directly associated with the filter output port *DOUT*. Observe that in this case, the filter is always able to accept input samples, as indicated by *RFD=1*. New output samples become available after M , in this case 4, input samples have been delivered to the filter. New output samples are produced in response to each new block of 4 input values. Delivering the final value in each M -tuple causes a new inner product calculation to commence. The resulting output sample becomes available a number of clock cycles k after the final sample in the M -tuple is delivered. The exact value of k is a function of the filter parameterization. It is tightly coupled to the input sample bit precision, the value specified for the *Clock Cycles/Output Sample* parameter, in addition to the number of internal pipeline stages and the data buffering depth in the filter. It is always recommended to use the output control signal *RDY* to coordinate all processes that are data sinks for the filter output port *DOUT*.

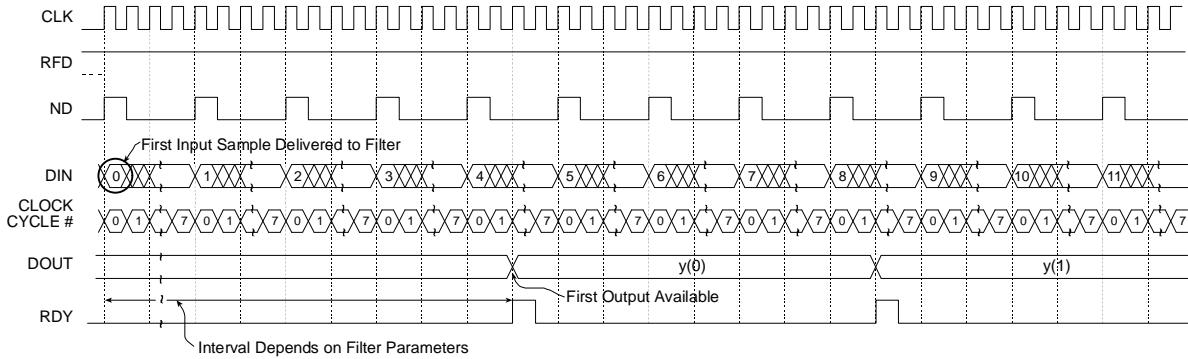


Figure 46: Polyphase decimator timing. 8-bit precision input samples, down-sampling factor $M=4, L=8$.

Figure 47 illustrates the timing for a 4-to-1 polyphase decimator with similar parameters to the filter considered in Figure 46, but in this case the number of *Clock Cycles/Output Sample* is $L=4$. Observe that even though the input sample precision ($B=8$) is the same as in the filter demonstrated in Figure 46, samples can be presented to filter every 4 clock cycles, in contrast to every 8 clock periods in the previous example. The filter supports double the input sample rate, and hence twice the bandwidth, of the filter with $L=8$.

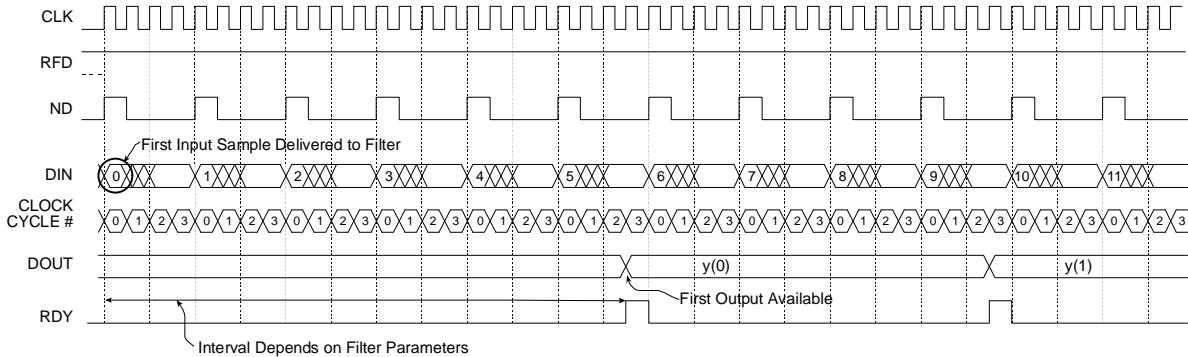


Figure 47: Polyphase decimator timing. 8-bit precision input samples, down-sampling factor $M=4, L=4$.

9.3.1 Polyphase Decimator – Burst Input Mode

Internal buffering in the polyphase decimator allows the user to burst samples into the *DIN* port. This is illustrated in Figure 48 for a down-sampling factor $M=4$, 12-bit input samples and $L=12$. This figure shows the timing for the filter starting from rest, that is, no data has previously been applied to the input port. Notice in this case that a total of 8 samples may be written to the filter before the device removes *RFD*.

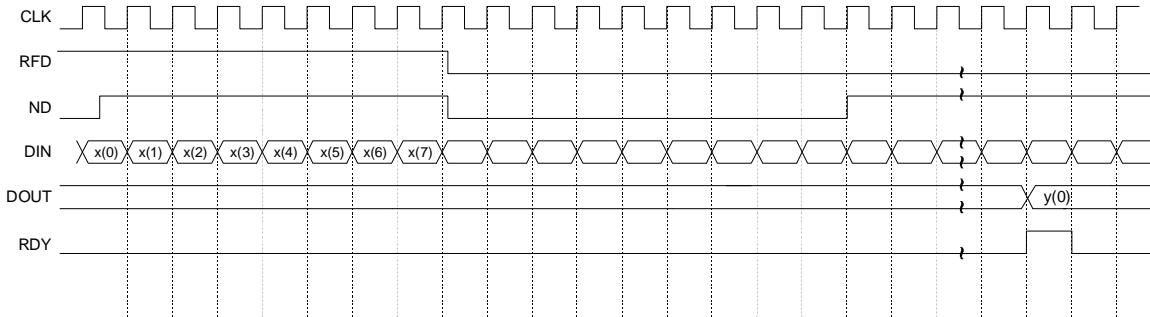


Figure 48: Polyphase decimator timing. 12-bit precision input samples, down-sampling factor $M=4$, $L=12$. Burst input data operation. Diagram shows the timing when the filter is started from rest, that is, no data has previously been applied to the input port.

Once the filter has moved out of this start-up state input samples must obey the timing diagram shown in Figure 49. Only 4 samples can be supplied in each data burst.

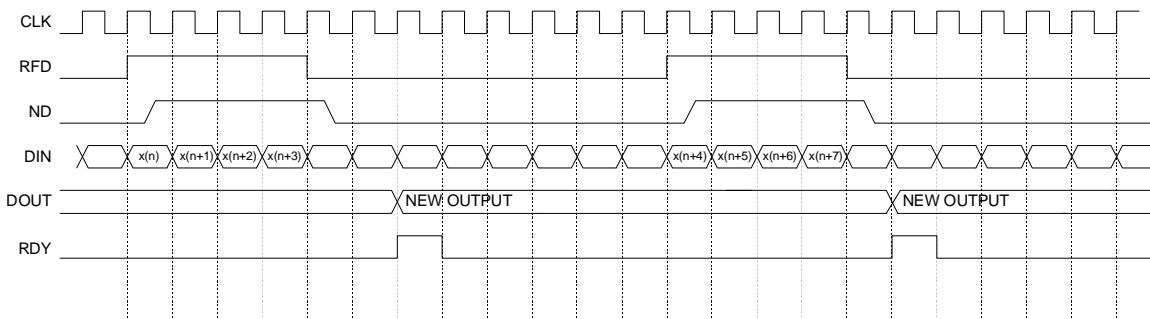


Figure 49: Polyphase decimator timing. 12-bit precision input samples, down-sampling factor $M=4$, $L=12$. Burst input data operation. Diagram shows the timing after the filter has moved out of the start-up timing shown in Figure 48.

As with the *Clock Cycles/Output Sample* parameter for the single-rate filters, this parameter can be used with all the multirate filters to tradeoff performance with silicon area.

9.4 Polyphase Interpolator Timing

Figure 50 shows the timing for a polyphase interpolator that supports a sample rate change of $P=4$, eight bit precision input samples ($B=8$) and 8 clock-cycles-per-output-point. Again, just like the polyphase decimator, the number of clock cycles specified per output point is associated with the individual sub-filters in the polyphase structure. In this example, each subfilter produces a new output sample every 8 clock cycles. The 4 polyphase segments are actually operating concurrently, so in fact, internal to the filter, 4 new output samples are available every 8 clock cycles. When the new block of output samples are available, they are sequenced to the filter output port *DOUT* using an internal multiplexor. The multiplexor select signal is referenced to the filter master clock signal *CLK*. As shown in Figure 50, the vector of P output samples is validated by the core output control signal *RDY*.

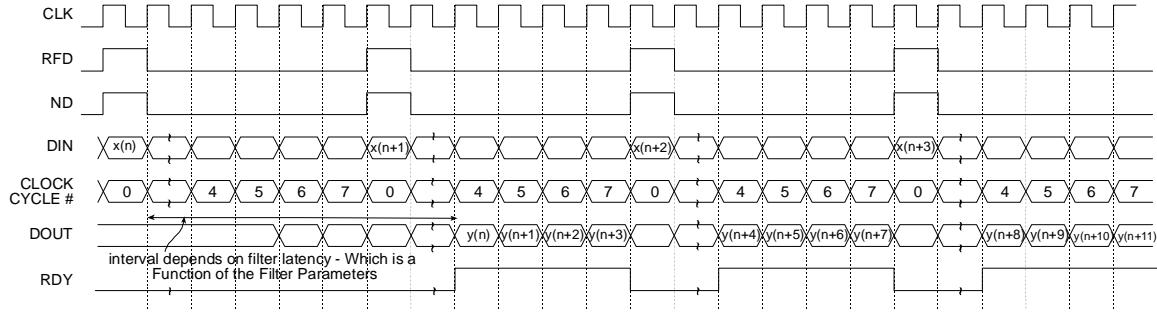


Figure 50: Polyphase interpolator timing. 8-bit precision input samples, up-sampling factor $P=4$. $L=8$.

Figure 51 shows the timing for an interpolator with similar parameters to the example demonstrated in Figure 50, but in this case a value of $L=4$ has been used. This means that each polyphase segment produces a new output sample every 4 clock cycles. In addition, all 4 outputs become available (internally) in parallel. Observe that after the initial startup latency a new interpolant is available at the filter output port $DOUT$ on each successive rising edge of the clock .

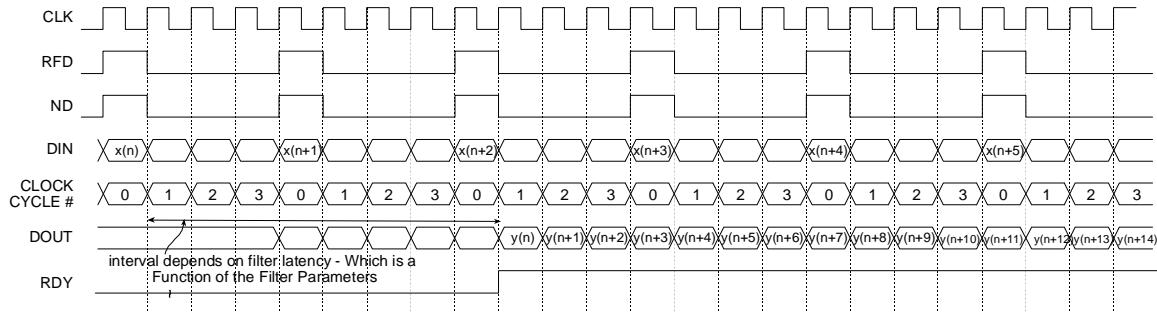


Figure 51: Polyphase interpolator timing. 8-bit precision input samples, up-sampling factor $P=4$. $L=4$.

10 Filter Coefficient Data

The filter coefficients are supplied to the filter compiler using a coefficient file with a .coe extension. This is an ASCII text file with a single line header that defines the radix of the number representation used for the coefficient data, followed by the coefficient values themselves. This is shown in Figure 52 for an N -tap filter.

```
radix=coefficient_radix;
coefdata=
a(0),
a(1),
a(2),
....
a(N-1);
```

Figure 52: Filter coefficient file format.

The filter coefficients must be supplied as integers in either base-10, base-16 or base-2 representation. This corresponds to `coefficient_radix=10`, `coefficient_radix=16` and `coefficient_radix=2` respectively.

The coefficient values may also be placed on a single line as shown in Figure 53.

```
radix=coefficient_radix;  
coefdata=a(0),a(1),a(2),...,a(N-1);
```

Figure 53: Filter coefficient file format – coefficient data on a single line.

The coefficient file format for each of the filter classes supported by the core are discussed below.

10.1 FIR

The coefficient file for the single-rate FIR filter is straightforward and consists of a one-line header followed by the filter coefficient data. For example, the filter coefficient file for an 8-tap filter using a base-10 representation for the coefficient values is shown in Figure 54.

```
radix=10;  
coefdata=20,-256,200,255,255,200,-256,20;
```

Figure 54: Filter coefficient file – 8-tap filter, base-10 coefficient values.

Irrespective of the filter possessing positive or negative symmetry, the coefficient file should contain the complete set of coefficient values. The filter coefficient file for the non-symmetric impulse response shown in Figure 55 is presented in Figure 56.

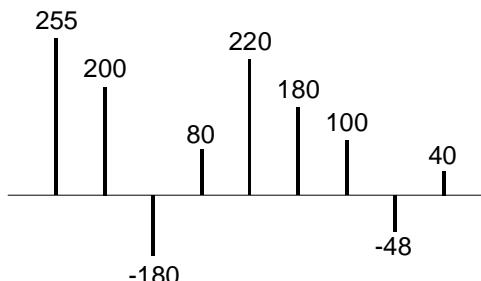


Figure 55: Non-symmetric impulse response.

```
radix=10;  
coefdata=255,200,-180,80,220,180,100,-48,40;
```

Figure 56: Coefficient file for the non-symmetric impulse response in Figure 55.

The coefficient file for the negative-symmetric filter characterized by the impulse response in Figure 57 is shown in Figure 58.

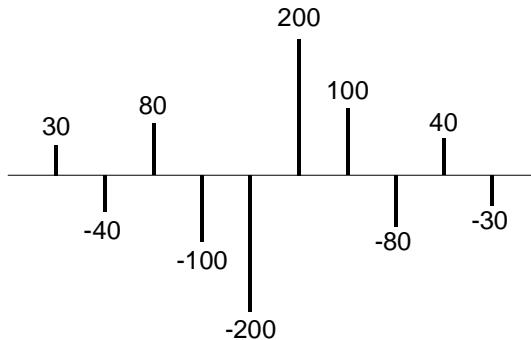


Figure 57: Symmetric impulse response.

```
radix=10;  
coefdata=30,-40,80,-100,-200,200,100,-80,40,-30;
```

Figure 58: Coefficient file for the symmetric impulse response in Figure 57.

10.2 Half-Band Filter

As described in a previous section, every second filter coefficient for a half-band filter with an odd number of terms will be zero. When specifying the filter coefficient data for this filter class, the zero value entries need to be included in the coefficient file. For example, the filter coefficient file that specifies the filter impulse response in Figure 59 is shown in Figure 60.

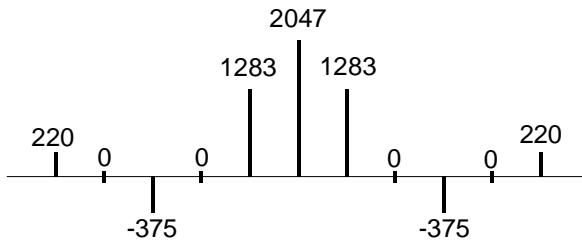


Figure 59: 11-tap half-band filter impulse response.

```
radix=10;  
coefdata=220,0,-375,0,1283,2047,1283,0,-375,0,220;
```

Figure 60: Coefficient file for the half-band filter impulse response shown in Figure 59.

The filter coefficient set is parsed by the filter compiler. If either the alternating zero entries are absent, or the coefficient set is not even-symmetric, this will be flagged as an error and the filter

will not be generated. A dialog box will be presented to indicate the nature of the problem under these circumstances.

Technically, the zero-valued entries for a half-band filter can occur at the filter impulse response extremities as shown in Figure 61. However, observe that these values do not contribute to the result.

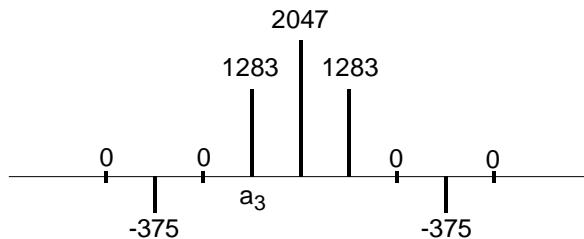


Figure 61: 9-tap half-band filter impulse response.

This condition is detected when the filter is specified. If the number of taps is such that the zero-valued coefficients form the first and last entry of the impulse response, the filter length is reported as an invalid value. The number of taps N for a half-band filter must obey $N = 3 + 4n$, where $n=0,1,2,3,\dots$. For example, a half-band filter may have 11, 15, 19 and 23 taps, but not 9, 13, 17 or 21 taps.

10.3 Hilbert Transform

The impulse response for a 10-term approximation to a Hilbert transformer is shown in Figure 62. The odd-symmetry and zero-valued coefficients are both exploited to generate an efficient FPGA realization. The coefficient data file for the Hilbert transform must contain the zero-valued entries. For example, the .coe file corresponding to Figure 62 is shown in Figure 63.

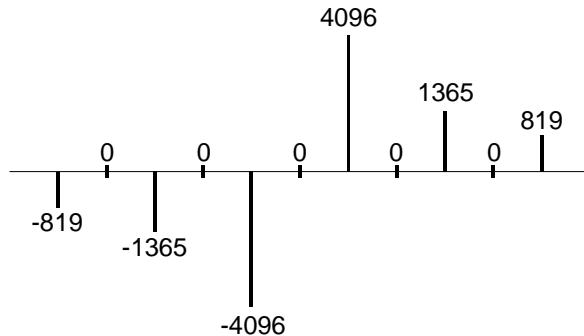


Figure 62: Hilbert transform – impulse response.

```
radix=10;  
coefdata=-819,0,-1365,0,-4096 ,0,4096 ,0,1365,0,819;
```

Figure 63: Coefficient file for the Hilbert transformer with the impulse response shown in Figure 62.

In practice, some optimization methods used for designing a Hilbert transform may lead to the presence of small even-numbered coefficients. If the *Hilbert Transform* filter class is used in the filter compiler, these terms must be forced to zero by the user.

Just like the half-band filter, the zero-valued entries for a Hilbert transformer can occur at the filter impulse response extremities. However, these values do not contribute to the result.

This condition is detected when the filter is specified. If the number of taps is such that the zero-valued coefficients form the first and last entry of the impulse response, the filter length is reported as an invalid value. The number of taps N for a Hilbert transformer must obey $N = 3 + 4n$, where $n=0,1,2,3,\dots$. For example, a Hilbert transform filter may have 11, 15, 19 and 23 taps, but not 9, 13, 17 or 21 taps.

10.4 Interpolated Filter

In a previous section it was explained that an IFIR filter is similar to a conventional FIR, but with the unit delay operator replaced by $k-1$ units of delay. k is referred to as the *zero-packing factor*. One way to realize this substitution is by the insertion of $k-1$ zeros between the coefficient values of a prototype filter. When specifying an IFIR architecture, the full set of prototype coefficients are supplied in the coefficient file, without the zeros implied by the zero-packing factor. The zero-packing factor is defined through the filter user interface. For example, consider the filter coefficient data in the .coe file shown in Figure 64.

```
radix=10;  
coefdata=-200,1200,2047,1200,-200;
```

Figure 64: Prototype coefficient data for IFIR example.

If a zero-packing factor of $k=2$ is specified, the equivalent filter impulse response will be as shown in Figure 65.

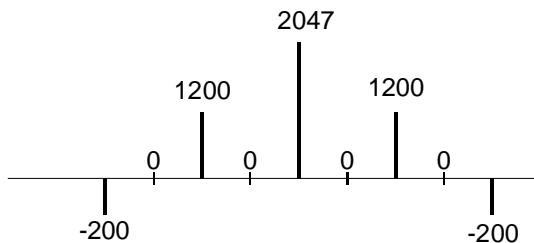


Figure 65: Equivalent IFIR impulse response for the coefficient data shown in Figure 64 with a zero-packing factor $k=2$.

If the zero-packing factor is changed to $k=3$, the impulse response will be as shown in Figure 66.

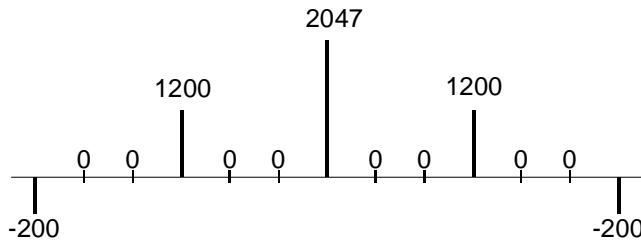


Figure 66: Equivalent IFIR impulse response for the coefficient data shown in Figure 64 with a zero-packing factor $k=3$.

These examples have utilized a symmetrical prototype impulse response, this is not a restriction of the filter core. The prototype filter coefficient set can be symmetrical, non-symmetrical or negative symmetric.

11 Core Resource Utilization

The logic utilization for a filter is a function of the filter length, coefficient precision, coefficient symmetry and input data precision. Table 2 through Table 8 provide logic resource requirements for a number of filter configurations.

Table 3: Virtex logic slice utilization for several filter FIR filter configurations. 10-bit filter coefficients. Filter coefficient optimization is off. Single channel. Signed input, signed coefficients, unregistered output.

Filter Length	Symmetry	Input Sample Precision				
		4-bit	8-bit	12-bit	16-bit	32-bit
4	Symmetrical	31	34	41	43	66
	Non-symmetrical	29	33	36	43	67
8	Symmetrical	36	38	44	49	72
	Non-symmetrical	45	50	53	60	82
32	Symmetrical	103	108	113	117	157
	Non-symmetrical	141	146	151	154	196
80	Symmetrical	247	251	255	261	332
	Non-symmetrical	363	369	373	376	454
128	Symmetrical	370	377	380	385	493
	Non-symmetrical	532	536	537	543	646
256	Symmetrical	731	747	740	749	940

Table 4: Virtex logic slice utilization for several filter FIR filter configurations. 12-bit filter coefficients. Filter coefficient optimization is off. Single channel. Signed input, signed coefficients, unregistered output.

Filter Length	Symmetry	Input Sample Precision				
		4-bit	8-bit	12-bit	16-bit	32-bit
4	Symmetrical	34	35	41	47	69
	Non-symmetrical	30	35	39	45	66
8	Symmetrical	36	41	45	52	75
	Non-symmetrical	50	53	56	62	87
32	Symmetrical	111	114	118	125	166
	Non-symmetrical	160	161	168	173	214
80	Symmetrical	268	273	277	279	353
	Non-symmetrical	408	414	413	424	498
128	Symmetrical	402	415	417	421	521
	Non-symmetrical	595	601	599	607	718
256	Symmetrical	797	806	819	810	1003

Table 5: Virtex logic slice utilization for several half-band filter configurations. 14-bit filter coefficients. Filter coefficient optimization is off. Single channel. Signed input, signed coefficients, unregistered output.

Filter Length	Symmetry	Input Sample Precision				
		4-bit	8-bit	12-bit	16-bit	32-bit
7	Symmetrical	38	42	47	53	77
31	Symmetrical	84	96	100	104	147
79	Symmetrical	171	194	203	206	274

Table 6: Virtex logic slice utilization for several Hilbert transformer configurations. 14-bit filter coefficients. Filter coefficient optimization is off. Single channel. Signed input, signed coefficients, unregistered output.

Filter Length	Symmetry	Input Sample Precision				
		4-bit	8-bit	12-bit	16-bit	32-bit
7	Odd symmetric	41	49	57	66	99
31	Odd symmetric	75	88	96	104	157
79	Odd symmetric	158	187	198	204	289

Table 7: Virtex logic slice utilization for several interpolated filter configurations. 16-bit filter coefficients. Filter coefficient optimization is off. Single channel. Signed input, signed coefficients, unregistered output. Zero packing factor is 4.

Filter Length	Symmetry	Input Sample Precision				
		4-bit	8-bit	12-bit	16-bit	32-bit
8	Symmetrical	44	54	63	69	107
	Non-symmetrical	56	66	71	84	122
32	Symmetrical	146	170	198	201	303
	Non-symmetrical	189	214	239	264	366
80	Symmetrical	359	410	474	477	705
	Non-symmetrical	488	550	609	668	897

Table 8: Virtex logic slice utilization for several PDA FIR filter configurations. 12-bit filter coefficients. 12-bit input data, 60-taps. Filter coefficient optimization is off. Single channel. Signed input, signed coefficients, unregistered output, non-symmetrical impulse response. Filter master clock frequency is 150 MHz.

Number of Clock Cycles per Output Sample	Slice Count	Filter Sample Rate [†] (MHz)
1	3072	150
2	1571	75
3	994	50
4	802	37.5
6	511	25
12	268	12.5

† The filter sample rate is *not* at all dependent on the number of filter taps.

12 Ordering Information

This core is downloadable free of charge from the Xilinx IP Center (www.xilinx.com/ipcenter), for use with version 3.1i and later versions of the Xilinx Core Generator System. The Core Generator System is bundled with the Alliance and Foundation implementation tools.

To order Xilinx software contact your local Xilinx sales representative. For information on the Xilinx sales office nearest you, please refer to:

<http://www.xilinx.com/company/sales.htm>

13 Reference

- [1] Peled and B. Liu, "A New Hardware Realization of Digital Filters", *IEEE Trans. on Acoust., Speech, Signal Processing*, vol. ASSP-22, pp. 456-462, Dec. 1974.
- [2] S. A. White, ``Applications of Distributed Arithmetic to Digital Signal Processing'', *IEEE ASSP Magazine*, Vol. 6(3), pp. 4-19, July 1989.
- [3] Xilinx Inc., *Xilinx Product Guide*, Xilinx Inc., San Jose California, 1999.
- [4] P.P. Vaidyanathan, *Multirate Systems and Filter Banks*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.
- [5] M. E. Frerking, *Digital Signal Processing in Communication Systems*, Van Nostrand Reinhold, New York, 1994.
- [6] C. H. Dick, "Implementing Area Optimized Narrow-Band FIR Filters Using Xilinx FPGAs", *SPIE International Symposium on Voice, Video and Data Communications – Configurable Computing: Technology an Applications Stream*, Boston, Massachusetts USA, pp. 227-238, Nov 1-6, 1998. Also available at: <http://www.xilinx.com/products/logicore/coredocs.htm>

March 2, 2001

Product Specification



Xilinx Inc.
 2100 Logic Drive
 San Jose, CA 95124
 Phone: +1 408-559-7778
 Fax: +1 408-559-7114
 URL: www.xilinx.com/ipcenter
 Support: support.xilinx.com

Features

- Drop-in module for Virtex™, Virtex™-II, Virtex™-E and Spartan™-II FPGAs
- Generates Adder, Subracter and Adder/Subracter
- Supports 2's complement signed and unsigned operations
- Supports inputs ranging from 1 to 256 bits wide

- Supports outputs ranging from 1 to 258 bits wide
- Optional registered output with optional clock enable and asynchronous and synchronous controls
- Optional Bypass (Load) capability
- Optional pipelined operation
- Incorporates Xilinx Smart-IP technology for maximum performance
- To be used with version 3.1i and later of the Xilinx CORE Generator System

Functional Description

The Adder/Subracter module can create adders ($A+B$), subracters ($A-B$) and adder/subracters which operate on signed or unsigned data. The data inputs are provided on the A and B input buses, and optionally, the B value can be set to a constant. The result is available on the output bus. Optional carry input and carry/borrow/overflow outputs are available. Outputs can be registered or non-registered. When a registered output is selected options are also pro-

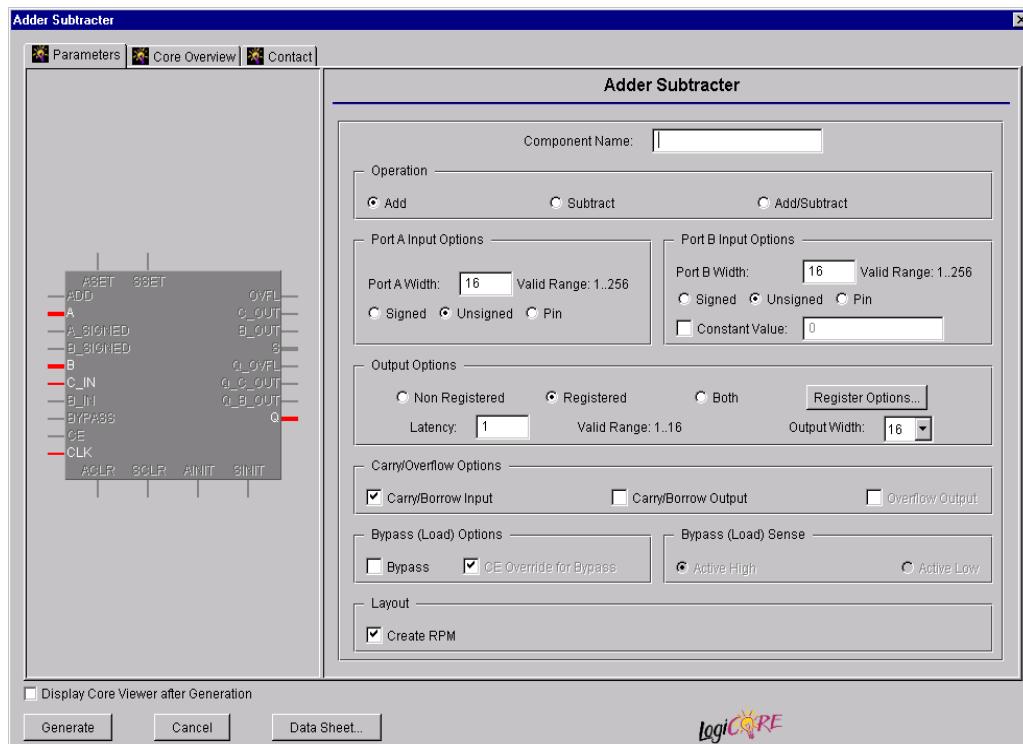


Figure 1: Main Adder/Subracter Parameterization Screen

vided for **Clock Enable**, **Asynchronous Set**, **Clear**, and **Init**, and **Synchronous Set**, **Clear** and **Init**. An optional **Bypass** capability is also provided which can load the value on the B port directly into the output register. A registered module can be optionally pipelined. The module can optionally be generated as a Relationally Placed Macro (RPM) or as unplaced logic. When an RPM is generated the logic is placed in a column.

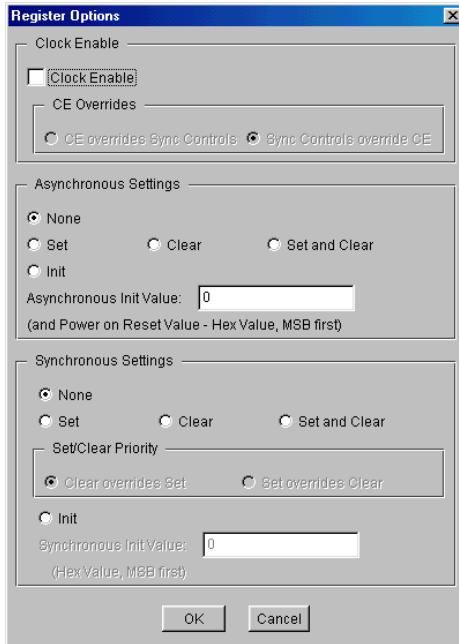


Figure 2: Adder/Subtractor Register Options Parameterization Screen

Pinout

Signal names for the schematic symbol are shown in Figure 3 and described in Table 1. Note that Figure 3 shows the C_OUT and Q_C_OUT pins which appear on adder configurations. For a subtracter these pins are named B_IN, B_OUT and Q_B_OUT, respectively.

Table 1: Core Signal Pinout

Signal	Signal Direction	Description
A[N:0]	Input	A Input bus
A_SIGNED	Input	A Input sign control
B_SIGNED	Input	B Input sign control
B[M:0]	Input	B Input bus
ADD	Input	Controls the operation performed by an Adder/Subtractor (High = Addition, Low = Subtraction).

Table 1: Core Signal Pinout (Cont.)

Signal	Signal Direction	Description
C_IN	Input	Carry Input
B_IN	Input	Borrow Input (Subtracter only)
OVFL	Output	Overflow Output (signed modules only)
C_OUT	Output	Carry Output (Adder and Adder/Subtracter only)
B_OUT	Output	Borrow Output (Subtracter only - active low)
S[P:0]	Output	Non-registered output
D_OVFL		Internal
D_C_OUT		Internal
D[P:0]		Internal
BYPASS	Input	Bypass Control Signal
CE	Input	Clock Enable
CLK	Input	Clock - rising edge clock signal
ASET	Input	Asynchronous Set: forces registered output to a high state when driven
ACLR	Input	Asynchronous Clear - forces outputs to a low state when driven
SSET	Input	Synchronous Set - forces registered output to a high state on next concurrent clock edge
SCLR	Input	Synchronous Clear - forces registered output to a low state on next concurrent clock edge
AINIT	Input	Asynchronous Initialize - forces registered outputs to user defined state when driven
SINIT	Input	Synchronous Initialize - forces registered outputs to user defined state on next concurrent clock edge
Q_OVFL	Output	Registered Overflow Output (signed modules only)
Q_C_OUT	Output	Registered Carry Output (Adder and Adder/Subtracter only)
Q_B_OUT	Output	Registered Borrow Output (Subtracter only)
Q[P:0]	Output	Registered output

Note:

All control inputs are Active High. Should an Active Low input be required for a particular control pin an inverter must be placed in the path to the pin. The inverter will be absorbed appropriately during mapping.

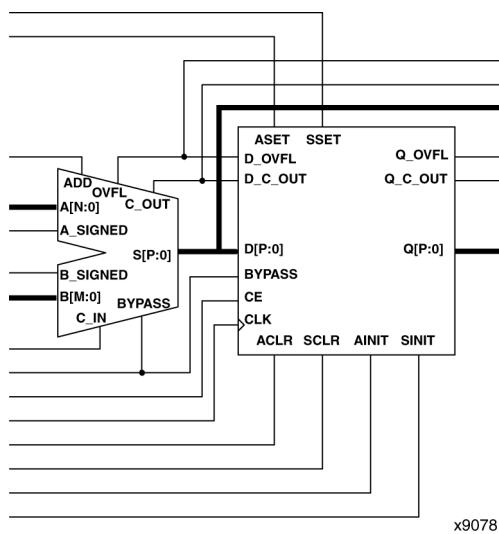


Figure 2: Core Schematic Symbol

CORE Generator Parameters

The main CORE Generator parameterization screen for this module is shown in Figure 1. The parameters are as follows:

- **Component Name:** The component name is used as the base name of the output files generated for this module. Names must begin with a letter and must be composed from the following characters: a to z, 0 to 9 and “_”.
- **Operation:** Select the appropriate radio button for the operation required. The default setting is **Add**.
- **Port A Input Options:**
 - **Port A Width:** Enter the width of the Port A input. The valid range is 1 to 256. The default value is 16.
 - **Port A Sign:** Enter the sign of the Port A input. The default value is **Unsigned**.
- **Port B Input Options:**
 - **Port B Width:** Enter the width of the Port B input. The valid range is 1 to 256. The default value is 16.
 - **Port B Sign:** Enter the sign of the Port B data. The default value is **Unsigned**.
 - **Constant Value:** When this check box is checked Port B is set to the value that is typed into the adjacent text box. The Constant Value must be entered in hex format and must not exceed the specified **Port B Width**. In most cases specifying Port B to be a constant will create a module without Port B. The only exception to this is when bypass functionality is requested, as Port B is needed to provide the bypass data in this case. The default setting is for the Port B value to be provided via Port B.

- **Output Options:**

- Select the appropriate radio button for the types of outputs required. The output options settings selected here apply to all outputs. The default setting is registered.
- **Latency:** Enter the required number of clock cycles delay from input to output for the module. See Pipelined Operation for more details. This control is available only when a registered output only has been requested via the Output Options.
- **Output Width:** The output width is specified using the pull-down list. The valid range varies depending on the settings of **Port A Width**, **Port A Sign**, and **Port B Width** and **Port B Sign** as shown in Table 2.
- **Register Options:** This button is only enabled when a registered output has been requested via the **Output Options**. Clicking on this button brings up the Register Options parameterization screen (see Figure 2).
- **Carry/Overflow Options:**

- **Carry/Borrow Input:** The presence of a C_IN or B_IN pin is controlled by the setting of this check box. The pin generated for adders and adder/subtractors is named C_IN. The pin generated for subtractors is named B_IN. The default behavior is to generate a C_IN or B_IN pin.
- **Carry/Borrow Output:** The presence of a C_OUT or B_OUT pin is controlled by the setting of this check box. This option is only enabled when the module generates an unsigned result (see Table 2). The pin generated for adders and adder/subtractors is named C_OUT. The pin generated for subtractors is named B_OUT. The default behavior is to not generate a C_OUT or B_OUT pin.
- **Overflow Output:** The presence of an OVFL pin is controlled by the setting of this check box. This option is only enabled when the module generates a signed result (see Table 2). The default behavior is to not generate an OVFL pin.
- **Bypass:** Activating the BYPASS pin allows the value on Port B to pass through the logic and be loaded into the output register on the next active clock edge. This check box is only available on a registered module. The default is for no BYPASS pin to be generated.
- **CE Override for Bypass:** This parameter controls whether or not the BYPASS input is qualified by CE. When this box is checked the activation of the BYPASS signal will also enable the register. When this box is unchecked the register needs to have CE active in order to load the Port B data. By default this check box is checked.
- **Bypass Sense:** BYPASS is the only pin that has a parameter to control its active sense. This is because selection of an **Active Low** bypass results in a significant area savings for the module. By default this parameter is set to **Active High** so that it conforms with the active sense of all other control signals.

Table 2: Availability of Carry/Borrow/Overflow Outputs and Output Data Type/Size Against Input Data Type

A[N:0]	B[M:0]	S[P:0]	Valid Values for P ¹	C_OUT/B_OUT	OVFL
Unsigned	Unsigned	Unsigned	P = Q	Available	Not Available
			P = Q + 1	Not Available	Not Available
Unsigned	Signed or by input pin	Signed	P = Q + 2	Not Available	Not Available
Signed or by input pin	Unsigned	Signed	P = Q + 2	Not Available	Not Available
Signed or by input pin	Signed or by input pin	Signed	P = Q	Not Available	Available
			P = Q + 1	Not Available	Not Available

Note:

1. Q represents the larger of N or M.

- **Create RPM:** When this box is checked the module will be generated with relative location attributes attached. The resulting placement of the module will be in a column with two bits per slice. The default operation is to create an RPM.

Note that when a module is created as an RPM it is possible that one or more of the module dimensions may exceed those of the device being targeted. If this is the case mapping errors will occur and the compilation process will fail. In this case the module should be regenerated with the **Create RPM** checkbox unchecked.

The Register Options parameterization screen for this module is shown in Figure 2. The parameters are as follows:

- **Clock Enable:** When this box is checked the module is generated with a clock enable input. The default setting is unchecked.
- **CE Overrides:** This parameter controls whether or not the SSET, SCLR, and SINIT inputs are qualified by CE. This parameter is only enabled when a **Clock Enable** input has been requested.

When **CE Overrides Sync Controls** is selected an active level on any of the synchronous control inputs will only be acted upon when the CE pin is Active. Note that this is not the way that the dedicated inputs on the flip-flop primitives work, and so setting the **CE Overrides** parameter to **CE Overrides Sync Controls** will force the synchronous control functionality to be implemented using logic in the Look Up Tables (LUTs) preceding the output register. This results in increased resource utilization even when asynchronous controls are not present.

When **Sync Controls override CE** is selected an active level on any of the synchronous control inputs will be acted upon irrespective of the state of the CE pin. This setting is more efficient when asynchronous inputs are not present because it allows the dedicated inputs on the flip-flop primitives to be used for the synchronous control functions. It is less efficient when the presence of asynchronous inputs force the synchronous control function-

ality to be implemented using logic in the LUTs preceding the output register. This is because the CE signal has to be gated with the synchronous control inputs so that they can all generate a CE signal to the flip-flops, slowing down the CE path and resulting in slower overall operation of the module.

The default setting is **Sync Controls Override CE** so that a more efficient implementation can be generated.

- **Asynchronous Settings:** All asynchronous controls are implemented using the dedicated inputs on the flip-flop primitives. The module can be generated with the following asynchronous control inputs by clicking on the appropriate button:
 - **None:** No asynchronous control inputs. This is the default setting.
 - **Set:** An ASET input pin is generated.
 - **Clear:** An ACLR input pin is generated.
 - **Set and Clear:** Both ASET and ACLR input pins are generated. ACLR has priority over ASET when both are asserted at the same time.
 - **Init:** An AINIT input pin is generated which, when asserted, will asynchronously set the output register to the value defined in the **Asynchronous Init Value** text box.
- **Asynchronous Init Value:** This text box accepts a hex value whose width must be less than or equal to the **Input Bus Width**. If a value is entered that is fewer bits than the data width of the output register it is padded with zeros. An invalid value is highlighted in red in the text box.
The default value is 0.
- **Synchronous Settings:** When no asynchronous controls are implemented (i.e. the **Asynchronous Setting** is **None**) the synchronous controls can be implemented using the dedicated inputs on the flip-flop primitives. There are exceptions to this, see the description of the **Set/Clear Priority** and **CE Overrides** parameters.

When asynchronous controls are present any synchronous control functionality must be implemented using logic in the Look Up Tables (LUTs) preceding the output register. In the case when a non-registered output is not present, this logic can (in some cases) be absorbed into the same LUTs used to implement the gate function. In cases where this is not possible the synchronous control logic will require an additional LUT per output bit.

The module can be generated with the following synchronous control inputs by clicking on the appropriate button:

- **None:** No synchronous control inputs. This is the default setting.
- **Set:** An SSET input pin is generated.
- **Clear:** An SCLR input pin is generated.
- **Set and Clear:** Both SSET and SCLR input pins are generated. SCLR/SSET priority is defined by the setting of the Set/Reset Priority parameter.
- **Init:** An SINIT input pin is generated which, when asserted, will asynchronously set the output register to the value defined in the **Synchronous Init Value** text box.
- **Set/Reset Priority:** By selecting the appropriate radio button the priority of synchronous clear to synchronous set can be controlled. This parameter is only enabled when both synchronous set and synchronous clear have been requested.

It is not possible for **Set** to override **Clear** when the synchronous control functionality is implemented using the dedicated inputs on the flip-flop primitives. This can only be implemented using logic in the LUTs preceding the output register.

The default setting is **Clear Overrides Set** so that a more efficient implementation can be generated.

- **Synchronous Init Value:** This text box accepts a hex value whose width must be less than or equal to the **Input Bus Width**. If a value is entered that is fewer bits than the data width of the register it is padded with zeros. An invalid value is highlighted in red in the text box. This parameter is only enabled when the **Synchronous Settings** parameter is set to **Init**. The default value is 0.

Pipelined Operation

The adder/subtractor module can be optionally pipelined in order to improve speed.

The pipelined operation is controlled by the Latency text box on the main parameterization screen. When the Output Options are set to Registered, this control becomes enabled. A latency of 1 is the normal registered operation of the module, and other values of latency specify the num-

ber of clock cycles between data operands being set on the inputs and the sum or difference appearing at the outputs.

When a pipelined adder/subtractor has been generated, the data within the pipeline will be invalidated by a change in the state of the ADD input; the outputs will not be valid until the number of clock cycles specified by the latency control. Similarly, after power up, the module will take the same number of clock cycles for the outputs to become valid.

If bypass is requested on a pipelined module, the bypass value will appear on the outputs after the number of clock cycles specified by the latency control.

Power On Conditions

See the **FD-based Register** datasheet for information on the power up values for registered modules.

Parameter Values in the XCO File

Names of XCO file parameters and their parameter values are identical to the names and values shown in the GUI, except that underscore characters (_) are used instead of spaces. The text in an XCO file is case insensitive.

Table 3 shows the XCO file parameters and values, and summarizes the GUI defaults. The following is an example of the CSET parameters in an XCO file:

```
CSET component_name = abc123
CSET operation = adder
CSET port_a_width = 16
CSET port_a_sign = unsigned
CSET port_b_width = 16
CSET port_b_sign = unsigned
CSET port_b_constant = FALSE
CSET port_b_constant_value = 0000
CSET output_options = registered
CSET latency = 1
CSET output_width = 16
CSET carry_borrow_input = TRUE
CSET carry_borrow_output = FALSE
CSET overflow_output = FALSE
CSET bypass = FALSE
CSET ce_override_for_bypass = FALSE
CSET bypass_sense = active_high
CSET create_rpm = TRUE
CSET clock_enable = FALSE
CSET ce_overrides = sync_controls_override_ce
CSET asynchronous_settings = none
CSET async_init_value = 0000
CSET synchronous_settings = none
CSET sync_init_value = 0000
CSET set_clear_priority = clear_overrides_set
```

Core Resource Utilization

For an accurate measure of the usage of primitives, slices, and CLBs for a particular point solution, check the **Display Core Viewer after Generation** checkbox in the CORE Generator.

Ordering Information

This core can be downloaded, free of cost, from the Xilinx IP Center (<http://www.xilinx.com/ipcenter>) for use with the Xilinx CORE Generator™ System V3.1i and later. The CORE Generator System tool is bundled with all Xilinx Alliance and Foundation Series Software packages.

To order Xilinx software online, visit the Xilinx Silicon Expresso Cafe at <http://toolbox.xilinx.com/cgi-bin/xilinx.storefront/en/catalog/1006>.

Xilinx software can also be ordered through your local Xilinx sales office. Information on the sales office nearest you is available at <http://www.xilinx.com/company/sales.htm>.

Table 3: Default Values and XCO File Values

Parameter	XCO File values	Default GUI Setting
component_name	ASCII text starting with a letter and based upon the following character set: a..z, 0..9 and _	blank
operation	One of the following keywords: add, subtract, add_subtract	adder
port_a_width	Integer in the range 1 to 256	16
port_a_sign	One of the following keywords: unsigned, signed, pin	unsigned
port_b_width	Integer in the range 1 to 256	16
port_b_sign	One of the following keywords: unsigned, signed, pin	unsigned
port_b_constant	One of the following keywords: true, false	false
port_b_constant_value	Hex value whose value does not exceed $2^{\text{port_b_width}} - 1$	0
output_options	One of the following keywords: non_registered, registered, both	registered
latency	Integer in the range 0 to either output_width or 64, whichever is smaller. If overflow is also required, the maximum value is reduced by 1.	1
output_width	See Table 2	16
carry_borrow_input	One of the following keywords: true, false	true
carry_borrow_output	One of the following keywords: true, false	false
overflow_output	One of the following keywords: true, false	false
bypass	One of the following keywords: true, false	false
ce_override_for_bypass	One of the following keywords: true, false	true
bypass_sense	One of the following keywords: active_high, active_low	active_high
create_rpm	One of the following keywords: true, false	true
clock_enable	One of the following keywords: true, false	false
ce_overrides	One of the following keywords: sync_controls_override_ce, ce_overrides_sync_controls	sync_controls_override_ce
asynchronous_settings	One of the following keywords: none, set, clear, set_and_clear, init	none
async_init_value	Hex value whose value does not exceed $2^{\text{output_width}} - 1$	0
synchronous_settings	One of the following keywords: none, set, clear, set_and_clear, init	none
sync_init_value	Hex value whose value does not exceed $2^{\text{output_width}} - 1$	0
set_clear_priority	One of the following keywords: clear_overrides_set, set_overrides_clear	clear_overrides_set



Xilinx Inc.
 2100 Logic Drive
 San Jose, CA 95124
 Phone: +1 408-559-7778
 Fax: +1 408-559-7114
 URL: www.xilinx.com/ipcenter
 Support: support.xilinx.com

Features

- Drop-in module for Virtex™-II, Virtex™, Virtex™-E and Spartan™-II FPGAs
- Generates comparison logic for $A = B$, $A <> B$, $A \leq B$, $A < B$, $A \geq B$ or $A > B$
- Operates on twos complement signed or unsigned data
- Supports inputs from 1 to 256 bits wide
- Optional compare to constant capability
- Optional clock enable and asynchronous and synchronous controls
- Incorporates Xilinx Smart-IP technology for maximum performance
- To be used with version 3.1i and later of the Xilinx CORE Generator System

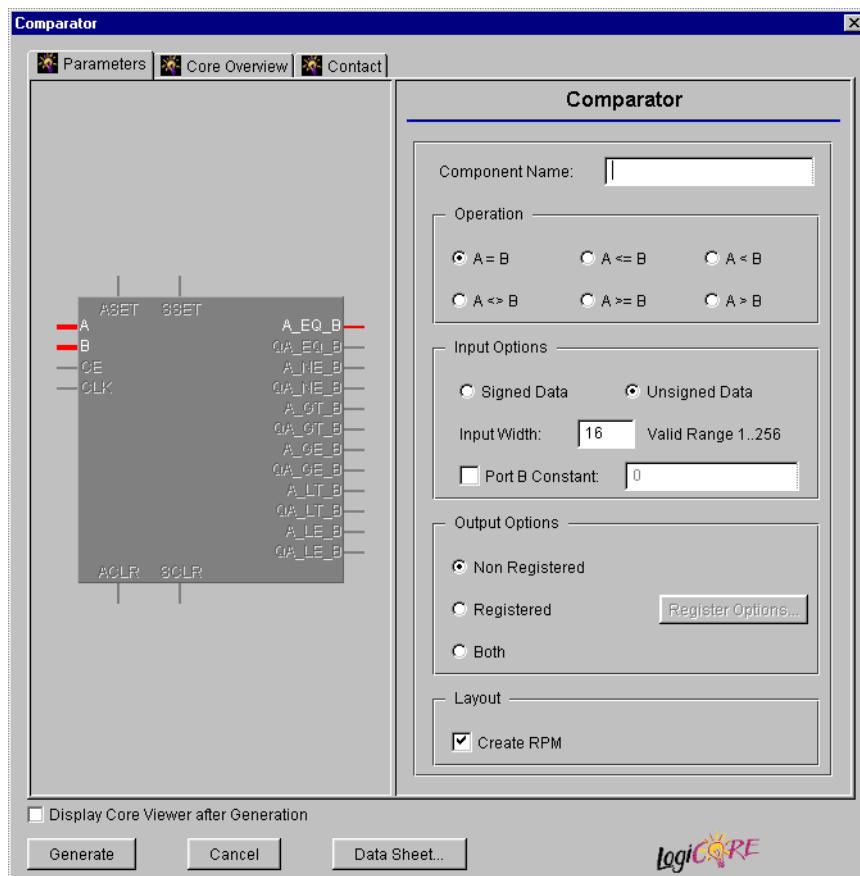


Figure 1: Main Comparator Parameterization Screen

Functional Description

The comparator is used to create comparison logic that performs one of the following functions: $A = B$, $A <> B$, $A < B$, $A \geq B$ or $A > B$. A and B are external ports of up to 256 bits wide and B can optionally be set to a constant value. The module can handle two's complement signed or unsigned data. Options are provided for **Clock Enable**, **Asynchronous Set** and **Clear**, and **Synchronous Set and Clear**. The module can optionally be generated as a Relationally Placed Macro (RPM) or as unplaced logic. When an RPM is generated the logic is placed in a column.

Pinout

Signal names for the schematic symbol are shown in Figure 3 and described in Table 1.

CORE Generator Parameters

The main CORE Generator parameterization screen for this module is shown in Figure 1. The parameters are as follows:

- Component Name:** The component name is used as the base name of the output files generated for this module. Names must begin with a letter and must be composed from the following characters: a to z, 0 to 9 and “_”.
- Operation:** Select the appropriate radio button for the

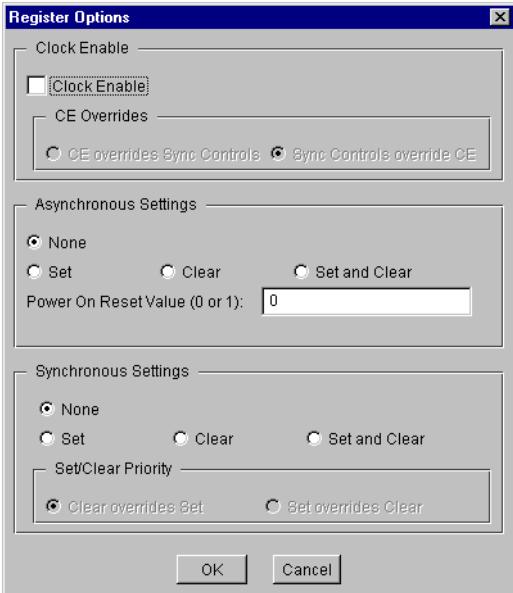
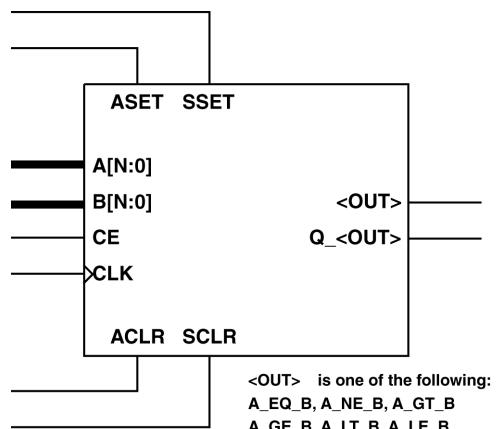


Figure 2: Comparator Register Options Parameterization Screen



X9081

Figure 2: Core Schematic Symbol

operation required. The default setting is **A = B**.

- Input Options:**

- **Input Width:** Enter the width of ports A and B . The valid range is 1 to 256. The default value is 16. Note that the minimum width is 2 for signed data.
- **Input Sign:** Enter the sign of the input data. The default setting is **Unsigned**.
- **Port B Constant:** When this check box is checked the Port B value will be set to the value specified in the **Port B Constant Value** text box. The default is for the Port B value to come from Port B .
- **Port B Constant Value:** Enter the value for the Port B constant. The Constant Value must be provided in hex and must not exceed the specified **Input Width**. This text box is only enabled when the **Port B Constant** check box is checked. The default value is 0.

- Output Options:** Select the appropriate radio button for the types of outputs required. The default setting is **Non Registered**.

- **Register Options:** This button is only enabled when a registered output has been requested via the **Output Options**. Clicking on this button brings up the Register Options parameterization screen (see Figure 2).
- **Create RPM:** When this box is checked the module is generated with relative location attributes attached. The resulting placement of the module is in a column with two bits per slice. The default operation is to create an RPM.

Note that when a module is created as an RPM it is possible that one or more of the module dimensions

Table 1: Core Signal Pinout

Signal	Signal Direction	Description
A[N:0]	Input	Comparator Input
B[N:0]	Input	Comparator Input
A_EQ_B	Output	A Equals B Asynchronous Output
Q_A_EQ_B	Output	A Equals B Registered Output
A_NE_B	Output	A Not Equal to B Asynchronous Output
Q_A_NE_B	Output	A Not Equal to B Registered Output
A_LE_B	Output	A Less Than or Equal to B Asynchronous Output
Q_A_LE_B	Output	A Less Than or Equal to B Registered Output
A_LT_B	Output	A Less Than B Asynchronous Output
Q_A_LT_B	Output	A Less Than B Registered Output
A_GE_B	Output	A Greater Than or Equal to B Asynchronous Output
Q_A_GE_B	Output	A Greater Than or Equal to B Registered Output
A_GT_B	Output	A Greater Than B Asynchronous Output
Q_A_GT_B	Output	A Greater Than B Registered Output
CE	Input	Clock Enable
CLK	Input	Clock - rising edge clock signal
ASET	Input	Asynchronous Set - forces the registered output to a high state when driven
ACLR	Input	Asynchronous Clear - forces outputs to a low state when driven
SSET	Input	Synchronous Set - forces the registered output to a high state on next concurrent clock edge
SCLR	Input	Synchronous Clear - forces the registered output to a low state on next concurrent clock edge

Note:

All control inputs are Active High. Should an Active Low input be required for a particular control pin an inverter must be placed in the path to the pin. The inverter will be absorbed appropriately during mapping.

may exceed those of the device being targeted. If this is the case mapping errors will occur and the compilation process will fail. In this case the module should be regenerated with the **Create RPM** checkbox unchecked.

The Register Options parameterization screen for this module is shown in Figure 2. The parameters are as follows:

- **Clock Enable:** When this box is checked the module is generated with a clock enable input. The default setting is unchecked.
- **CE Overrides:** This parameter controls whether or not the SSET and SCLR inputs are qualified by CE. This parameter is only enabled when a **Clock Enable** input has been requested.

When **CE Overrides Sync Controls** is selected an active level on any of the synchronous control inputs will only be acted upon when the CE pin is also Active. Note that this is not the way that the dedicated inputs on the flip-flop primitives work, and so setting the **CE Overrides** parameter to **CE Overrides Sync Controls** will force any synchronous control functionality to be implemented using logic in the Look Up Tables (LUTs) preceding the output register. This results in increased resource utilization.

When **Sync Controls Override CE** is selected an active level on any of the synchronous control inputs is acted upon irrespective of the state of the CE pin. This setting allows the dedicated inputs on the flip-flop primitives to be used for the synchronous control functions provided that asynchronous controls are not requested. If both asynchronous and synchronous controls are requested, the synchronous control functionality must be implemented using logic in the LUTs preceding the output register. In this case, the CE input has to be gated with the synchronous control inputs so that each synchronous control input and the CE input can generate a CE signal to the flip-flops. This results in a performance degradation for the module due to the additional gating in the CE path.

The default setting is **Sync Controls Override CE** so that a more efficient implementation can be generated.

- **Asynchronous Settings:** All asynchronous controls are implemented using the dedicated inputs on the flip-flop primitives. The module can be generated with the following asynchronous control inputs by clicking on the appropriate button:
 - **None:** No asynchronous control inputs. This is the default setting.
 - **Set:** An ASET control pin is generated.
 - **Clear:** An ACLR control pin is generated.
 - **Set and Clear:** Both ASET and ACLR control pins are generated. ACLR has priority over ASET when both are asserted at the same time.
- **Power On Reset Value:** This text box accepts a value of 0 or 1 and defines the power on value for the output

register. The default value is 0.

- **Synchronous Settings:** When no asynchronous controls are requested (i.e. the **Asynchronous Setting** is **None**) the synchronous controls can be implemented using the dedicated inputs on the flip-flop primitives. There are exceptions to this which are described in the sections for the **Set/Clear Priority** and **CE Overrides** parameters.

When asynchronous controls are present any synchronous control functionality must be implemented using logic in the Look Up Tables (LUTs) preceding the output register. With modules where a non-registered output is not required there are combinations of parameters that allow this logic to be absorbed into the same LUTs used to implement the function. In cases where this absorption is not possible the synchronous control logic will require an additional LUT per output bit.

The module can be generated with the following synchronous control inputs by clicking on the appropriate button:

- **None:** No synchronous control inputs. This is the default setting.
- **Set:** An SSET control pin is generated.
- **Clear:** An SCLR control pin is generated.
- **Set and Clear:** Both SSET and SCLR control pins are generated. SCLR/SSET priority is defined by the setting of the **Set/Clear Priority** parameter.
- **Set/Clear Priority:** By selecting the appropriate radio button the relative priority of SCLR and SSET can be controlled. This parameter is only enabled when **Set** and **Clear** is selected for **Synchronous Settings**.

A setting of **Clear Overrides Set** corresponds to the native operation of the flip-flop primitive. This setting will result in a more efficient implementation when asynchronous controls are not requested. A setting of **Set Overrides Clear** can only be implemented using logic in the LUTs preceding the output register.

The default setting is **Clear Overrides Set** so that the dedicated inputs on the flip-flops can be used if available.

Parameter Values in the XCO File

Names of XCO file parameters and their parameter values are identical to the names and values shown in the GUI, except that underscore characters (_) are used instead of spaces. The text in an XCO file is case insensitive.

Table 2 shows the XCO file parameters and values, and summarizes the GUI defaults. The following is an example of the CSET parameters in an XCO file:

```
CSET component_name = abc123
CSET operation = a_eq_b
CSET input_width = 16
CSET input_sign = unsigned
CSET port_b_constant = FALSE
CSET port_b_constant_value = 0
CSET output_options = non_registered
CSET create_rpm = TRUE
CSET clock_enable = FALSE
CSET ce_overrides = sync_controls_override_ce
CSET asynchronous_settings = none
CSET power_on_reset_value = 0
CSET synchronous_settings = none
CSET set_clear_priority = clear_overrides_set
```

Core Resource Utilization

For an accurate measure of the usage of primitives, slices, and CLBs for a particular point solution, check the **Display Core Viewer after Generation** checkbox in the CORE Generator.

Ordering Information

This core can be downloaded, free of cost, from the Xilinx IP Center (<http://www.xilinx.com/ipcenter>) for use with the Xilinx CORE Generator™ System V3.1i and later. The CORE Generator System tool is bundled with all Xilinx Alliance and Foundation Series Software packages.

To order Xilinx software online, visit the Xilinx Silicon Expresso Cafe at <http://toolbox.xilinx.com/cgi-bin/xilinx.storefront/en/catalog/1006>.

Xilinx software can also be ordered through your local Xilinx sales office. Information on the sales office nearest you is available at <http://www.xilinx.com/company/sales.htm>.

Table 2: XCO File Values and Default Values

Parameter	XCO File Values	Default GUI Setting
component_name	ASCII text starting with a letter and based upon the following character set: a..z, 0..9 and _	blank
operation	One of the following keywords: a_eq_b, a_le_b, a_lt_b, a_ne_b, a_ge_b or a_gt_b	a_eq_b
input_width	Integer in the range 1 to 256	16
input_sign	One of the following keywords: unsigned or signed	unsigned
port_b_constant	One of the following keywords: true, false	false
port_b_constant_value	Hex value whose value does not exceed $2^{\text{input_width}} - 1$	0
output_options	One of the following keywords: non_registered, registered or both	non_registered
create_rpm	One of the following keywords: true, false	true
clock_enable	One of the following keywords: true, false	false
ce_overrides	One of the following keywords: sync_controls_override_ce, ce_overrides_sync_controls	sync_controls_override_ce
asynchronous_settings	One of the following keywords: none, set, clear, set_and_clear	none
power_on_reset_value	One of the following values: 0, 1	0
synchronous_settings	One of the following keywords: none, set, clear, set_and_clear	none
set_clear_priority	One of the following keywords: clear_overrides_set or set_overrides_clear	clear_overrides_set

APPENDIX-E

XILINX FPGA FLOORPLAN & LAYOUT

APPENDIX-F

**XILINX VIRTEX 1000E DATA SHEETS AND XILINX
FPGA TEST BOARD & TEST SETUP**

Features

- Fast, High-Density 1.8 V FPGA Family
 - Densities from 58 Kb to 4 Mb system gates
 - 130 MHz internal performance (four LUT levels)
 - Designed for low-power operation
 - PCI compliant 3.3 V, 32/64-bit, 33/ 66-MHz
- Highly Flexible SelectI/O+™ Technology
 - Supports 20 high-performance interface standards
 - Up to 804 singled-ended I/Os or 344 differential I/O pairs for an aggregate bandwidth of > 100 Gb/s
- Differential Signalling Support
 - LVDS (622 Mb/s), BLVDS (Bus LVDS), LVPECL
 - Differential I/O signals can be input, output, or I/O
 - Compatible with standard differential devices
 - LVPECL and LVDS clock inputs for 300+ MHz clocks
- Proprietary High-Performance SelectLink™ Technology
 - Double Data Rate (DDR) to Virtex-E link
 - Web-based HDL generation methodology
- Sophisticated SelectRAM+™ Memory Hierarchy
 - 1 Mb of internal configurable distributed RAM
 - Up to 832 Kb of synchronous internal block RAM
 - True Dual-Port™ BlockRAM capability
 - Memory bandwidth up to 1.66 Tb/s (equivalent bandwidth of over 100 RAMBUS channels)
 - Designed for high-performance Interfaces to External Memories
 - 200 MHz ZBT* SRAMs
 - 200 Mb/s DDR SDRAMs
 - Supported by free Synthesizable reference design
- High-Performance Built-In Clock Management Circuitry
 - Eight fully digital Delay-Locked Loops (DLLs)
 - Digitally-Synthesized 50% duty cycle for Double Data Rate (DDR) Applications
 - Clock Multiply and Divide
 - Zero-delay conversion of high-speed LVPECL/LVDS clocks to any I/O standard
- Flexible Architecture Balances Speed and Density
 - Dedicated carry logic for high-speed arithmetic
 - Dedicated multiplier support
 - Cascade chain for wide-input function
 - Abundant registers/latches with clock enable, and dual synchronous/asynchronous set and reset
 - Internal 3-state bussing
 - IEEE 1149.1 boundary-scan logic
 - Die-temperature sensor diode
- Supported by Xilinx Foundation™ and Alliance Series™ Development Systems
 - Further compile time reduction of 50%
 - Internet Team Design (ITD) tool ideal for million-plus gate density designs
 - Wide selection of PC and workstation platforms
- SRAM-Based In-System Configuration
 - Unlimited re-programmability
- Advanced Packaging Options
 - 0.8 mm Chip-scale
 - 1.0 mm BGA
 - 1.27 mm BGA
 - HQ/PQ
- 0.18 µm 6-Layer Metal Process
- 100% Factory Tested

* ZBT is a trademark of Integrated Device Technology, Inc.

Table 1: Virtex-E Field-Programmable Gate Array Family Members

Device	System Gates	Logic Gates	CLB Array	Logic Cells	Differential I/O Pairs	User I/O	BlockRAM Bits	Distributed RAM Bits
XCV50E	71,693	20,736	16 x 24	1,728	83	176	65,536	24,576
XCV100E	128,236	32,400	20 x 30	2,700	83	196	81,920	38,400
XCV200E	306,393	63,504	28 x 42	5,292	119	284	114,688	75,264
XCV300E	411,955	82,944	32 x 48	6,912	137	316	131,072	98,304
XCV400E	569,952	129,600	40 x 60	10,800	183	404	163,840	153,600
XCV600E	985,882	186,624	48 x 72	15,552	247	512	294,912	221,184
XCV1000E	1,569,178	331,776	64 x 96	27,648	281	660	393,216	393,216
XCV1600E	2,188,742	419,904	72 x 108	34,992	344	724	589,824	497,664
XCV2000E	2,541,952	518,400	80 x 120	43,200	344	804	655,360	614,400
XCV2600E	3,263,755	685,584	92 x 138	57,132	344	804	753,664	812,544
XCV3200E	4,074,387	876,096	104 x 156	73,008	344	804	851,968	1,038,336

Virtex-E Compared to Virtex Devices

The Virtex-E family offers up to 43,200 logic cells in devices up to 30% faster than the Virtex family.

I/O performance is increased to 622 Mb/s using Source Synchronous data transmission architectures and synchronous system performance up to 240 MHz using singled-ended SelectI/O technology. Additional I/O standards are supported, notably LVPECL, LVDS, and BLVDS, which use two pins per signal. Almost all signal pins can be used for these new standards.

Virtex-E devices have up to 640 Kb of faster (250 MHz) block SelectRAM, but the individual RAMs are the same size and structure as in the Virtex family. They also have eight DLLs instead of the four in Virtex devices. Each individual DLL is slightly improved with easier clock mirroring and 4x frequency multiplication.

V_{CCINT} , the supply voltage for the internal logic and memory, is 1.8 V, instead of 2.5 V for Virtex devices. Advanced processing and 0.18 μ m design rules have resulted in smaller dice, faster speed, and lower power consumption.

I/O pins are 3 V tolerant, and can be 5 V tolerant with an external 100 Ω resistor. PCI 5 V is not supported. With the addition of appropriate external resistors, any pin can tolerate any voltage desired.

Banking rules are different. With Virtex devices, all input buffers are powered by V_{CCINT} . With Virtex-E devices, the LVTTL, LVCMSO2, and PCI input buffers are powered by the I/O supply voltage V_{CCO} .

The Virtex-E family is not bitstream-compatible with the Virtex family, but Virtex designs can be compiled into equivalent Virtex-E devices.

The same device in the same package for the Virtex-E and Virtex families are pin-compatible with some minor exceptions. See the data sheet pinout section for details.

General Description

The Virtex-E FPGA family delivers high-performance, high-capacity programmable logic solutions. Dramatic increases in silicon efficiency result from optimizing the new architecture for place-and-route efficiency and exploiting an aggressive 6-layer metal 0.18 μ m CMOS process. These advances make Virtex-E FPGAs powerful and flexible alternatives to mask-programmed gate arrays. The Virtex-E family includes the nine members in Table 1.

Building on experience gained from Virtex FPGAs, the Virtex-E family is an evolutionary step forward in programmable logic design. Combining a wide variety of programmable system features, a rich hierarchy of fast, flexible interconnect resources, and advanced process technology, the Virtex-E family delivers a high-speed and high-capacity programmable logic solution that enhances design flexibility while reducing time-to-market.

Virtex-E Architecture

Virtex-E devices feature a flexible, regular architecture that comprises an array of configurable logic blocks (CLBs) surrounded by programmable input/output blocks (IOBs), all interconnected by a rich hierarchy of fast, versatile routing

resources. The abundance of routing resources permits the Virtex-E family to accommodate even the largest and most complex designs.

Virtex-E FPGAs are SRAM-based, and are customized by loading configuration data into internal memory cells. Configuration data can be read from an external SPROM (master serial mode), or can be written into the FPGA (SelectMAP™, slave serial, and JTAG modes).

The standard Xilinx Foundation Series™ and Alliance Series™ Development systems deliver complete design support for Virtex-E, covering every aspect from behavioral and schematic entry, through simulation, automatic design translation and implementation, to the creation and downloading of a configuration bit stream.

Higher Performance

Virtex-E devices provide better performance than previous generations of FPGAs. Designs can achieve synchronous system clock rates up to 240 MHz including I/O or 622 Mb/s using Source Synchronous data transmission architectures. Virtex-E I/Os comply fully with 3.3 V PCI specifications, and interfaces can be implemented that operate at 33 MHz or 66 MHz.

While performance is design-dependent, many designs operate internally at speeds in excess of 133 MHz and can

achieve over 311 MHz. **Table 2** shows performance data for representative circuits, using worst-case timing parameters.

Table 2: Performance for Common Circuit Functions

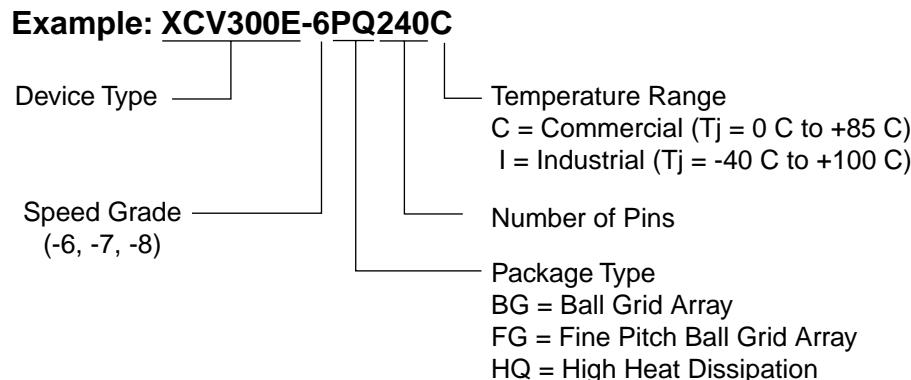
Function	Bits	Virtex-E (-7)
Register-to-Register		
Adder	16	4.3 ns
	64	6.3 ns
Pipelined Multiplier		
	8 x 8	4.4 ns
	16 x 16	5.1 ns
Address Decoder		
	16	3.8 ns
	64	5.5 ns
16:1 Multiplexer		4.6 ns
Parity Tree		
	9	3.5 ns
	18	4.3 ns
	36	5.9 ns
Chip-to-Chip		
HSTL Class IV		
LVTTL,16mA, fast slew		
LVDS		

Virtex-E Device/Package Combinations and Maximum I/O

Table 3: Virtex-E Family Maximum User I/O by Device/Package (Excluding Dedicated Clock Pins)

	XCV50E	XCV100E	XCV200E	XCV300E	XCV400E	XCV600E	XCV1000E	XCV1600E	XCV2000E	XCV2600E	XCV3200E
CS144	94	94	94								
PQ240	158	158	158	158	158						
HQ240						158	158				
BG352		196	260	260							
BG432				316	316	316					
BG560					404	404	404	404	404		
FG256	176	176	176	176							
FG456			284	312							
FG676					404	444					
FG680						512	512	512	512		
FG860							660	660	660		
FG900						512	660	700			
FG1156							660	724	804	804	
CG1156											804

Virtex-E Ordering Information



DS022_043_072000

Figure 1: Ordering Information

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/7/99	1.0	Initial Xilinx release.
1/10/00	1.1	Re-released with spd.txt v. 1.18, FG860/900/1156 package information, and additional DLL, Select RAM and SelectI/O information.
1/28/00	1.2	Added Delay Measurement Methodology table, updated SelectI/O section, Figures 30, 54, & 55, text explaining Table 5, T_{BYP} values, buffered Hex Line info, p. 8, I/O Timing Measurement notes, notes for Tables 15, 16, and corrected F1156 pinout table footnote references.
2/29/00	1.3	Updated pinout tables, V_{CC} page 20, and corrected Figure 20.
5/23/00	1.4	Correction to table on p. 22.
7/10/00	1.5	<ul style="list-style-type: none"> • Numerous minor edits. • Data sheet upgraded to Preliminary. • Preview -8 numbers added to Virtex-E Electrical Characteristics tables.
8/1/00	1.6	<ul style="list-style-type: none"> • Reformatted entire document to follow new style guidelines. • Changed speed grade values in tables on pages 35-37.
9/20/00	1.7	<ul style="list-style-type: none"> • Min values added to Virtex-E Electrical Characteristics tables. • XCV2600E and XCV3200E numbers added to Virtex-E Electrical Characteristics tables (Module 3). • Corrected user I/O count for XCV100E device in Table 1 (Module 1). • Changed several pins to “No Connect in the XCV100E” and removed duplicate V_{CCINT} pins in Table ~ (Module 4). • Changed pin J10 to “No connect in XCV600E” in Table 74 (Module 4). • Changed pin J30 to “VREF option only in the XCV600E” in Table 74 (Module 4). • Corrected pair 18 in Table 75 (Module 4) to be “AO in the XCV1000E, XCV1600E”.

Date	Version	Revision
11/20/00	1.8	<ul style="list-style-type: none"> Upgraded speed grade -8 numbers in Virtex-E Electrical Characteristics tables to Preliminary. Updated minimums in Table 13 and added notes to Table 14. Added to note 2 to Absolute Maximum Ratings. Changed speed grade -8 numbers for $T_{SHCKO32}$, T_{REG}, T_{BCCS}, and T_{ICKOF}. Changed all minimum hold times to -0.4 under Global Clock Set-Up and Hold for LVTTI Standard, with DLL. Revised maximum T_{DLLPW} in -6 speed grade for DLL Timing Parameters. Changed GCLK0 to BA22 for FG860 package in Table 46.
2/12/01	1.9	<ul style="list-style-type: none"> Revised footnote for Table 14. Added numbers to Virtex-E Electrical Characteristics tables for XCV1000E and XCV2000E devices. Updated Table 27 and Table 78 to include values for XCV400E and XCV600E devices. Revised Table 62 to include pinout information for the XCV400E and XCV600E devices in the BG560 package. Updated footnotes 1 and 2 for Table 76 to include XCV2600E and XCV3200E devices.
4/2/01	2.0	<ul style="list-style-type: none"> Updated numerous values in Virtex-E Switching Characteristics tables. Converted data sheet to modularized format. See the Virtex-E Data Sheet section.

Virtex-E Data Sheet

The Virtex-E Data Sheet contains the following modules:

- DS022-1, Virtex-E 1.8V FPGAs:
[Introduction and Ordering Information \(Module 1\)](#)
- DS022-2, Virtex-E 1.8V FPGAs:
[Functional Description \(Module 2\)](#)
- DS022-3, Virtex-E 1.8V FPGAs:
[DC and Switching Characteristics \(Module 3\)](#)
- DS022-4, Virtex-E 1.8V FPGAs:
[Pinout Tables \(Module 4\)](#)

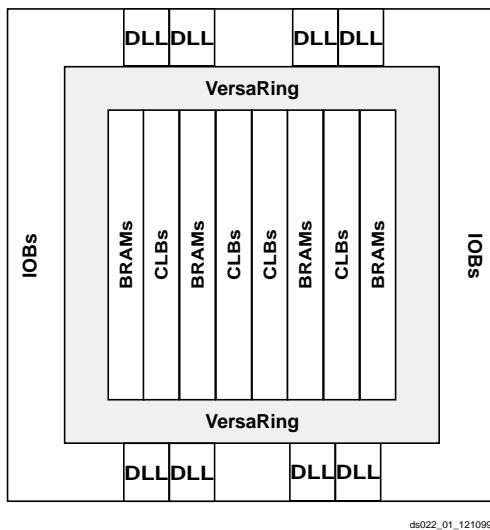
Architectural Description

Virtex-E Array

The Virtex-E user-programmable gate array, shown in [Figure 1](#), comprises two major configurable elements: configurable logic blocks (CLBs) and input/output blocks (IOBs).

- CLBs provide the functional elements for constructing logic
- IOBs provide the interface between the package pins and the CLBs

CLBs interconnect through a general routing matrix (GRM). The GRM comprises an array of routing switches located at the intersections of horizontal and vertical routing channels. Each CLB nests into a VersaBlock™ that also provides local routing resources to connect the CLB to the GRM.



[Figure 1: Virtex-E Architecture Overview](#)

The VersaRing™ I/O interface provides additional routing resources around the periphery of the device. This routing improves I/O routability and facilitates pin locking.

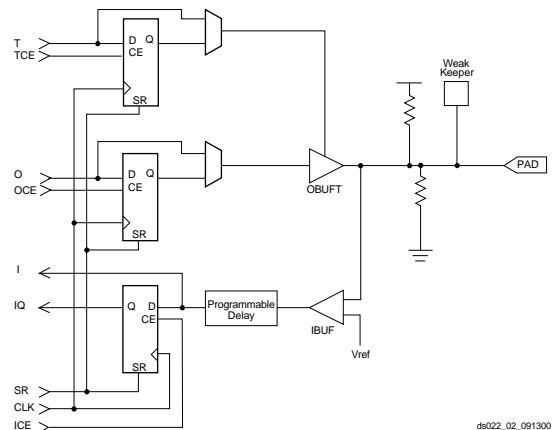
The Virtex-E architecture also includes the following circuits that connect to the GRM.

- Dedicated block memories of 4096 bits each
- Clock DLLs for clock-distribution delay compensation and clock domain control
- 3-State buffers (BUFTs) associated with each CLB that drive dedicated segmentable horizontal routing resources

Values stored in static memory cells control the configurable logic elements and interconnect resources. These values load into the memory cells on power-up, and can reload if necessary to change the function of the device.

Input/Output Block

The Virtex-E IOB, [Figure 2](#), features SelectI/O+ inputs and outputs that support a wide variety of I/O signalling standards, see [Table 1](#).



[Figure 2: Virtex-E Input/Output Block \(IOB\)](#)

The three IOB storage elements function either as edge-triggered D-type flip-flops or as level-sensitive latches. Each IOB has a clock signal (CLK) shared by the three flip-flops and independent clock enable signals for each flip-flop.

Table 1: Supported I/O Standards

I/O Standard	Output V_{CCO}	Input V_{CCO}	Input V_{REF}	Board Termination Voltage (V_{TT})
LVTTL	3.3	3.3	N/A	N/A
LVCMOS2	2.5	2.5	N/A	N/A
LVCMOS18	1.8	1.8	N/A	N/A
SSTL3 I & II	3.3	N/A	1.50	1.50
SSTL2 I & II	2.5	N/A	1.25	1.25
GTL	N/A	N/A	0.80	1.20
GTL+	N/A	N/A	1.0	1.50
HSTL I	1.5	N/A	0.75	0.75
HSTL III & IV	1.5	N/A	0.90	1.50
CTT	3.3	N/A	1.50	1.50
AGP-2X	3.3	N/A	1.32	N/A
PCI33_3	3.3	3.3	N/A	N/A
PCI66_3	3.3	3.3	N/A	N/A
BLVDS & LVDS	2.5	N/A	N/A	N/A
LVPECL	3.3	N/A	N/A	N/A

In addition to the CLK and CE control signals, the three flip-flops share a Set/Reset (SR). For each flip-flop, this signal can be independently configured as a synchronous Set, a synchronous Reset, an asynchronous Preset, or an asynchronous Clear.

The output buffer and all of the IOB control signals have independent polarity controls.

All pads are protected against damage from electrostatic discharge (ESD) and from over-voltage transients. When PCI 3.3 V compliance is required, a conventional clamp diode is connected to the output supply voltage, V_{CCO} .

Optional pull-up, pull-down and weak-keeper circuits are attached to each pad. Prior to configuration all outputs not involved in configuration are forced into their high-impedance state. The pull-down resistors and the weak-keeper circuits are inactive, but I/Os can optionally be pulled up.

The activation of pull-up resistors prior to configuration is controlled on a global basis by the configuration mode pins. If the pull-up resistors are not activated, all the pins are in a high-impedance state. Consequently, external pull-up or pull-down resistors must be provided on pins required to be at a well-defined logic level prior to configuration.

All Virtex-E IOBs support IEEE 1149.1-compatible boundary scan testing.

Input Path

The Virtex-E IOB input path routes the input signal directly to internal logic and/ or through an optional input flip-flop.

An optional delay element at the D-input of this flip-flop eliminates pad-to-pad hold time. The delay is matched to the internal clock-distribution delay of the FPGA, and when used, assures that the pad-to-pad hold time is zero.

Each input buffer can be configured to conform to any of the low-voltage signalling standards supported. In some of these standards the input buffer utilizes a user-supplied threshold voltage, V_{REF} . The need to supply V_{REF} imposes constraints on which standards can be used in close proximity to each other. [See “I/O Banking” on page 2.](#)

There are optional pull-up and pull-down resistors at each input for use after configuration. Their value is in the range 50 – 100 kΩ.

Output Path

The output path includes a 3-state output buffer that drives the output signal onto the pad. The output signal can be routed to the buffer directly from the internal logic or through an optional IOB output flip-flop.

The 3-state control of the output can also be routed directly from the internal logic or through a flip-flop that provides synchronous enable and disable.

Each output driver can be individually programmed for a wide range of low-voltage signalling standards. Each output buffer can source up to 24 mA and sink up to 48 mA. Drive strength and slew rate controls minimize bus transients.

In most signalling standards, the output High voltage depends on an externally supplied V_{CCO} voltage. The need to supply V_{CCO} imposes constraints on which standards can be used in close proximity to each other. [See “I/O Banking” on page 2.](#)

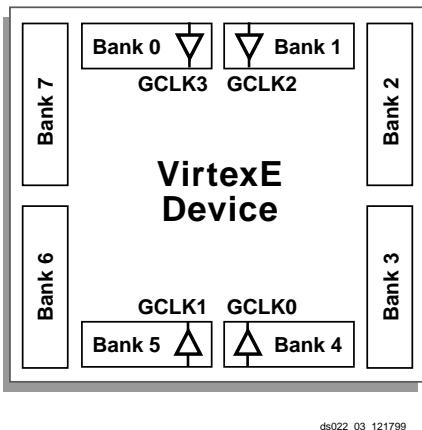
An optional weak-keeper circuit is connected to each output. When selected, the circuit monitors the voltage on the pad and weakly drives the pin High or Low to match the input signal. If the pin is connected to a multiple-source signal, the weak keeper holds the signal in its last state if all drivers are disabled. Maintaining a valid logic level in this way eliminates bus chatter.

Since the weak-keeper circuit uses the IOB input buffer to monitor the input level, an appropriate V_{REF} voltage must be provided if the signalling standard requires one. The provision of this voltage must comply with the I/O banking rules.

I/O Banking

Some of the I/O standards described above require V_{CCO} and/or V_{REF} voltages. These voltages are externally supplied and connected to device pins that serve groups of IOBs, called banks. Consequently, restrictions exist about which I/O standards can be combined within a given bank.

Eight I/O banks result from separating each edge of the FPGA into two banks, as shown in [Figure 3](#). Each bank has multiple V_{CCO} pins, all of which must be connected to the same voltage. This voltage is determined by the output standards in use.



[Figure 3: Virtex-E I/O Banks](#)

Within a bank, output standards can be mixed only if they use the same V_{CCO} . Compatible standards are shown in [Table 2](#). GTL and GTL+ appear under all voltages because their open-drain outputs do not depend on V_{CCO} .

[Table 2: Compatible Output Standards](#)

V_{CCO}	Compatible Standards
3.3 V	PCI, LVTTI, SSTL3 I, SSTL3 II, CTT, AGP, GTL, GTL+, LVPECL
2.5 V	SSTL2 I, SSTL2 II, LVCMOS2, GTL, GTL+, BLVDS, LVDS
1.8 V	LVCMOS18, GTL, GTL+
1.5 V	HSTL I, HSTL III, HSTL IV, GTL, GTL+

Some input standards require a user-supplied threshold voltage, V_{REF} . In this case, certain user-I/O pins are automatically configured as inputs for the V_{REF} voltage. Approximately one in six of the I/O pins in the bank assume this role.

The V_{REF} pins within a bank are interconnected internally and consequently only one V_{REF} voltage can be used within each bank. All V_{REF} pins in the bank, however, must be connected to the external voltage source for correct operation.

Within a bank, inputs that require V_{REF} can be mixed with those that do not. However, only one V_{REF} voltage can be used within a bank.

In Virtex-E, input buffers with LVTTI, LVCMOS2, LVCMOS18, PCI33_3, PCI66_3 standards are supplied by V_{CCO} rather than V_{CCINT} . For these standards, only input and output buffers that have the same V_{CCO} can be mixed together.

The V_{CCO} and V_{REF} pins for each bank appear in the device pin-out tables and diagrams. The diagrams also show the bank affiliation of each I/O.

Within a given package, the number of V_{REF} and V_{CCO} pins can vary depending on the size of device. In larger devices, more I/O pins convert to V_{REF} pins. Since these are always a super set of the V_{REF} pins used for smaller devices, it is possible to design a PCB that permits migration to a larger device if necessary. All the V_{REF} pins for the largest device anticipated must be connected to the V_{REF} voltage, and not used for I/O.

In smaller devices, some V_{CCO} pins used in larger devices do not connect within the package. These unconnected pins can be left unconnected externally, or can be connected to the V_{CCO} voltage to permit migration to a larger device if necessary.

Configurable Logic Blocks

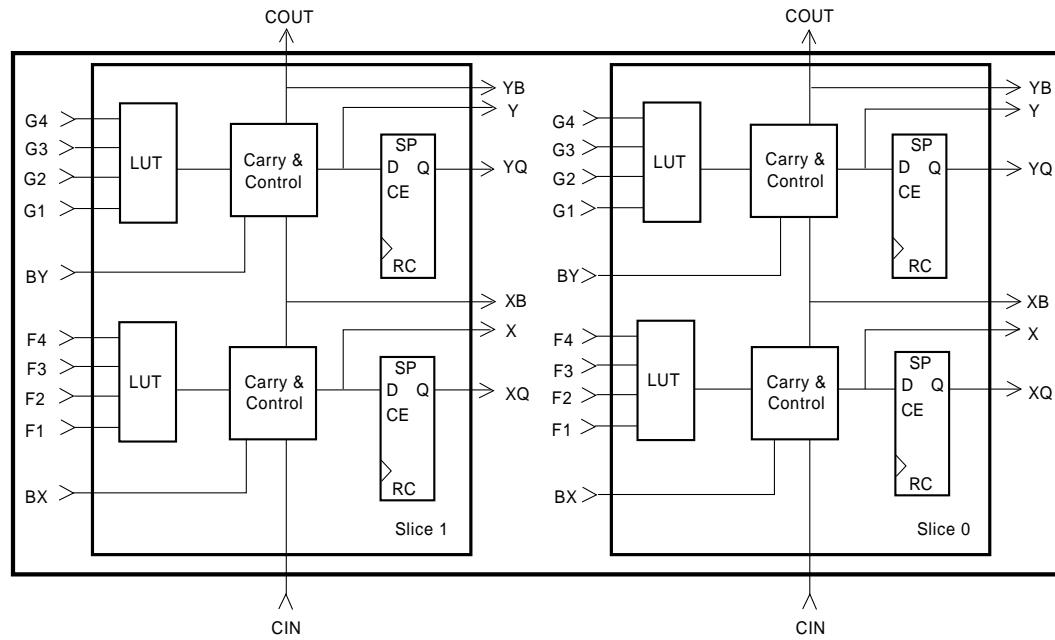
The basic building block of the Virtex-E CLB is the logic cell (LC). An LC includes a 4-input function generator, carry logic, and a storage element. The output from the function generator in each LC drives both the CLB output and the D input of the flip-flop. Each Virtex-E CLB contains four LCs, organized in two similar slices, as shown in [Figure 4](#). [Figure 5](#) shows a more detailed view of a single slice.

In addition to the four basic LCs, the Virtex-E CLB contains logic that combines function generators to provide functions of five or six inputs. Consequently, when estimating the number of system gates provided by a given device, each CLB counts as 4.5 LCs.

Look-Up Tables

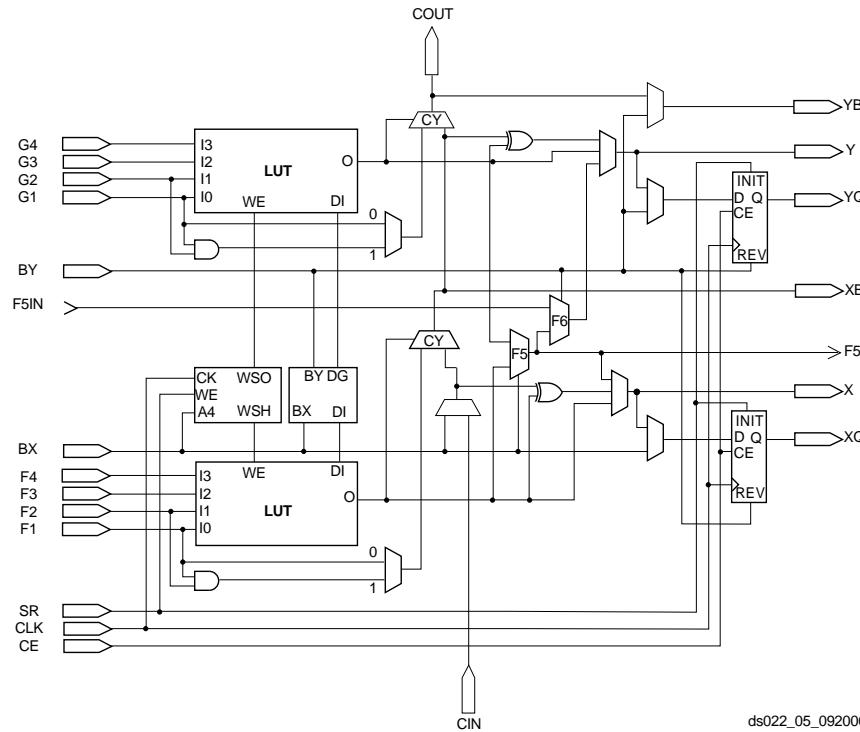
Virtex-E function generators are implemented as 4-input look-up tables (LUTs). In addition to operating as a function generator, each LUT can provide a 16 x 1-bit synchronous RAM. Furthermore, the two LUTs within a slice can be combined to create a 16 x 2-bit or 32 x 1-bit synchronous RAM, or a 16 x 1-bit dual-port synchronous RAM.

The Virtex-E LUT can also provide a 16-bit shift register that is ideal for capturing high-speed or burst-mode data. This mode can also be used to store data in applications such as Digital Signal Processing.



ds022_04_121799

Figure 4: 2-Slice Virtex-E CLB



ds022_05_092000

Figure 5: Detailed View of Virtex-E Slice

Storage Elements

The storage elements in the Virtex-E slice can be configured either as edge-triggered D-type flip-flops or as level-sensitive latches. The D inputs can be driven either by

the function generators within the slice or directly from slice inputs, bypassing the function generators.

In addition to Clock and Clock Enable signals, each Slice has synchronous set and reset signals (SR and BY). SR

forces a storage element into the initialization state specified for it in the configuration. BY forces it into the opposite state. Alternatively, these signals can be configured to operate asynchronously. All of the control signals are independently invertible, and are shared by the two flip-flops within the slice.

Additional Logic

The F5 multiplexer in each slice combines the function generator outputs. This combination provides either a function generator that can implement any 5-input function, a 4:1 multiplexer, or selected functions of up to nine inputs.

Similarly, the F6 multiplexer combines the outputs of all four function generators in the CLB by selecting one of the F5-multiplexer outputs. This permits the implementation of any 6-input function, an 8:1 multiplexer, or selected functions of up to 19 inputs.

Each CLB has four direct feedthrough paths, two per slice. These paths provide extra data input lines or additional local routing that does not consume logic resources.

Arithmetic Logic

Dedicated carry logic provides fast arithmetic carry capability for high-speed arithmetic functions. The Virtex-E CLB supports two separate carry chains, one per Slice. The height of the carry chains is two bits per CLB.

The arithmetic logic includes an XOR gate that allows a 2-bit full adder to be implemented within a slice. In addition, a dedicated AND gate improves the efficiency of multiplier implementation. The dedicated carry path can also be used to cascade function generators for implementing wide logic functions.

BUFTs

Each Virtex-E CLB contains two 3-state drivers (BUFTs) that can drive on-chip busses. <Link>See “Dedicated Routing” on page 7. Each Virtex-E BUFT has an independent 3-state control pin and an independent input pin.

Block SelectRAM

Virtex-E FPGAs incorporate large block SelectRAM memories. These complement the Distributed SelectRAM memories that provide shallow RAM structures implemented in CLBs.

Block SelectRAM memory blocks are organized in columns, starting at the left (column 0) and right outside edges and inserted every 12 CLB columns (see notes for smaller devices). Each memory block is four CLBs high, and each memory column extends the full height of the chip, immediately adjacent (to the right, except for column 0) of the CLB column locations indicated in **Table 3**.

Table 3: CLB/Block RAM Column Locations

Device/Col.	0	12	24	36	48	60	72	84	96	108	120
XCV50E	Columns 0, 6, 18, & 24										
XCV100E	Columns 0, 12, 18, & 30										
XCV200E	Columns 0, 12, 30, & 42										
XCV300E	✓	✓		✓	✓						
XCV400E	✓	✓			✓	✓					
XCV600E	✓	✓	✓		✓	✓	✓				
XCV1000E	✓	✓	✓				✓	✓	✓		
XCV1600E	✓	✓	✓	✓			✓	✓	✓	✓	
XCV2000E	✓	✓	✓	✓				✓	✓	✓	✓
XCV2600E	TBD										
XCV3200E	TBD										

Table 4 shows the amount of block SelectRAM memory that is available in each Virtex-E device.

Table 4: Virtex-E Block SelectRAM Amounts

Virtex-E Device	# of Blocks	Block SelectRAM Bits
XCV50E	16	65,536
XCV100E	20	81,920
XCV200E	28	114,688
XCV300E	32	131,072
XCV400E	40	163,840
XCV600E	72	294,912
XCV1000E	96	393,216
XCV1600E	144	589,824
XCV2000E	160	655,360
XCV2600E	184	753,664
XCV3200E	208	851,968

As illustrated in **Figure 6**, each block SelectRAM cell is a fully synchronous dual-ported (True Dual Port™) 4096-bit RAM with independent control signals for each port. The data widths of the two ports can be configured independently, providing built-in bus-width conversion.

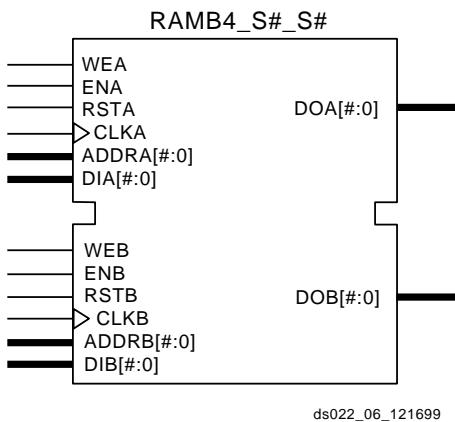


Figure 6: Dual-Port Block SelectRAM

Table 5 shows the depth and width aspect ratios for the block SelectRAM. The Virtex-E block SelectRAM also includes dedicated routing to provide an efficient interface with both CLBs and other block SelectRAMs.

Table 5: Block SelectRAM Port Aspect Ratios

Width	Depth	ADDR Bus	Data Bus
1	4096	ADDR<11:0>	DATA<0>
2	2048	ADDR<10:0>	DATA<1:0>
4	1024	ADDR<9:0>	DATA<3:0>
8	512	ADDR<8:0>	DATA<7:0>
16	256	ADDR<7:0>	DATA<15:0>

Programmable Routing Matrix

It is the longest delay path that limits the speed of any worst-case design. Consequently, the Virtex-E routing architecture and its place-and-route software were defined in a joint optimization process. This joint optimization minimizes long-path delays, and consequently, yields the best system performance.

The joint optimization also reduces design compilation times because the architecture is software-friendly. Design cycles are correspondingly reduced due to shorter design iteration times.

Local Routing

The VersaBlock provides local routing resources (see **Figure 7**), providing three types of connections:

- Interconnections among the LUTs, flip-flops, and GRM
- Internal CLB feedback paths that provide high-speed connections to LUTs within the same CLB, chaining them together with minimal routing delay
- Direct paths that provide high-speed connections between horizontally adjacent CLBs, eliminating the delay of the GRM.

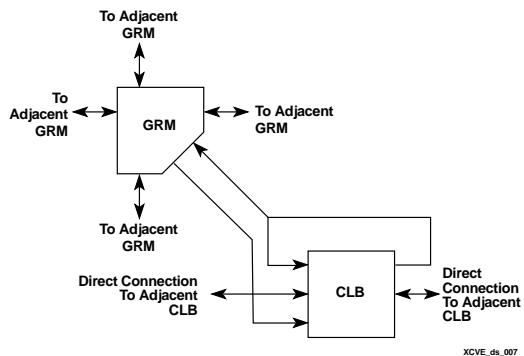


Figure 7: Virtex-E Local Routing

General Purpose Routing

Most Virtex-E signals are routed on the general purpose routing, and consequently, the majority of interconnect resources are associated with this level of the routing hierarchy. General-purpose routing resources are located in horizontal and vertical routing channels associated with the CLB rows and columns and are as follows:

- Adjacent to each CLB is a General Routing Matrix (GRM). The GRM is the switch matrix through which horizontal and vertical routing resources connect, and is also the means by which the CLB gains access to the general purpose routing.
- 24 single-length lines route GRM signals to adjacent GRMs in each of the four directions.
- 72 buffered Hex lines route GRM signals to another GRMs six-blocks away in each one of the four directions. Organized in a staggered pattern, Hex lines are driven only at their endpoints. Hex-line signals can be accessed either at the endpoints or at the midpoint (three blocks from the source). One third of the Hex lines are bidirectional, while the remaining ones are uni-directional.
- 12 Longlines are buffered, bidirectional wires that distribute signals across the device quickly and efficiently. Vertical Longlines span the full height of the device, and horizontal ones span the full width of the device.

I/O Routing

Virtex-E devices have additional routing resources around their periphery that form an interface between the CLB array and the IOBs. This additional routing, called the VersaRing, facilitates pin-swapping and pin-locking, such that logic redesigns can adapt to existing PCB layouts. Time-to-market is reduced, since PCBs and other system components can be manufactured while the logic design is still in progress.

Dedicated Routing

Some classes of signal require dedicated routing resources to maximize performance. In the Virtex-E architecture, dedicated routing resources are provided for two classes of signal.

- Horizontal routing resources are provided for on-chip 3-state busses. Four partitionable bus lines are provided per CLB row, permitting multiple busses within a row, as shown in [Figure 8](#).
- Two dedicated nets per CLB propagate carry signals vertically to the adjacent CLB. Global Clock Distribution Network
- DLL Location

Clock Routing

Clock Routing resources distribute clocks and other signals with very high fanout throughout the device. Virtex-E devices include two tiers of clock routing resources referred to as global and local clock routing resources.

- The global routing resources are four dedicated global nets with dedicated input pins that are designed to distribute high-fanout clock signals with minimal skew. Each global clock net can drive all CLB, IOB, and block RAM clock pins. The global nets can be driven only by global buffers. There are four global buffers, one for each global net.
- The local clock routing resources consist of 24 backbone lines, 12 across the top of the chip and 12 across bottom. From these lines, up to 12 unique signals per column can be distributed via the 12 longlines in the column. These local resources are more flexible than the global resources since they are not restricted to routing only to clock pins.

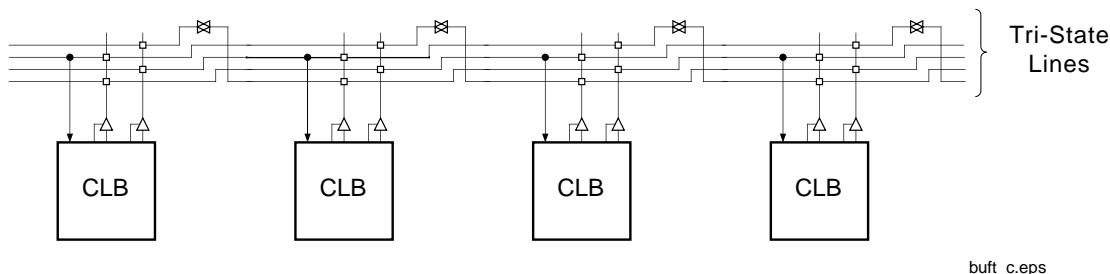


Figure 8: BUFT Connections to Dedicated Horizontal Bus Lines

Global Clock Distribution

Virtex-E provides high-speed, low-skew clock distribution through the global routing resources described above. A typical clock distribution net is shown in [Figure 9](#).

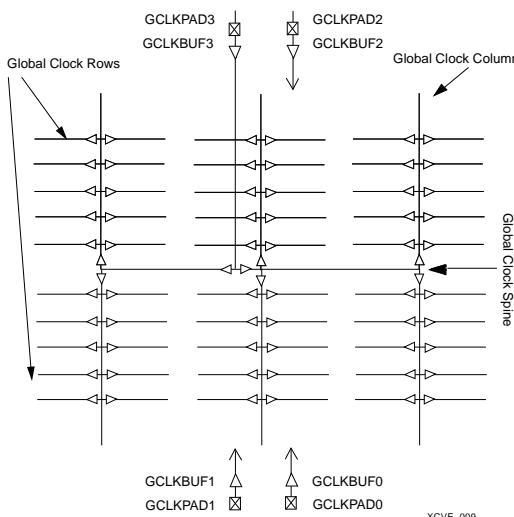


Figure 9: Global Clock Distribution Network

Four global buffers are provided, two at the top center of the device and two at the bottom center. These drive the four global nets that in turn drive any clock pin.

Four dedicated clock pads are provided, one adjacent to each of the global buffers. The input to the global buffer is selected either from these pads or from signals in the general purpose routing.

Digital Delay-Locked Loops

There are eight DLLs (Delay-Locked Loops) per device, with four located at the top and four at the bottom, [Figure 10](#). The DLLs can be used to eliminate skew between the clock input pad and the internal clock input pins throughout the device. Each DLL can drive two global clock networks. The DLL monitors the input clock and the distributed clock, and automatically adjusts a clock delay element. Additional delay is introduced such that clock edges arrive at internal flip-flops synchronized with clock edges arriving at the input.

In addition to eliminating clock-distribution delay, the DLL provides advanced control of multiple clock domains. The DLL provides four quadrature phases of the source clock, and can double the clock or divide the clock by 1.5, 2, 2.5, 3, 4, 5, 8, or 16.

The DLL also operates as a clock mirror. By driving the output from a DLL off-chip and then back on again, the DLL can be used to de-skew a board level clock among multiple devices.

To guarantee that the system clock is operating correctly prior to the FPGA starting up after configuration, the DLL can delay the completion of the configuration process until after it has achieved lock. For more information about DLL functionality, see the Design Consideration section of the data sheet.

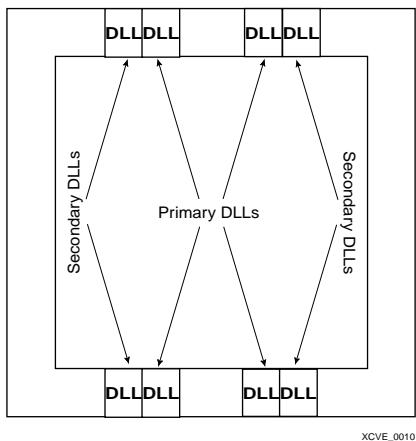


Figure 10: DLL Locations

Boundary Scan

Virtex-E devices support all the mandatory boundary-scan instructions specified in the IEEE standard 1149.1. A Test Access Port (TAP) and registers are provided that implement the EXTEST, INTEST, SAMPLE/PRELOAD, BYPASS, IDCODE, USERCODE, and HIGHZ instructions. The TAP

also supports two internal scan chains and configuration/readback of the device.

The JTAG input pins (TDI, TMS, TCK) do not have a V_{CCO} requirement and operate with either 2.5 V or 3.3 V input signalling levels. The output pin (TDO) is sourced from the V_{CCO} in bank 2, and for proper operation of LVTTL 3.3 V levels, the bank should be supplied with 3.3 V.

Boundary-scan operation is independent of individual IOB configurations, and unaffected by package type. All IOBs, including un-bonded ones, are treated as independent 3-state bidirectional pins in a single scan chain. Retention of the bidirectional test capability after configuration facilitates the testing of external interconnections.

Table 6 lists the boundary-scan instructions supported in Virtex-E FPGAs. Internal signals can be captured during EXTEST by connecting them to un-bonded or unused IOBs. They can also be connected to the unused outputs of IOBs defined as unidirectional input pins.

Before the device is configured, all instructions except USER1 and USER2 are available. After configuration, all instructions are available. During configuration, it is recommended that those operations using the boundary-scan register (SAMPLE/PRELOAD, INTEST, EXTEST) not be performed.

In addition to the test instructions outlined above, the boundary-scan circuitry can be used to configure the FPGA, and also to read back the configuration data.

Figure 11 is a diagram of the Virtex-E Series boundary scan logic. It includes three bits of Data Register per IOB, the IEEE 1149.1 Test Access Port controller, and the Instruction Register with decodes.

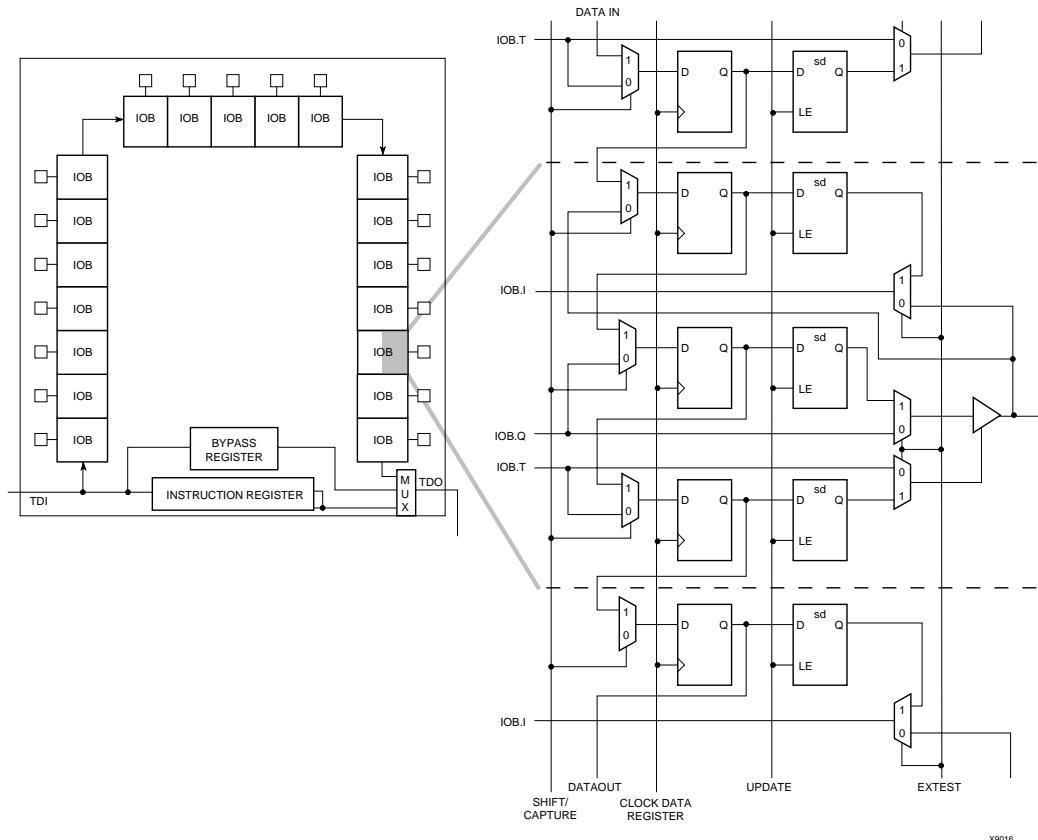


Figure 11: Virtex-E Family Boundary Scan Logic

Instruction Set

The Virtex-E Series boundary scan instruction set also includes instructions to configure the device and read back configuration data (CFG_IN, CFG_OUT, and JSTART). The complete instruction set is coded as shown in [Table 6](#).

Table 6: Boundary Scan Instructions

Boundary-Scan Command	Binary Code(4:0)	Description
EXTEST	00000	Enables boundary-scan EXTEST operation
SAMPLE/ PRELOAD	00001	Enables boundary-scan SAMPLE/PRELOAD operation
USER1	00010	Access user-defined register 1
USER2	00011	Access user-defined register 2
CFG_OUT	00100	Access the configuration bus for read operations.

Table 6: Boundary Scan Instructions (Continued)

Boundary-Scan Command	Binary Code(4:0)	Description
CFG_IN	00101	Access the configuration bus for write operations.
INTEST	00111	Enables boundary-scan INTEST operation
USERCODE	01000	Enables shifting out USER code
IDCODE	01001	Enables shifting out of ID Code
HIGHZ	01010	3-states output pins while enabling the Bypass Register
JSTART	01100	Clock the start-up sequence when StartupClk is TCK
BYPASS	11111	Enables BYPASS
RESERVED	All other codes	Xilinx reserved instructions

Data Registers

The primary data register is the boundary scan register. For each IOB pin in the FPGA, bonded or not, it includes three bits for In, Out, and 3-State Control. Non-IOB pins have appropriate partial bit population if input-only or output-only. Each EXTEST CAPTURED-OR state captures all In, Out, and 3-state pins.

The other standard data register is the single flip-flop BYPASS register. It synchronizes data being passed through the FPGA to the next downstream boundary scan device.

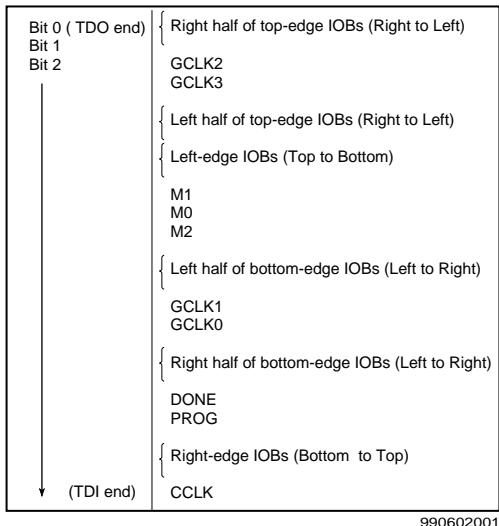
The FPGA supports up to two additional internal scan chains that can be specified using the BSCAN macro. The macro provides two user pins (SEL1 and SEL2) which are decodes of the USER1 and USER2 instructions respectively. For these instructions, two corresponding pins (TDO1 and TDO2) allow user scan data to be shifted out of TDO.

Likewise, there are individual clock pins (DRCK1 and DRCK2) for each user register. There is a common input pin (TDI) and shared output pins that represent the state of the TAP controller (RESET, SHIFT, and UPDATE).

Bit Sequence

The order within each IOB is: In, Out, 3-State. The input-only pins contribute only the In bit to the boundary scan I/O data register, while the output-only pins contributes all three bits.

From a cavity-up view of the chip (as shown in EPIC), starting in the upper right chip corner, the boundary scan data-register bits are ordered as shown in [Figure 12](#).



[Figure 12: Boundary Scan Bit Sequence](#)

BSDL (Boundary Scan Description Language) files for Virtex-E Series devices are available on the Xilinx web site in the File Download area.

Identification Registers

The IDCODE register is supported. By using the IDCODE, the device connected to the JTAG port can be determined.

The IDCODE register has the following binary format:

vvv:ffff:fffa:aaaa:aaaa:cccc:cccc:ccc1

where

v = the die version number

f = the family code (05 for Virtex-E family)

a = the number of CLB rows (ranges from 16 for XCV50E to 104 for XCV3200E)

c = the company code (49h for Xilinx)

The USERCODE register is supported. By using the USERCODE, a user-programmable identification code can be loaded and shifted out for examination. The identification code (see [Table 7](#)) is embedded in the bitstream during bit-stream generation and is valid only after configuration.

[Table 7: IDCODEs Assigned to Virtex-E FPGAs](#)

FPGA	IDCODE
XCV50E	v0A10093h
XCV100E	v0A14093h
XCV200E	v0A1C093h
XCV300E	v0A20093h
XCV400E	v0A28093h
XCV600E	v0A30093h
XCV1000E	v0A40093h
XCV1600E	v0A48093h
XCV2000E	v0A50093h
XCV2600E	v0A5C093h
XCV3200E	v0A68093h

Including Boundary Scan in a Design

Since the boundary scan pins are dedicated, no special element needs to be added to the design unless an internal data register (USER1 or USER2) is desired.

If an internal data register is used, insert the boundary scan symbol and connect the necessary pins as appropriate.

Development System

Virtex-E FPGAs are supported by the Xilinx Foundation and Alliance Series CAE tools. The basic methodology for Virtex-E design consists of three interrelated steps: design entry, implementation, and verification. Industry-standard tools are used for design entry and simulation (for example, Synopsys FPGA Express), while Xilinx provides proprietary architecture-specific tools for implementation.

The Xilinx development system is integrated under the Xilinx Design Manager (XDM™) software, providing designers with a common user interface regardless of their choice of entry and verification tools. The XDM software simplifies the selection of implementation options with pull-down menus and on-line help.

Application programs ranging from schematic capture to Placement and Routing (PAR) can be accessed through the XDM software. The program command sequence is generated prior to execution, and stored for documentation.

Several advanced software features facilitate Virtex-E design. RPMs, for example, are schematic-based macros with relative location constraints to guide their placement. They help ensure optimal implementation of common functions.

For HDL design entry, the Xilinx FPGA Foundation development system provides interfaces to the following synthesis design environments.

- Synopsys (FPGA Compiler, FPGA Express)
- Exemplar (Spectrum)
- Synplicity (Synplify)

For schematic design entry, the Xilinx FPGA Foundation and Alliance development system provides interfaces to the following schematic-capture design environments.

- Mentor Graphics V8 (Design Architect, QuickSim II)
- Viewlogic Systems (Viewdraw)

Third-party vendors support many other environments.

A standard interface-file specification, Electronic Design Interchange Format (EDIF), simplifies file transfers into and out of the development system.

Virtex-E FPGAs are supported by a unified library of standard functions. This library contains over 400 primitives and macros, ranging from 2-input AND gates to 16-bit accumulators, and includes arithmetic functions, comparators, counters, data registers, decoders, encoders, I/O functions, latches, Boolean functions, multiplexers, shift registers, and barrel shifters.

The “soft macro” portion of the library contains detailed descriptions of common logic functions, but does not contain any partitioning or placement information. The performance of these macros depends, therefore, on the partitioning and placement obtained during implementation.

RPMs, on the other hand, do contain predetermined partitioning and placement information that permits optimal implementation of these functions. Users can create their own library of soft macros or RPMs based on the macros and primitives in the standard library.

The design environment supports hierarchical design entry, with high-level schematics that comprise major functional blocks, while lower-level schematics define the logic in these blocks. These hierarchical design elements are automatically combined by the implementation tools. Different design entry tools can be combined within a hierarchical design, thus allowing the most convenient entry method to be used for each portion of the design.

Design Implementation

The place-and-route tools (PAR) automatically provide the implementation flow described in this section. The partitioner takes the EDIF net list for the design and maps the logic into the architectural resources of the FPGA (CLBs and IOBs, for example). The placer then determines the best locations for these blocks based on their interconnections and the desired performance. Finally, the router interconnects the blocks.

The PAR algorithms support fully automatic implementation of most designs. For demanding applications, however, the user can exercise various degrees of control over the process. User partitioning, placement, and routing information is optionally specified during the design-entry process. The implementation of highly structured designs can benefit greatly from basic floor planning.

The implementation software incorporates Timing Wizard® timing-driven placement and routing. Designers specify timing requirements along entire paths during design entry. The timing path analysis routines in PAR then recognize these user-specified requirements and accommodate them.

Timing requirements are entered on a schematic in a form directly relating to the system requirements, such as the targeted clock frequency, or the maximum allowable delay between two registers. In this way, the overall performance of the system along entire signal paths is automatically tailored to user-generated specifications. Specific timing information for individual nets is unnecessary.

Design Verification

In addition to conventional software simulation, FPGA users can use in-circuit debugging techniques. Because Xilinx devices are infinitely reprogrammable, designs can be verified in real time without the need for extensive sets of software simulation vectors.

The development system supports both software simulation and in-circuit debugging techniques. For simulation, the system extracts the post-layout timing information from the design database, and back-annotates this information into the net list for use by the simulator. Alternatively, the user

can verify timing-critical portions of the design using the TRCE® static timing analyzer.

For in-circuit debugging, an optional download and read-back cable is available. This cable connects the FPGA in the target system to a PC or workstation. After downloading the design into the FPGA, the designer can single-step the logic, readback the contents of the flip-flops, and so observe the internal logic state. Simple modifications can be downloaded into the system in a matter of minutes.

Configuration

Virtex-E devices are configured by loading configuration data into the internal configuration memory. Some of the pins used for this are dedicated configuration pins, while others can be re-used as general purpose inputs and outputs once configuration is complete.

The dedicated pins are the mode pins (M2, M1, M0), the configuration clock pin (CCLK), the $\overline{\text{INIT}}$ pin, the DONE pin and the boundary-scan pins (TDI, TDO, TMS, TCK). Depending on the configuration mode chosen, CCLK can be an output generated by the FPGA, or can be generated externally and provided to the FPGA as an input.

For correct operation, these pins require a V_{CCO} of 3.3 V or 2.5 V. At 3.3 V the pins operate as LVTTL, and at 2.5 V they operate as LVC MOS. All affected pins fall in banks 2 or 3.

Configuration Modes

Virtex-E supports the following four configuration modes.

- Slave-serial mode
- Master-serial mode
- SelectMAP mode
- Boundary-scan mode (JTAG)

The Configuration mode pins (M2, M1, M0) select among these configuration modes with the option in each case of having the IOB pins either pulled up or left floating prior to configuration. The selection codes are listed in [Table 8](#).

Configuration through the boundary-scan port is always available, independent of the mode selection. Selecting the boundary-scan mode simply turns off the other modes. The three mode pins have internal pull-up resistors, and default to a logic High if left unconnected.

Table 8: Configuration Codes

Configuration Mode	M2	M1	M0	CCLK Direction	Data Width	Serial D_{out}	Configuration Pull-ups
Master-serial mode	0	0	0	Out	1	Yes	No
Boundary-scan mode	1	0	1	N/A	1	No	No
SelectMAP mode	1	1	0	In	8	No	No
Slave-serial mode	1	1	1	In	1	Yes	No
Master-serial mode	1	0	0	Out	1	Yes	Yes
Boundary-scan mode	0	0	1	N/A	1	No	Yes
SelectMAP mode	0	1	0	In	8	No	Yes
Slave-serial mode	0	1	1	In	1	Yes	Yes

Table 9 lists the total number of bits required to configure each device.

Table 9: Virtex-E Bitstream Lengths

Device	# of Configuration Bits
XCV50E	630,048
XCV100E	863,840
XCV200E	1,442,016
XCV300E	1,875,648
XCV400E	2,693,440
XCV600E	3,961,632
XCV1000E	6,587,520
XCV1600E	8,308,992
XCV2000E	10,159,648
XCV2600E	12,922,336
XCV3200E	16,283,712

Slave Serial Mode

In slave serial mode, the FPGA receives configuration data in bit-serial form from a serial PROM or other source of serial configuration data. The serial bitstream must be setup

at the DIN input pin a short time before each rising edge of an externally generated CCLK.

For more information on serial PROMs, see the PROM data sheet at <http://www.xilinx.com/partinfo/ds026.pdf>.

Multiple FPGAs can be daisy-chained for configuration from a single source. After a particular FPGA has been configured, the data for the next device is routed to the DOUT pin. The data on the DOUT pin changes on the rising edge of CCLK.

The change of DOUT on the rising edge of CCLK differs from previous families, but does not cause a problem for mixed configuration chains. This change was made to improve serial-configuration rates for Virtex and Virtex-E only chains.

Figure 13 shows a full master/slave system. A Virtex-E device in slave serial mode should be connected as shown in the third device from the left

Slave-serial mode is selected by applying <111> or <011> to the mode pins (M2, M1, M0). A weak pull-up on the mode pins makes slave serial the default mode if the pins are left unconnected. **Figure 14** shows slave-serial configuration timing.

Table 10 provides more detail about the characteristics shown in **Figure 14**. Configuration must be delayed until the INIT pins of all daisy-chained FPGAs are High.

Table 10: Master/Slave Serial Mode Programming Switching

	Description	Figure 14 References	Symbol	Values	Units
CCLK	DIN setup/hold, slave mode	1/2	T_{DCC}/T_{CCD}	5.0 / 0.0	ns, min
	DIN setup/hold, master mode	1/2	T_{DSCK}/T_{SCKD}	5.0 / 0.0	ns, min
	DOUT	3	T_{CCO}	12.0	ns, max
	High time	4	T_{CCH}	5.0	ns, min
	Low time	5	T_{CCL}	5.0	ns, min
	Maximum Frequency		F_{cc}	66	MHz, max
	Frequency Tolerance, master mode with respect to nominal			+45% –30%	

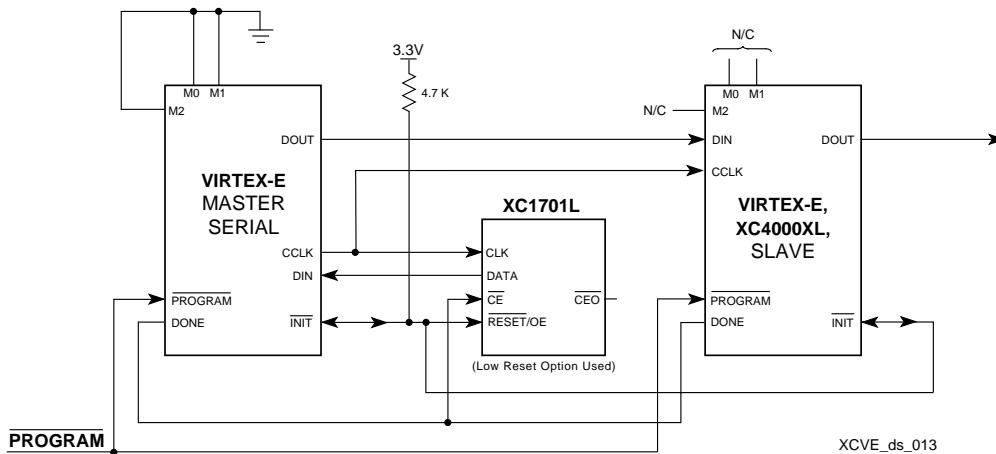


Figure 13: Master/Slave Serial Mode Circuit Diagram

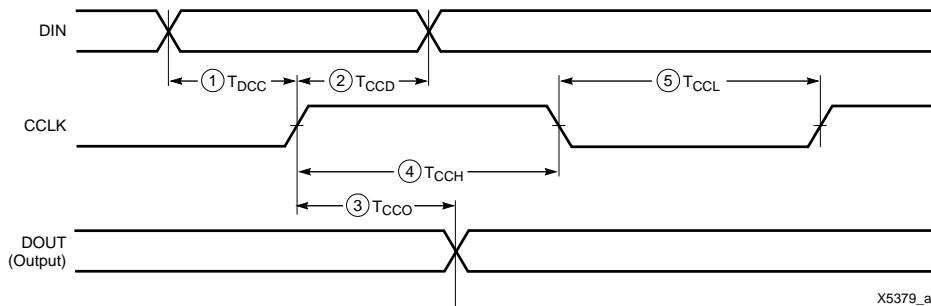


Figure 14: Slave Serial Mode Programming Switching Characteristics

Master Serial Mode

In master serial mode, the CCLK output of the FPGA drives a Xilinx Serial PROM that feeds bit-serial data to the DIN input. The FPGA accepts this data on each rising CCLK edge. After the FPGA has been loaded, the data for the next device in a daisy-chain is presented on the DOUT pin after the rising CCLK edge.

The interface is identical to slave serial except that an internal oscillator is used to generate the configuration clock (CCLK). A wide range of frequencies can be selected for CCLK which always starts at a slow default frequency. Configuration bits then switch CCLK to a higher frequency for the remainder of the configuration. Switching to a lower frequency is prohibited.

The CCLK frequency is set using the ConfigRate option in the bitstream generation software. The maximum CCLK frequency that can be selected is 60 MHz. When selecting a CCLK frequency, ensure that the serial PROM and any daisy-chained FPGAs are fast enough to support the clock rate.

On power-up, the CCLK frequency is approximately 2.5 MHz. This frequency is used until the ConfigRate bits have been loaded when the frequency changes to the selected ConfigRate. Unless a different frequency is specified in the design, the default ConfigRate is 4 MHz.

In a full master/slave system (Figure 13), the left-most device operates in master-serial mode. The remaining devices operate in slave-serial mode. The SPROM RESET pin is driven by INIT, and the CE input is driven by DONE. There is the potential for contention on the DONE pin, depending on the start-up sequence options chosen.

The sequence of operations necessary to configure a Virtex-E FPGA serially appears in Figure 15.

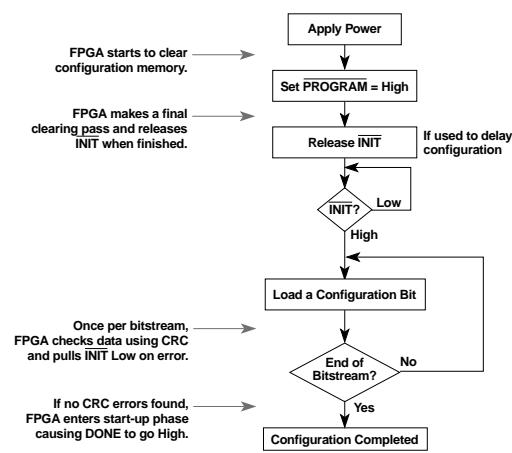


Figure 15: Serial Configuration Flowchart

Figure 16 shows the timing of master-serial configuration. Master serial mode is selected by a <000> or <100> on the mode pins (M2, M1, M0). **Table 10** shows the timing information for Figure 16.

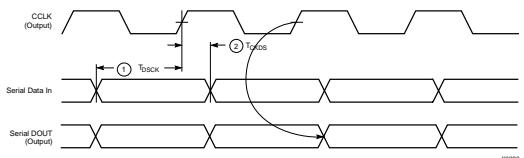


Figure 16: Master Serial Mode Programming Switching Characteristics

At power-up, V_{CC} must rise from 1.0 V to V_{CC} Min in less than 50 ms, otherwise delay configuration by pulling PROGRAM Low until V_{CC} is valid.

SelectMAP Mode

The SelectMAP mode is the fastest configuration option. Byte-wide data is written into the FPGA with a BUSY flag controlling the flow of data.

An external data source provides a byte stream, CCLK, a Chip Select (\overline{CS}) signal and a Write signal (\overline{WRITE}). If BUSY is asserted (High) by the FPGA, the data must be held until BUSY goes Low.

Data can also be read using the SelectMAP mode. If \overline{WRITE} is not asserted, configuration data is read out of the FPGA as part of a readback operation.

After configuration, the pins of the SelectMAP port can be used as additional user I/O. Alternatively, the port can be retained to permit high-speed 8-bit readback.

Retention of the SelectMAP port is selectable on a design-by-design basis when the bitstream is generated. If retention is selected, PROHIBIT constraints are required to

prevent the SelectMAP-port pins from being used as user I/O.

Multiple Virtex-E FPGAs can be configured using the SelectMAP mode, and be made to start-up simultaneously. To configure multiple devices in this way, wire the individual CCLK, Data, WRITE, and BUSY pins of all the devices in parallel. The individual devices are loaded separately by asserting the CS pin of each device in turn and writing the appropriate data. see **Table 11** for SelectMAP Write Timing Characteristics.

Write

Write operations send packets of configuration data into the FPGA. The sequence of operations for a multi-cycle write operation is shown below. Note that a configuration packet can be split into many such sequences. The packet does not have to complete within one assertion of \overline{CS} , illustrated in **Figure 17**.

1. Assert \overline{WRITE} and \overline{CS} Low. Note that when \overline{CS} is asserted on successive CCLKs, \overline{WRITE} must remain either asserted or de-asserted. Otherwise, an abort is initiated, as described below.
2. Drive data onto D[7:0]. Note that to avoid contention, the data source should not be enabled while \overline{CS} is Low and \overline{WRITE} is High. Similarly, while \overline{WRITE} is High, no more than one \overline{CS} should be asserted.
3. At the rising edge of CCLK: If BUSY is Low, the data is accepted on this clock. If BUSY is High (from a previous write), the data is not accepted. Acceptance instead occurs on the first clock after BUSY goes Low, and the data must be held until this has happened.
4. Repeat steps 2 and 3 until all the data has been sent.
5. De-assert \overline{CS} and \overline{WRITE} .

Table 11: SelectMAP Write Timing Characteristics

	Description		Symbol		Units
CCLK	D ₀₋₇ Setup/Hold	1/2	T_{SMDCC}/T_{SMCCD}	5.0 / 1.0	ns, min
	CS Setup/Hold	3/4	T_{SMCSCC}/T_{SMCCCS}	7.0 / 1.0	ns, min
	WRITE Setup/Hold	5/6	T_{SMCCW}/T_{SMWCC}	7.0 / 1.0	ns, min
	BUSY Propagation Delay	7	T_{SMCKBY}	12.0	ns, max
	Maximum Frequency		F_{CC}	66	MHz, max
	Maximum Frequency with no handshake		F_{CCNH}	50	MHz, max

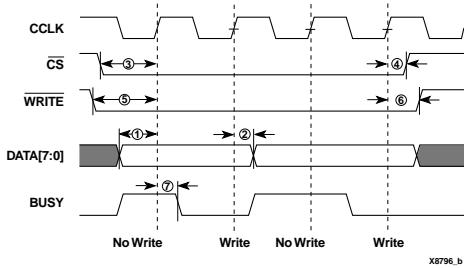


Figure 17: Write Operations

A flowchart for the write operation appears in [Figure 18](#). Note that if CCLK is slower than f_{CCNH} , the FPGA never asserts BUSY. In this case, the above handshake is unnecessary, and data can simply be entered into the FPGA every CCLK cycle.

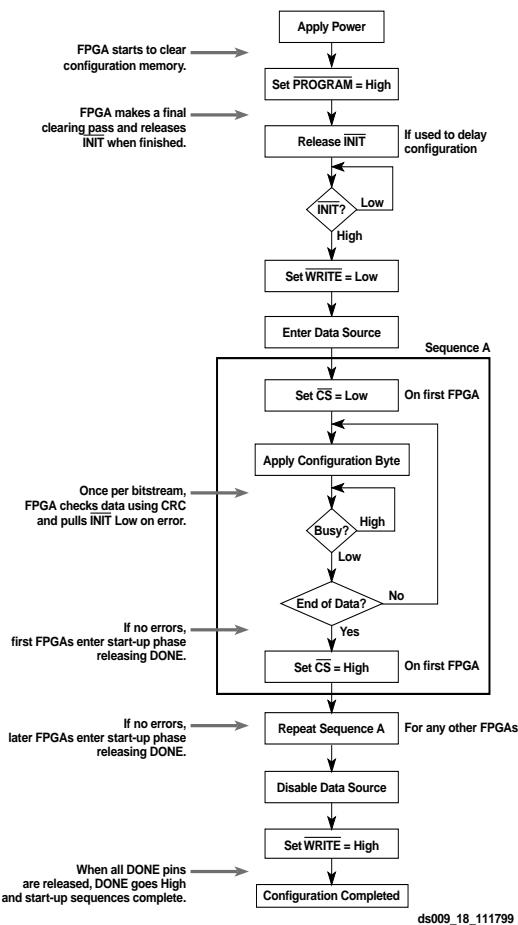


Figure 18: SelectMAP Flowchart for Write Operations

Abort

During a given assertion of CS, the user cannot switch from a write to a read, or vice-versa. This action causes the current packet command to be aborted. The device remains BUSY until the aborted operation has completed. Following an abort, data is assumed to be unaligned to word bound-

aries, and the FPGA requires a new synchronization word prior to accepting any new packets.

To initiate an abort during a write operation, de-assert WRITE. At the rising edge of CCLK, an abort is initiated, as shown in [Figure 19](#).

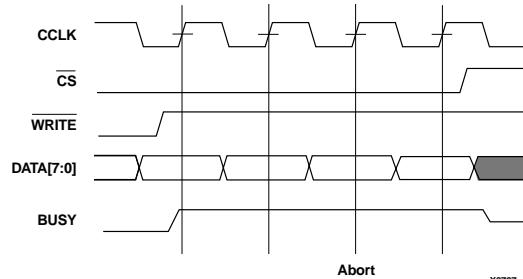


Figure 19: SelectMAP Write Abort Waveforms

Boundary-Scan Mode

In the boundary-scan mode, no non-dedicated pins are required, configuration being done entirely through the IEEE 1149.1 Test Access Port.

Configuration through the TAP uses the CFG_IN instruction. This instruction allows data input on TDI to be converted into data packets for the internal configuration bus.

The following steps are required to configure the FPGA through the boundary-scan port (when using TCK as a start-up clock).

1. Load the CFG_IN instruction into the boundary-scan instruction register (IR).
2. Enter the Shift-DR (SDR) state.
3. Shift a configuration bitstream into TDI.
4. Return to Run-Test-Idle (RTI).
5. Load the JSTART instruction into IR.
6. Enter the SDR state.
7. Clock TCK through the startup sequence.
8. Return to RTI.

Configuration and readback via the TAP is always available. The boundary-scan mode is selected by a <101> or <001> on the mode pins (M2, M1, M0).

Configuration Sequence

The configuration of Virtex-E devices is a three-phase process. First, the configuration memory is cleared. Next, configuration data is loaded into the memory, and finally, the logic is activated by a start-up process.

Configuration is automatically initiated on power-up unless it is delayed by the user, as described below. The configuration process can also be initiated by asserting PROGRAM. The end of the memory-clearing phase is signalled by INIT going High, and the completion of the entire process is signalled by DONE going High.

The power-up timing of configuration signals is shown in Figure 20.

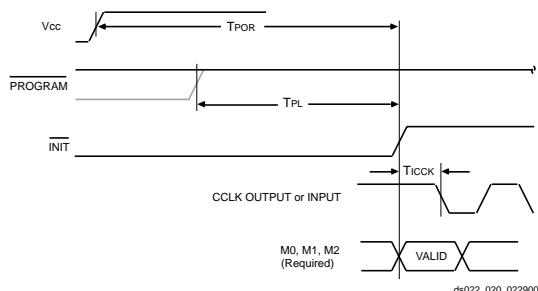


Figure 20: Power-up Timing Configuration Signals

The corresponding timing characteristics are listed in Table 12.

Table 12: Power-up Timing Characteristics

Description	Symbol	Value	Units
Power-on Reset ¹	T _{POR}	2.0	ms, max
Program Latency	T _{PL}	100.0	μs, max
CCLK (output) Delay	T _{iCCK}	0.5	μs, min
		4.0	μs, max
Program Pulse Width	T _{PROGRAM}	300	ns, min

Notes:

1. T_{POR} delay is the initialization time required after V_{CCINT} and V_{CCO} in Bank 2 reach the recommended operating voltage.

Delaying Configuration

INIT can be held Low using an open-drain driver. An open-drain is required since INIT is a bidirectional open-drain pin that is held Low by the FPGA while the configuration memory is being cleared. Extending the time that the pin is Low causes the configuration sequencer to wait. Thus, configuration is delayed by preventing entry into the phase where data is loaded.

Start-Up Sequence

The default Start-up sequence is that one CCLK cycle after DONE goes High, the global 3-state signal (GTS) is released. This permits device outputs to turn on as necessary.

One CCLK cycle later, the Global Set/Reset (GSR) and Global Write Enable (GWE) signals are released. This permits the internal storage elements to begin changing state in response to the logic and the user clock.

The relative timing of these events can be changed. In addition, the GTS, GSR, and GWE events can be made dependent on the DONE pins of multiple devices all going High, forcing the devices to start synchronously. The sequence can also be paused at any stage until lock has been achieved on any or all DLLs.

Readback

The configuration data stored in the Virtex-E configuration memory can be readback for verification. Along with the configuration data it is possible to readback the contents all flip-flops/latches, LUT RAMs, and block RAMs. This capability is used for real-time debugging. For more detailed information, see application note XAPP138 "Virtex FPGA Series Configuration and Readback".

Design Considerations

This section contains more detailed design information on the following features.

- Delay-Locked Loop . . . see [page 18](#)
- BlockRAM . . . see [page 22](#)
- SelectI/O . . . see [page 28](#)

Using DLLs

The Virtex-E FPGA series provides up to eight fully digital dedicated on-chip Delay-Locked Loop (DLL) circuits which provide zero propagation delay, low clock skew between output clock signals distributed throughout the device, and advanced clock domain control. These dedicated DLLs can be used to implement several circuits which improve and simplify system level design.

Introduction

As FPGAs grow in size, quality on-chip clock distribution becomes increasingly important. Clock skew and clock delay impact device performance and the task of managing clock skew and clock delay with conventional clock trees becomes more difficult in large devices. The Virtex-E series of devices resolve this potential problem by providing up to eight fully digital dedicated on-chip DLL circuits, which provide zero propagation delay and low clock skew between output clock signals distributed throughout the device.

Each DLL can drive up to two global clock routing networks within the device. The global clock distribution network minimizes clock skews due to loading differences. By monitoring a sample of the DLL output clock, the DLL can compensate for the delay on the routing network, effectively eliminating the delay from the external input port to the individual clock loads within the device.

In addition to providing zero delay with respect to a user source clock, the DLL can provide multiple phases of the source clock. The DLL can also act as a clock doubler or it can divide the user source clock by up to 16.

Clock multiplication gives the designer a number of design alternatives. For instance, a 50 MHz source clock doubled by the DLL can drive an FPGA design operating at 100 MHz. This technique can simplify board design because the clock path on the board no longer distributes such a high-speed signal. A multiplied clock also provides designers the option of time-domain-multiplexing, using one circuit twice per clock cycle, consuming less area than two copies of the same circuit. Two DLLs in can be connected in series to increase the effective clock multiplication factor to four.

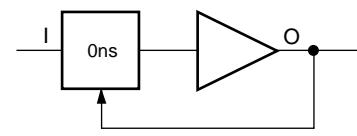
The DLL can also act as a clock mirror. By driving the DLL output off-chip and then back in again, the DLL can be used to de-skew a board level clock between multiple devices.

In order to guarantee the system clock establishes prior to the device “waking up,” the DLL can delay the completion of the device configuration process until after the DLL achieves lock.

By taking advantage of the DLL to remove on-chip clock delay, the designer can greatly simplify and improve system level design involving high-fanout, high-performance clocks.

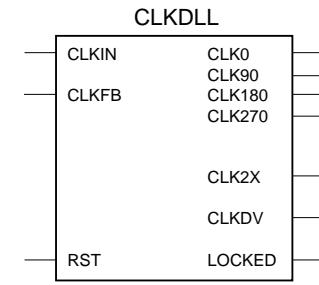
Library DLL Symbols

[Figure 21](#) shows the simplified Xilinx library DLL macro symbol, BUFGDLL. This macro delivers a quick and efficient way to provide a system clock with zero propagation delay throughout the device. [Figure 22](#) and [Figure 23](#) show the two library DLL primitives. These symbols provide access to the complete set of DLL features when implementing more complex applications.



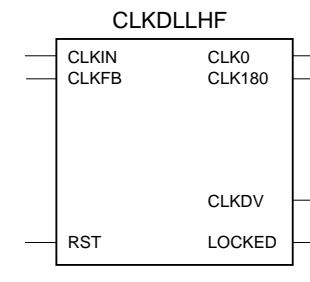
ds022_25_121099

Figure 21: Simplified DLL Macro Symbol BUFGDLL



ds022_26_121099

Figure 22: Standard DLL Symbol CLKDLL



ds022_027_121099

Figure 23: High Frequency DLL Symbol CLKDLLHF

BUFGDLL Pin Descriptions

Use the BUFGDLL macro as the simplest way to provide zero propagation delay for a high-fanout on-chip clock from an external input. This macro uses the IBUFG, CLKDLL and BUFG primitives to implement the most basic DLL application as shown in Figure 24.

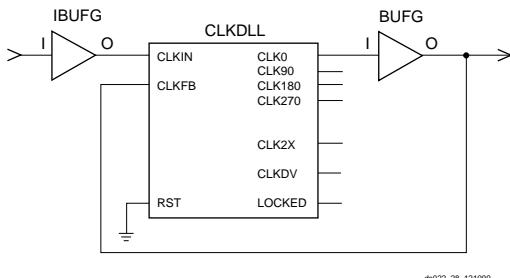


Figure 24: BUFGDLL Schematic

This symbol does not provide access to the advanced clock domain controls or to the clock multiplication or clock division features of the DLL. This symbol also does not provide access to the RST, or LOCKED pins of the DLL. For access to these features, a designer must use the library DLL primitives described in the following sections.

Source Clock Input — I

The I pin provides the user source clock, the clock signal on which the DLL operates, to the BUFGDLL. For the BUFGDLL macro the source clock frequency must fall in the low frequency range as specified in the data sheet. The BUFGDLL requires an external signal source clock. Therefore, only an external input port can source the signal that drives the BUFGDLL I pin.

Clock Output — O

The clock output pin O represents a delay-compensated version of the source clock (I) signal. This signal, sourced by a global clock buffer BUFG symbol, takes advantage of the dedicated global clock routing resources of the device.

The output clock has a 50-50 duty cycle unless you deactivate the duty cycle correction property.

CLKDLL Primitive Pin Descriptions

The library CLKDLL primitives provide access to the complete set of DLL features needed when implementing more complex applications with the DLL.

Source Clock Input — CLKIN

The CLKIN pin provides the user source clock (the clock signal on which the DLL operates) to the DLL. The CLKIN frequency must fall in the ranges specified in the data sheet. A global clock buffer (BUFG) driven from another CLKDLL, one of the global clock input buffers (IBUFG), or an IO_LVDS_DLL pin on the same edge of the device (top or bottom) must source this clock signal. There are four IO_LVDS_DLL input pins that can be used as inputs to the

DLLs. This makes a total of eight usable input pins for DLLs in the Virtex-E family.

Feedback Clock Input — CLKFB

The DLL requires a reference or feedback signal to provide the delay-compensated output. Connect only the CLK0 or CLK2X DLL outputs to the feedback clock input (CLKFB) pin to provide the necessary feedback to the DLL. The feedback clock input can also be provided through one of the following pins.

IBUFG - Global Clock Input Pad

IO_LVDS_DLL - the pin adjacent to IBUF

If an IBUFG sources the CLKFB pin, the following special rules apply.

1. An external input port must source the signal that drives the IBUFG I pin.
2. The CLK2X output must feedback to the device if both the CLK0 and CLK2X outputs are driving off chip devices.
3. That signal must directly drive only OBUs and nothing else.

These rules enable the software determine which DLL clock output sources the CLKFB pin.

Reset Input — RST

When the reset pin RST activates the LOCKED signal deactivates within four source clock cycles. The RST pin, active High, must either connect to a dynamic signal or tied to ground. As the DLL delay taps reset to zero, glitches can occur on the DLL clock output pins. Activation of the RST pin can also severely affect the duty cycle of the clock output pins. Furthermore, the DLL output clocks no longer de-skew with respect to one another. For these reasons, rarely use the reset pin unless re-configuring the device or changing the input frequency.

2x Clock Output — CLK2X

The output pin CLK2X provides a frequency-doubled clock with an automatic 50/50 duty-cycle correction. Until the CLKDLL has achieved lock, the CLK2X output appears as a 1x version of the input clock with a 25/75 duty cycle. This behavior allows the DLL to lock on the correct edge with respect to source clock. This pin is not available on the CLKDLLHF primitive.

Clock Divide Output — CLKDV

The clock divide output pin CLKDV provides a lower frequency version of the source clock. The CLKDV_DIVIDE property controls CLKDV such that the source clock is divided by N where N is either 1.5, 2, 2.5, 3, 4, 5, 8, or 16.

This feature provides automatic duty cycle correction such that the CLKDV output pin always has a 50/50 duty cycle, with the exception of noninteger divides in HF mode, where the duty cycle is 1/3 for N=1.5 and 2/5 for N=2.5.

1x Clock Outputs — CLK[0/90/180/270]

The 1x clock output pin CLK0 represents a delay-compensated version of the source clock (CLKIN) signal. The CLKDLL primitive provides three phase-shifted versions of the CLK0 signal while CLKDLLHF provides only the 180° phase-shifted version. The relationship between phase shift and the corresponding period shift appears in [Table 13](#).

Table 13: Relationship of Phase-Shifted Output Clock to Period Shift

Phase (degrees)	Period Shift (percent)
0	0%
90	25%
180	50%
270	75%

The timing diagrams in [Figure 25](#) illustrate the DLL clock output characteristics.

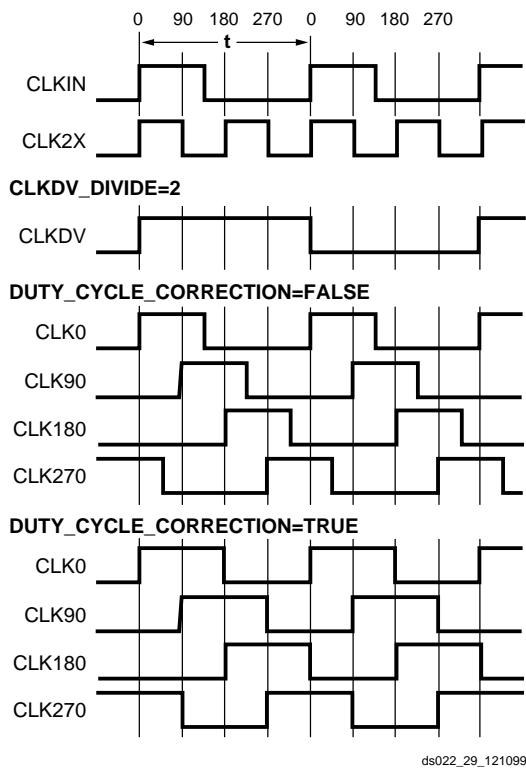


Figure 25: DLL Output Characteristics

The DLL provides duty cycle correction on all 1x clock outputs such that all 1x clock outputs by default have a 50/50 duty cycle. The DUTY_CYCLE_CORRECTION property (TRUE by default), controls this feature. In order to deactivate the DLL duty cycle correction, attach the DUTY_CYCLE_CORRECTION=False property to the DLL symbol. When duty cycle correction deactivates, the output clock has the same duty cycle as the source clock.

The DLL clock outputs can drive an OBUF, a BUFG, or they can route directly to destination clock pins. The DLL clock outputs can only drive the BUFGs that reside on the same edge (top or bottom).

Locked Output — LOCKED

To achieve lock, the DLL might need to sample several thousand clock cycles. After the DLL achieves lock, the LOCKED signal activates. The DLL timing parameter section of the data sheet provides estimates for locking times.

To guarantee that the system clock is established prior to the device “waking up,” the DLL can delay the completion of the device configuration process until after the DLL locks. The STARTUP_WAIT property activates this feature.

Until the LOCKED signal activates, the DLL output clocks are not valid and can exhibit glitches, spikes, or other spurious movement. In particular the CLK2X output appears as a 1x clock with a 25/75 duty cycle.

DLL Properties

Properties provide access to some of the Virtex-E series DLL features, (for example, clock division and duty cycle correction).

Duty Cycle Correction Property

The 1x clock outputs, CLK0, CLK90, CLK180, and CLK270, use the duty-cycle corrected default, exhibiting a 50/50 duty cycle. The DUTY_CYCLE_CORRECTION property (by default TRUE) controls this feature. To deactivate the DLL duty-cycle correction for the 1x clock outputs, attach the DUTY_CYCLE_CORRECTION=False property to the DLL symbol. When duty-cycle correction deactivates, the output clock has the same duty cycle as the source clock.

Clock Divide Property

The CLKDV_DIVIDE property specifies how the signal on the CLKDV pin is frequency divided with respect to the CLK0 pin. The values allowed for this property are 1.5, 2, 2.5, 3, 4, 5, 8, or 16; the default value is 2.

Startup Delay Property

This property, STARTUP_WAIT, takes on a value of TRUE or FALSE (the default value). When TRUE the device configuration DONE signal waits until the DLL locks before going to High.

Virtex-E DLL Location Constraints

As shown in [Figure 26](#), there are four additional DLLs in the Virtex-E devices, for a total of eight per Virtex-E device. These DLLs are located in silicon, at the top and bottom of the two innermost block SelectRAM columns. The location constraint LOC, attached to the DLL symbol with the identifier DLL0S, DLL0P, DLL1S, DLL1P, DLL2S, DLL2P, DLL3S, or DLL3P, controls the DLL location.

The LOC property uses the following form:

LOC = DLL0P

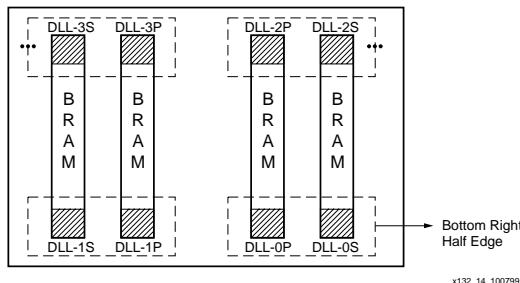


Figure 26: Virtex Series DLLs

Design Factors

Use the following design considerations to avoid pitfalls and improve success designing with Xilinx devices.

Input Clock

The output clock signal of a DLL, essentially a delayed version of the input clock signal, reflects any instability on the input clock in the output waveform. For this reason the quality of the DLL input clock relates directly to the quality of the output clock waveforms generated by the DLL. The DLL input clock requirements are specified in the data sheet.

In most systems a crystal oscillator generates the system clock. The DLL can be used with any commercially available quartz crystal oscillator. For example, most crystal oscillators produce an output waveform with a frequency tolerance of 100 PPM, meaning 0.01 percent change in the clock period. The DLL operates reliably on an input waveform with a frequency drift of up to 1 ns — orders of magnitude in excess of that needed to support any crystal oscillator in the industry. However, the cycle-to-cycle jitter must be kept to less than 300 ps in the low frequencies and 150 ps for the high frequencies.

Input Clock Changes

Changing the period of the input clock beyond the maximum drift amount requires a manual reset of the CLKDLL. Failure to reset the DLL produces an unreliable lock signal and output clock.

It is possible to stop the input clock with little impact to the DLL. Stopping the clock should be limited to less than 100 μ s to keep device cooling to a minimum. The clock should be stopped during a Low phase, and when restored the full High period should be seen. During this time, LOCKED stays High and remains High when the clock is restored.

When the clock is stopped, one to four more clocks are still observed as the delay line is flushed. When the clock is restarted, the output clocks are not observed for one to four

clocks as the delay line is filled. The most common case is two or three clocks.

In a similar manner, a phase shift of the input clock is also possible. The phase shift propagates to the output one to four clocks after the original shift, with no disruption to the CLKDLL control.

Output Clocks

As mentioned earlier in the DLL pin descriptions, some restrictions apply regarding the connectivity of the output pins. The DLL clock outputs can drive an OBUF, a global clock buffer BUFG, or they can route directly to destination clock pins. The only BUFGs that the DLL clock outputs can drive are the two on the same edge of the device (top or bottom). In addition, the CLK2X output of the secondary DLL can connect directly to the CLKIN of the primary DLL in the same quadrant.

Do not use the DLL output clock signals until after activation of the LOCKED signal. Prior to the activation of the LOCKED signal, the DLL output clocks are not valid and can exhibit glitches, spikes, or other spurious movement.

Useful Application Examples

The Virtex-E DLL can be used in a variety of creative and useful applications. The following examples show some of the more common applications. The Verilog and VHDL example files are available at:

<ftp://ftp.xilinx.com/pub/applications/xapp/xapp132.zip>

Standard Usage

The circuit shown in Figure 27 resembles the BUFGDLL macro implemented to provide access to the RST and LOCKED pins of the CLKDLL.

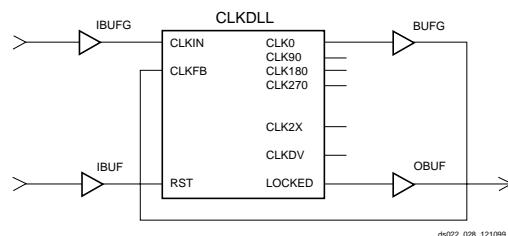


Figure 27: Standard DLL Implementation

Board Level De-skew of Multiple Non-Virtex-E Devices

The circuit shown in Figure 28 can be used to de-skew a system clock between a Virtex-E chip and other non-Virtex-E chips on the same board. This application is commonly used when the Virtex-E device is used in conjunction with other standard products such as SRAM or DRAM devices. While designing the board level route, ensure that

the return net delay to the source equals the delay to the other chips involved.

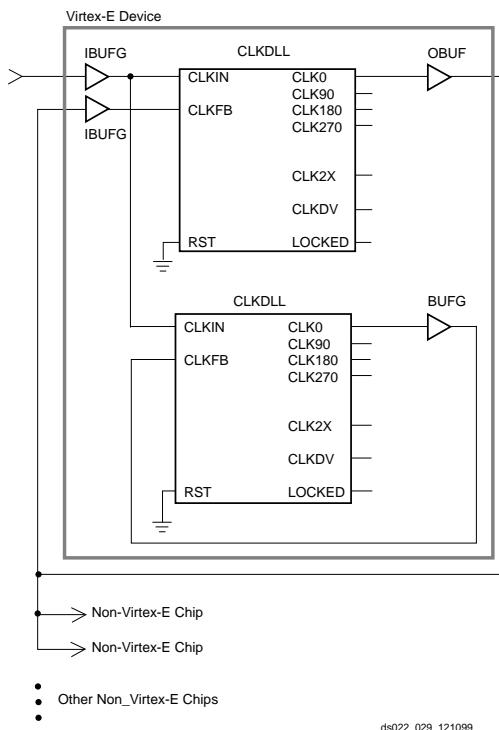


Figure 28: DLL De-skew of Board Level Clock

Board-level de-skew is not required for low-fanout clock networks. It is recommended for systems that have fanout limitations on the clock network, or if the clock distribution chip cannot handle the load.

Do not use the DLL output clock signals until after activation of the LOCKED signal. Prior to the activation of the LOCKED signal, the DLL output clocks are not valid and can exhibit glitches, spikes, or other spurious movement.

The dll_mirror_1 files in the [xapp132.zip](#) file show the VHDL and Verilog implementation of this circuit.

De-skew of Clock and Its 2x Multiple

The circuit shown in [Figure 29](#) implements a 2x clock multiplier and also uses the CLK0 clock output with a zero ns skew between registers on the same chip. Alternatively, a clock divider circuit can be implemented using similar connections.

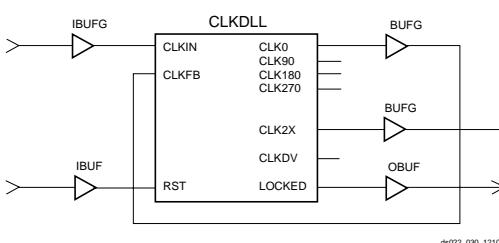


Figure 29: DLL De-skew of Clock and 2x Multiple

Because any single DLL can access only two BUFGs at most, any additional output clock signals must be routed from the DLL in this example on the high speed backbone routing.

The dll_2x files in the [xapp132.zip](#) file show the VHDL and Verilog implementation of this circuit.

Virtex-E 4x Clock

Two DLLs located in the same half-edge (top-left, top-right, bottom-right, bottom-left) can be connected together, without using a BUFG between the CLKDLLs, to generate a 4x clock as shown in [Figure 30](#). Virtex-E devices, like the Virtex devices, have four clock networks that are available for internal de-skewing of the clock. Each of the eight DLLs have access to two of the four clock networks. Although all the DLLs can be used for internal de-skewing, the presence of two GCLKBUFs on the top and two on the bottom indicate that only two of the four DLLs on the top (and two of the four DLLs on the bottom) can be used for this purpose.

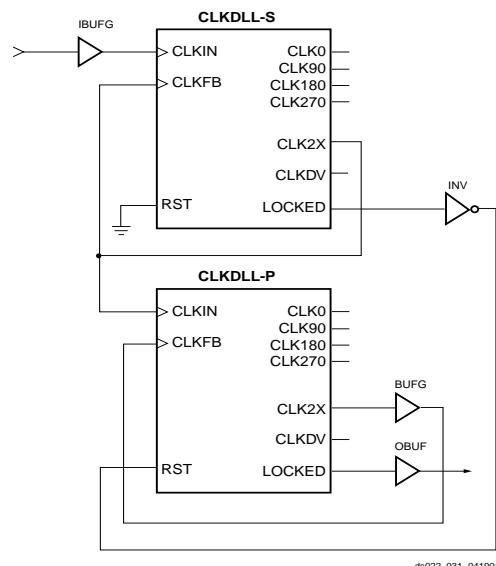


Figure 30: DLL Generation of 4x Clock in Virtex-E Devices

The dll_4xe files in the [xapp132.zip](#) file show the DLL implementation in Verilog for Virtex-E devices. These files can be found at:

<ftp://ftp.xilinx.com/pub/applications/xapp/xapp132.zip>

Using Block SelectRAM+ Features

The Virtex FPGA Series provides dedicated blocks of on-chip, true dual-read/write port synchronous RAM, with 4096 memory cells. Each port of the block SelectRAM+ memory can be independently configured as a read/write port, a read port, a write port, and can be configured to a specific data width. The block SelectRAM+ memory offers new capabilities allowing the FPGA designer to simplify designs.

Operating Modes

Vlortex-E block SelectRAM+ memory supports two operating modes:

- Read Through
- Write Back

Read Through (one clock edge)

The read address is registered on the read port clock edge and data appears on the output after the RAM access time. Some memories might place the latch/register at the outputs, depending on whether a faster clock-to-out versus set-up time is desired. This is generally considered to be an inferior solution, since it changes the read operation to an asynchronous function with the possibility of missing an address/control line transition during the generation of the read pulse clock.

Write Back (one clock edge)

The write address is registered on the write port clock edge and the data input is written to the memory and mirrored on the output.

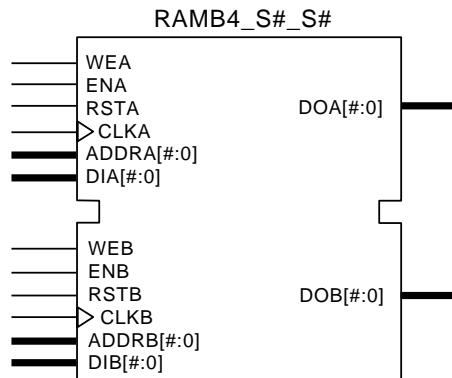
Block SelectRAM+ Characteristics

- All inputs are registered with the port clock and have a set-up to clock timing specification.
- All outputs have a read through or write back function depending on the state of the port WE pin. The outputs relative to the port clock are available after the clock-to-out timing specification.
- The block SelectRAMs are true SRAM memories and do not have a combinatorial path from the address to the output. The LUT SelectRAM+ cells in the CLBs are still available with this function.
- The ports are completely independent from each other (*i.e.*, clocking, control, address, read/write function, and data width) without arbitration.
- A write operation requires only one clock edge.
- A read operation requires only one clock edge.

The output ports are latched with a self timed circuit to guarantee a glitch free read. The state of the output port does not change until the port executes another read or write operation.

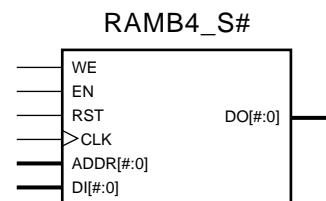
Library Primitives

[Figure 31](#) and [Figure 32](#) show the two generic library block SelectRAM+ primitives. [Table 14](#) describes all of the available primitives for synthesis and simulation.



ds022_032_121399

Figure 31: Dual-Port Block SelectRAM+ Memory



ds022_033_121399

Figure 32: Single-Port Block SelectRAM+ Memory

Table 14: Available Library Primitives

Primitive	Port A Width	Port B Width
RAMB4_S1		N/A
RAMB4_S1_S1		1
RAMB4_S1_S2		2
RAMB4_S1_S4		4
RAMB4_S1_S8		8
RAMB4_S1_S16		16
RAMB4_S2		N/A
RAMB4_S2_S2		2
RAMB4_S2_S4		4
RAMB4_S2_S8		8
RAMB4_S2_S16		16

Table 14: Available Library Primitives

Primitive	Port A Width	Port B Width
RAMB4_S4	4	N/A
RAMB4_S4_S4		4
RAMB4_S4_S8		8
RAMB4_S4_S16		16
RAMB4_S8	8	N/A
RAMB4_S8_S8		8
RAMB4_S8_S16		16
RAMB4_S16	16	N/A
RAMB4_S16_S16		16

Port Signals

Each block SelectRAM+ port operates independently of the others while accessing the same set of 4096 memory cells.

Table 15 describes the depth and width aspect ratios for the block SelectRAM+ memory.

Table 15: Block SelectRAM+ Port Aspect Ratios

Width	Depth	ADDR Bus	Data Bus
1	4096	ADDR<11:0>	DATA<0>
2	2048	ADDR<10:0>	DATA<1:0>
4	1024	ADDR<9:0>	DATA<3:0>
8	512	ADDR<8:0>	DATA<7:0>
16	256	ADDR<7:0>	DATA<15:0>

Clock—CLK[A/B]

Each port is fully synchronous with independent clock pins. All port input pins have setup time referenced to the port CLK pin. The data output bus has a clock-to-out time referenced to the CLK pin.

Enable—EN[A/B]

The enable pin affects the read, write and reset functionality of the port. Ports with an inactive enable pin keep the output pins in the previous state and do not write data to the memory cells.

Write Enable—WE[A/B]

Activating the write enable pin allows the port to write to the memory cells. When active, the contents of the data input bus are written to the RAM at the address pointed to by the address bus, and the new data also reflects on the data out bus. When inactive, a read operation occurs and the contents of the memory cells referenced by the address bus reflect on the data out bus.

Reset—RST[A/B]

The reset pin forces the data output bus latches to zero synchronously. This does not affect the memory cells of the RAM and does not disturb a write operation on the other port.

Address Bus—ADDR[A/B]<#:0>

The address bus selects the memory cells for read or write. The width of the port determines the required width of this bus as shown in Table 15.

Data In Bus—DI[A/B]<#:0>

The data in bus provides the new data value to be written into the RAM. This bus and the port have the same width, as shown in Table 15.

Data Output Bus—DO[A/B]<#:0>

The data out bus reflects the contents of the memory cells referenced by the address bus at the last active clock edge. During a write operation, the data out bus reflects the data in bus. The width of this bus equals the width of the port. The allowed widths appear in Table 15.

Inverting Control Pins

The four control pins (CLK, EN, WE and RST) for each port have independent inversion control as a configuration option.

Address Mapping

Each port accesses the same set of 4096 memory cells using an addressing scheme dependent on the width of the port.

The physical RAM location addressed for a particular width are described in the following formula (of interest only when the two ports use different aspect ratios).

$$\text{Start} = ((\text{ADDR}_{\text{port}} + 1) * \text{Width}_{\text{port}}) - 1$$

$$\text{End} = \text{ADDR}_{\text{port}} * \text{Width}_{\text{port}}$$

Table 16 shows low order address mapping for each port width.

Table 16: Port Address Mapping

Port Width	Port Addresses																	
	Addresses																	
1	4095...	1	1	1	1	1	1	0	0	0	0	0						
		5	4	3	2	1	0	9	8	7	6	5						
2	2047...	07	06	05	04	03	02	01	00									
4	1023...	03			02			01			00							
8	511...	01						00										
16	255...	00																

Creating Larger RAM Structures

The block SelectRAM+ columns have specialized routing to allow cascading blocks together with minimal routing delays. This achieves wider or deeper RAM structures with a smaller timing penalty than when using normal routing channels.

Location Constraints

Block SelectRAM+ instances can have LOC properties attached to them to constrain the placement. The block SelectRAM+ placement locations are separate from the CLB location naming convention, allowing the LOC properties to transfer easily from array to array.

The LOC properties use the following form.

$$\text{LOC} = \text{RAMB4_R}\#\text{C}\#$$

RAMB4_R0C0 is the upper left RAMB4 location on the device.

Conflict Resolution

The block SelectRAM+ memory is a true dual-read/write port RAM that allows simultaneous access of the same memory cell from both ports. When one port writes to a given memory cell, the other port must not address that

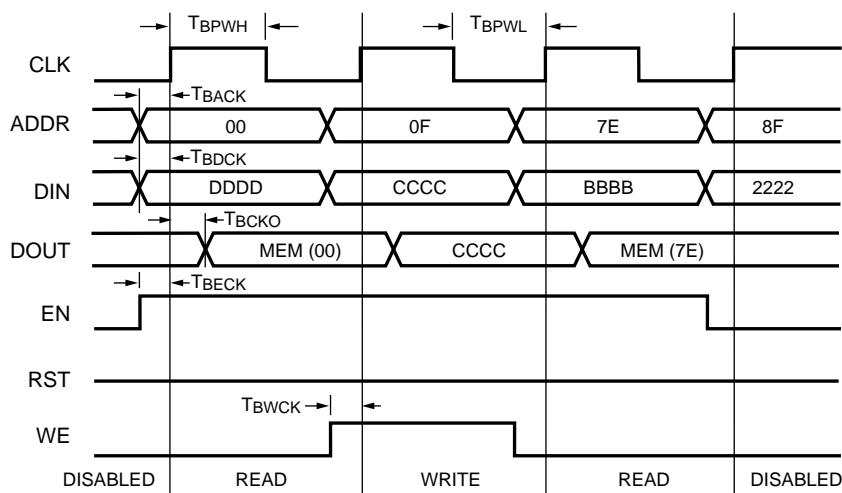
memory cell (for a write or a read) within the clock-to-clock setup window. The following lists specifics of port and memory cell write conflict resolution.

- If both ports write to the same memory cell simultaneously, violating the clock-to-clock setup requirement, consider the data stored as invalid.
- If one port attempts a read of the same memory cell the other simultaneously writes, violating the clock-to-clock setup requirement, the following occurs.
 - The write succeeds
 - The data out on the writing port accurately reflects the data written.
 - The data out on the reading port is invalid.

Conflicts do not cause any physical damage.

Single Port Timing

Figure 33 shows a timing diagram for a single port of a block SelectRAM+ memory. The block SelectRAM+ AC switching characteristics are specified in the data sheet. The block SelectRAM+ memory is initially disabled.



ds022_0343_121399

Figure 33: Timing Diagram for Single Port Block SelectRAM+ Memory

At the first rising edge of the CLK pin, the ADDR, DI, EN, WE, and RST pins are sampled. The EN pin is High and the WE pin is Low indicating a read operation. The DO bus contains the contents of the memory location, 0x00, as indicated by the ADDR bus.

At the second rising edge of the CLK pin, the ADDR, DI, EN, WR, and RST pins are sampled again. The EN and WE pins are High indicating a write operation. The DO bus mirrors the DI bus. The DI bus is written to the memory location 0x0F.

At the third rising edge of the CLK pin, the ADDR, DI, EN, WR, and RST pins are sampled again. The EN pin is High and the WE pin is Low indicating a read operation. The DO

bus contains the contents of the memory location 0x7E as indicated by the ADDR bus.

At the fourth rising edge of the CLK pin, the ADDR, DI, EN, WR, and RST pins are sampled again. The EN pin is Low indicating that the block SelectRAM+ memory is now disabled. The DO bus retains the last value.

Dual Port Timing

Figure 34 shows a timing diagram for a true dual-port read/write block SelectRAM+ memory. The clock on port A has a longer period than the clock on Port B. The timing parameter T_{BCCS} , (clock-to-clock set-up) is shown on this diagram. The parameter, T_{BCCS} is violated once in the dia-

gram. All other timing parameters are identical to the single port version shown in [Figure 33](#).

T_{BCCS} is only of importance when the address of both ports are the same and at least one port is performing a write operation. When the clock-to-clock set-up parameter is violated for a WRITE-WRITE condition, the contents of the memory at that location are invalid. When the clock-to-clock set-up parameter is violated for a WRITE-READ condition, the contents of the memory are correct, but the read port has invalid data.

At the first rising edge of the CLKA, memory location 0x00 is to be written with the value 0xAAAA and is mirrored on the

DOA bus. The last operation of Port B was a read to the same memory location 0x00. The DOB bus of Port B does not change with the new value on Port A, and retains the last read value. A short time later, Port B executes another read to memory location 0x00, and the DOB bus now reflects the new memory value written by Port A.

At the second rising edge of CLKA, memory location 0x7E is written with the value 0x9999 and is mirrored on the DOA bus. Port B then executes a read operation to the same memory location without violating the T_{BCCS} parameter and the DOB reflects the new memory values written by Port A.

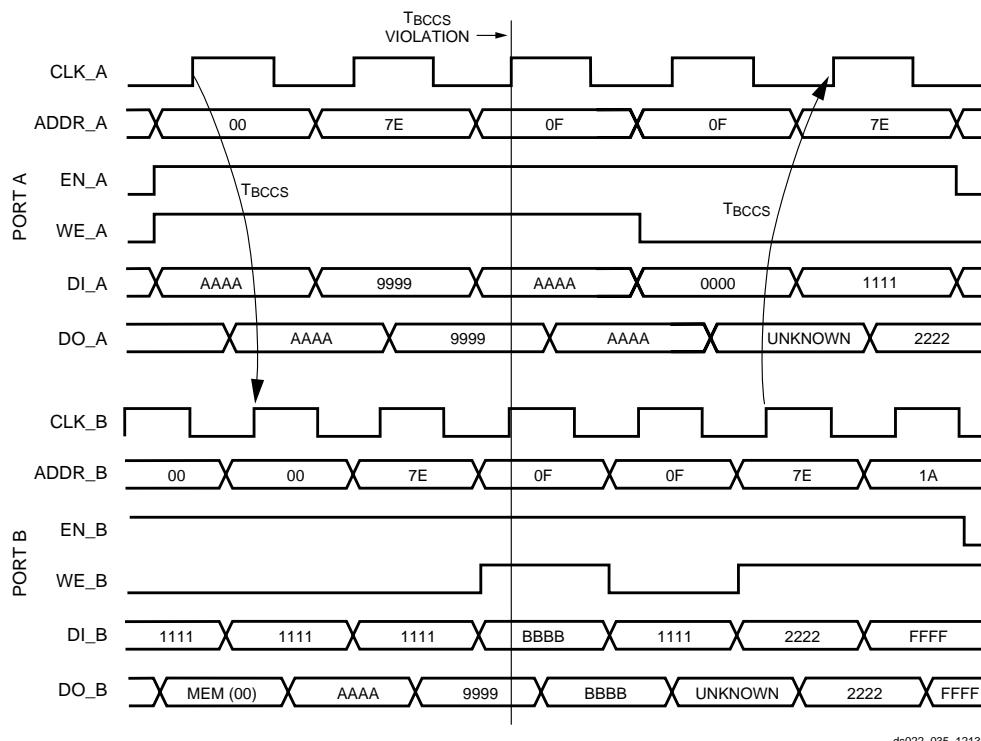


Figure 34: Timing Diagram for a True Dual-port Read/Write Block SelectRAM+ Memory

At the third rising edge of CLKA, the T_{BCCS} parameter is violated with two writes to memory location 0x0F. The DOA and DOB busses reflect the contents of the DIA and DIB busses, but the stored value at 0x0F is invalid.

At the fourth rising edge of CLKA, a read operation is performed at memory location 0x0F and invalid data is present on the DOA bus. Port B also executes a read operation to memory location 0x0F and also reads invalid data.

At the fifth rising edge of CLKA a read operation is performed that does not violate the T_{BCCS} parameter to the previous write of 0x7E by Port B. The DOA bus reflects the recently written value by Port B.

Initialization

The block SelectRAM+ memory can initialize during the device configuration sequence. The 16 initialization proper-

ties of 64 hex values each (a total of 4096 bits) set the initialization of each RAM. These properties appear in [Table 17](#). Any initialization properties not explicitly set configure as zeros. Partial initialization strings pad with zeros. Initialization strings greater than 64 hex values generate an error. The RAMs can be simulated with the initialization values using generics in VHDL simulators and parameters in Verilog simulators.

Initialization in VHDL and Synopsys

The block SelectRAM+ structures can be initialized in VHDL for both simulation and synthesis for inclusion in the EDIF output file. The simulation of the VHDL code uses a generic to pass the initialization. The Synopsys FPGA compiler does not presently support generics. The initialization values instead attach as attributes to the RAM by a built-in Synopsys dc_script. The translate_off statement stops syn-

thesis translation of the generic statements. The VHDL Initialization Example illustrates a code module that employs these techniques.

Table 17: RAM Initialization Properties

Property	Memory Cells
INIT_00	255 to 0
INIT_01	511 to 256
INIT_02	767 to 512
INIT_03	1023 to 768
INIT_04	1279 to 1024
INIT_05	1535 to 1280
INIT_06	1791 to 2047
INIT_07	2047 to 1792
INIT_08	2303 to 2048
INIT_09	2559 to 2304
INIT_0a	2815 to 2560
INIT_0b	3071 to 2816
INIT_0c	3327 to 3072
INIT_0d	3583 to 3328
INIT_0e	3839 to 3584
INIT_0f	4095 to 3840

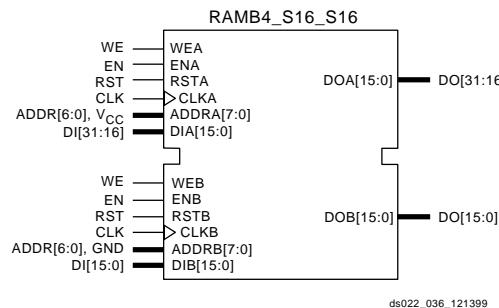
Initialization in Verilog and Synopsys

The block SelectRAM+ structures can be initialized in Verilog for both simulation and synthesis for inclusion in the EDIF output file. The simulation of the Verilog code uses a defparam to pass the initialization. The Synopsys FPGA compiler does not presently support defparam. The initialization values instead attach as attributes to the RAM by a built-in Synopsys dc_script. The translate_off statement stops synthesis translation of the defparam statements. The Verilog Initialization Example illustrates a code module that employs these techniques.

Design Examples

Creating a 32-bit Single-Port RAM

The true dual-read/write port functionality of the block SelectRAM+ memory allows a single port, 128 deep by 32-bit wide RAM to be created using a single block SelectRAM+ cell as shown in [Figure 35](#).

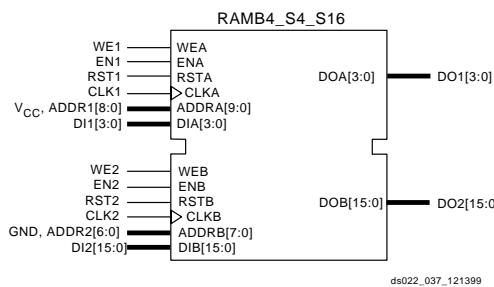


[Figure 35: Single Port 128 x 32 RAM](#)

Interleaving the memory space, setting the LSB of the address bus of Port A to 1 (V_{CC}), and the LSB of the address bus of Port B to 0 (GND), allows a 32-bit wide single port RAM to be created.

Creating Two Single-Port RAMs

The true dual-read/write port functionality of the block SelectRAM+ memory allows a single RAM to be split into two single port memories of 2K bits each as shown in [Figure 36](#).



[Figure 36: 512 x 4 RAM and 128 x 16 RAM](#)

In this example, a 512K x 4 RAM (Port A) and a 128 x 16 RAM (Port B) are created out of a single block SelectRAM+. The address space for the RAM is split by fixing the MSB of Port A to 1 (V_{CC}) for the upper 2K bits and the MSB of Port B to 0 (GND) for the lower 2K bits.

VHDL Initialization Example

Block Memory Generation

The CoreGen program generates memory structures using the block SelectRAM+ features. This program outputs VHDL or Verilog simulation code templates and an EDIF file for inclusion in a design.

```

library IEEE;
use IEEE.std_logic_1164.all;
entity MYMEM is
port (CLK, WE:in std_logic;
ADDR: in std_logic_vector(8 downto 0);
DIN: in std_logic_vector(7 downto 0);
DOUT: out std_logic_vector(7 downto 0));
end MYMEM;
architecture BEHAVE of MYMEM is
signal logic0, logic1: std_logic;
component RAMB4_S8
--synopsys translate_off
generic( INIT_00,INIT_01, INIT_02, INIT_03,
INIT_04, INIT_05, INIT_06, INIT_07,
INIT_08, INIT_09, INIT_0a, INIT_0b, INIT_0c,
INIT_0d, INIT_0e, INIT_0f : BIT_VECTOR(255
downto 0)
:=
X"0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000");
--synopsys translate_on
port (WE, EN, RST, CLK: in STD_LOGIC;
ADDR: in STD_LOGIC_VECTOR(8 downto 0);
DI: in STD_LOGIC_VECTOR(7 downto 0);
DO: out STD_LOGIC_VECTOR(7 downto 0));
end component;

```

```

--synopsys dc_script_begin
--set_attribute ram0 INIT_00
"0123456789ABCDEF0123456789ABCDEF0123456789ABC
DEF0123456789ABCDEF" -type string
--set_attribute ram0 INIT_01
"FEDCBA9876543210FEDCBA9876543210FEDCBA9876543
210FEDCBA9876543210" -type string
--synopsys dc_script_end
begin
logic0 <='0';
logic1 <='1';
ram0: RAMB4_S8
--synopsys translate_off
generic map (
INIT_00 =>
X"0123456789ABCDEF0123456789ABCDEF0123456789AB
CDEF0123456789ABCDEF",
INIT_01 =>
X"FEDCBA9876543210FEDCBA9876543210FEDCBA987654
3210FEDCBA9876543210")
--synopsys translate_on
port map (WE=>WE, EN=>logic1, RST=>logic0,
CLK=>CLK, ADDR=>ADDR, DI=>DIN, DO=>DOUT);
end BEHAVE;

```

Verilog Initialization Example

```

module MYMEM (CLK, WE, ADDR, DIN, DOUT);
input CLK, WE;
input [8:0] ADDR;
input [7:0] DIN;
output [7:0] DOUT;
wire logic0, logic1;
//synopsys dc_script_begin
//set_attribute ram0 INIT_00
"0123456789ABCDEF0123456789ABCDEF0123456789ABC
DEF0123456789ABCDEF" -type string
//set_attribute ram0 INIT_01
"FEDCBA9876543210FEDCBA9876543210FEDCBA9876543
210FEDCBA9876543210" -type string
//synopsys dc_script_end

```

```

assign logic0 = 1'b0;
assign logic1 = 1'b1;
RAMB4_S8 ram0 (.WE(WE), .EN(logic1),
.RST(logic0), .CLK(CLK), .ADDR(ADDR), .DI(DIN),
.DO(DOUT));
//synopsys translate_off
defparam ram0.INIT_00 =
256'h'0123456789ABCDEF0123456789ABCDEF012345678
9ABCDEF0123456789ABCDEF;
defparam ram0.INIT_01 =
256'h'FEDCBA9876543210FEDCBA9876543210FEDCBA987
6543210FEDCBA9876543210;
//synopsys translate_on
endmodule

```

Using SelectI/O

The Virtex-E FPGA series includes a highly configurable, high-performance I/O resource, called SelectI/O™ to provide support for a wide variety of I/O standards. The SelectI/O resource is a robust set of features including programmable control of output drive strength, slew rate, and input delay and hold time. Taking advantage of the flexibility and SelectI/O features and the design considerations described in this document can improve and simplify system level design.

Introduction

As FPGAs continue to grow in size and capacity, the larger and more complex systems designed for them demand an increased variety of I/O standards. Furthermore, as system clock speeds continue to increase, the need for high performance I/O becomes more important.

While chip-to-chip delays have an increasingly substantial impact on overall system speed, the task of achieving the

desired system performance becomes more difficult with the proliferation of low-voltage I/O standards. SelectI/O, the revolutionary input/output resources of Virtex-E devices, resolve this potential problem by providing a highly configurable, high-performance alternative to the I/O resources of more conventional programmable devices. Virtex-E SelectI/O features combine the flexibility and time-to-market advantages of programmable logic with the high performance previously available only with ASICs and custom ICs.

Each SelectI/O block can support up to 20 I/O standards. Supporting such a variety of I/O standards allows the support of a wide variety of applications, from general purpose standard applications to high-speed low-voltage memory busses.

SelectI/O blocks also provide selectable output drive strengths and programmable slew rates for the LVTTL output buffers, as well as an optional, programmable weak pull-up, weak pull-down, or weak “keeper” circuit ideal for use in external bussing applications.

Each Input/Output Block (IOB) includes three registers, one each for the input, output, and 3-state signals within the IOB. These registers are optionally configurable as either a D-type flip-flop or as a level sensitive latch.

The input buffer has an optional delay element used to guarantee a zero hold time requirement for input signals registered within the IOB.

The Virtex-E SelectI/O features also provide dedicated resources for input reference voltage (V_{REF}) and output source voltage (V_{CCO}), along with a convenient banking system that simplifies board design.

By taking advantage of the built-in features and wide variety of I/O standards supported by the SelectI/O features, system-level design and board design can be greatly simplified and improved.

Fundamentals

Modern bus applications, pioneered by the largest and most influential companies in the digital electronics industry, are commonly introduced with a new I/O standard tailored specifically to the needs of that application. The bus I/O standards provide specifications to other vendors who create products designed to interface with these applications. Each standard often has its own specifications for current, voltage, I/O buffering, and termination techniques.

The ability to provide the flexibility and time-to-market advantages of programmable logic is increasingly dependent on the capability of the programmable logic device to support an ever increasing variety of I/O standards.

The SelectI/O resources feature highly configurable input and output buffers which provide support for a wide variety of I/O standards. As shown in **Table 18**, each buffer type can support a variety of voltage requirements.

Table 18: Virtex-E Supported I/O Standards

I/O Standard	Output V_{CCO}	Input V_{CCO}	Input V_{REF}	Board Termination Voltage (V_{TT})
LVTTL	3.3	3.3	N/A	N/A
LVCMOS2	2.5	2.5	N/A	N/A
LVCMOS18	1.8	1.8	N/A	N/A
SSTL3 I & II	3.3	N/A	1.50	1.50
SSTL2 I & II	2.5	N/A	1.25	1.25
GTL	N/A	N/A	0.80	1.20
GTL+	N/A	N/A	1.0	1.50
HSTL I	1.5	N/A	0.75	0.75
HSTL III & IV	1.5	N/A	0.90	1.50
CTT	3.3	N/A	1.50	1.50
AGP-2X	3.3	N/A	1.32	N/A
PCI33_3	3.3	3.3	N/A	N/A
PCI66_3	3.3	3.3	N/A	N/A
BLVDS & LVDS	2.5	N/A	N/A	N/A
LVPECL	3.3	N/A	N/A	N/A

Overview of Supported I/O Standards

This section provides a brief overview of the I/O standards supported by all Virtex-E devices.

While most I/O standards specify a range of allowed voltages, this document records typical voltage values only. Detailed information on each specification can be found on the Electronic Industry Alliance Jedec website at:

<http://www.jedec.org>

LVTTL — Low-Voltage TTL

The Low-Voltage TTL, or LVTTL standard is a general purpose EIA/JESDSA standard for 3.3V applications that uses an LVTTL input buffer and a Push-Pull output buffer. This standard requires a 3.3V output source voltage (V_{CCO}), but does not require the use of a reference voltage (V_{REF}) or a termination voltage (V_{TT}).

LVCMOS2 — Low-Voltage CMOS for 2.5 Volts

The Low-Voltage CMOS for 2.5 Volts or lower, or LVCMOS2 standard is an extension of the LVCMOS standard (JESD 8.-5) used for general purpose 2.5V applications. This standard requires a 2.5V output source voltage (V_{CCO}), but does not require the use of a reference voltage (V_{REF}) or a board termination voltage (V_{TT}).

LVCMOS18 — 1.8 V Low Voltage CMOS

This standard is an extension of the LVCMOS standard. It is used in general purpose 1.8 V applications. The use of a reference voltage (V_{REF}) or a board termination voltage (V_{TT}) is not required.

PCI — Peripheral Component Interface

The Peripheral Component Interface, or PCI standard specifies support for both 33 MHz and 66 MHz PCI bus applications. It uses a LVTTL input buffer and a Push-Pull output buffer. This standard does not require the use of a reference voltage (V_{REF}) or a board termination voltage (V_{TT}), however, it does require a 3.3V output source voltage (V_{CCO}).

GTL — Gunning Transceiver Logic Terminated

The Gunning Transceiver Logic, or GTL standard is a high-speed bus standard (JESD8.3) invented by Xerox. Xilinx has implemented the terminated variation for this standard. This standard requires a differential amplifier input buffer and a Open Drain output buffer.

GTL+ — Gunning Transceiver Logic Plus

The Gunning Transceiver Logic Plus, or GTL+ standard is a high-speed bus standard (JESD8.3) first used by the Pentium Pro processor.

HSTL — High-Speed Transceiver Logic

The High-Speed Transceiver Logic, or HSTL standard is a general purpose high-speed, 1.5V bus standard sponsored by IBM (EIA/JESD 8-6). This standard has four variations or classes. SelectI/O devices support Class I, III, and IV. This standard requires a Differential Amplifier input buffer and a Push-Pull output buffer.

SSTL3 — Stub Series Terminated Logic for 3.3V

The Stub Series Terminated Logic for 3.3V, or SSTL3 standard is a general purpose 3.3V memory bus standard also sponsored by Hitachi and IBM (JESD8-8). This standard has two classes, I and II. SelectI/O devices support both classes for the SSTL3 standard. This standard requires a Differential Amplifier input buffer and an Push-Pull output buffer.

SSTL2 — Stub Series Terminated Logic for 2.5V

The Stub Series Terminated Logic for 2.5V, or SSTL2 standard is a general purpose 2.5V memory bus standard sponsored by Hitachi and IBM (JESD8-9). This standard has two classes, I and II. SelectI/O devices support both classes for the SSTL2 standard. This standard requires a Differential Amplifier input buffer and an Push-Pull output buffer.

CTT — Center Tap Terminated

The Center Tap Terminated, or CTT standard is a 3.3V memory bus standard sponsored by Fujitsu (JESD8-4). This standard requires a Differential Amplifier input buffer and a Push-Pull output buffer.

AGP-2X — Advanced Graphics Port

The Intel AGP standard is a 3.3V Advanced Graphics Port-2X bus standard used with the Pentium II processor for graphics applications. This standard requires a Push-Pull output buffer and a Differential Amplifier input buffer.

LVDS — Low Voltage Differential Signal

LVDS is a differential I/O standard. It requires that one data bit is carried through two signal lines. As with all differential signaling standards, LVDS has an inherent noise immunity over single-ended I/O standards. The voltage swing between two signal lines is approximately 350mV. The use of a reference voltage (V_{REF}) or a board termination voltage (V_{TT}) is not required. LVDS requires the use of two pins per input or output. LVDS requires external resistor termination.

BLVDS — Bus LVDS

This standard allows for bidirectional LVDS communication between two or more devices. The external resistor termination is different than the one for standard LVDS.

LVPECL — Low Voltage Positive Emitter Coupled Logic

LVPECL is another differential I/O standard. It requires two signal lines for transmitting one data bit. This standard specifies two pins per input or output. The voltage swing between these two signal lines is approximately 850 mV. The use of a reference voltage (V_{REF}) or a board termination voltage (V_{TT}) is not required. The LVPECL standard requires external resistor termination.

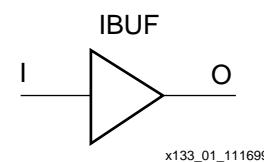
Library Symbols

The Xilinx library includes an extensive list of symbols designed to provide support for the variety of SelectI/O features. Most of these symbols represent variations of the five generic SelectI/O symbols.

- IBUF (input buffer)
- IBUFG (global clock input buffer)
- OBUF (output buffer)
- OBUFT (3-state output buffer)
- IOBUF (input/output buffer)

IBUF

Signals used as inputs to the Virtex-E device must source an input buffer (IBUF) via an external input port. The generic Virtex-E IBUF symbol appears in [Figure 37](#). The extension



[Figure 37: Input Buffer \(IBUF\) Symbols](#)

to the base name defines which I/O standard the IBUF uses. The assumed standard is LVTTL when the generic IBUF has no specified extension.

The following list details the variations of the IBUF symbol:

- IBUF
- IBUF_LVCMOS2
- IBUF_PCI33_3
- IBUF_PCI66_3
- IBUF_GTL
- IBUF_GTLB
- IBUF_HSTL_I
- IBUF_HSTL_III
- IBUF_HSTL_IV
- IBUF_SSTL3_I
- IBUF_SSTL3_II
- IBUF_SSTL2_I
- IBUF_SSTL2_II
- IBUF_CTT
- IBUF_AGP
- IBUF_LVCMOS18
- IBUF_LVDS
- IBUF_LVPECL

When the IBUF symbol supports an I/O standard that requires a V_{REF} , the IBUF automatically configures as a differential amplifier input buffer. The V_{REF} voltage must be supplied on the V_{REF} pins. In the case of LVDS, LVPECL, and BLVDS, V_{REF} is not required.

The voltage reference signal is “banked” within the Virtex-E device on a half-edge basis such that for all packages there are eight independent V_{REF} banks internally. See [Figure 38](#) for a representation of the Virtex-E I/O banks. Within each bank approximately one of every six I/O pins is automatically configured as a V_{REF} input. After placing a differential amplifier input signal within a given V_{REF} bank, the same external source must drive all I/O pins configured as a V_{REF} input.

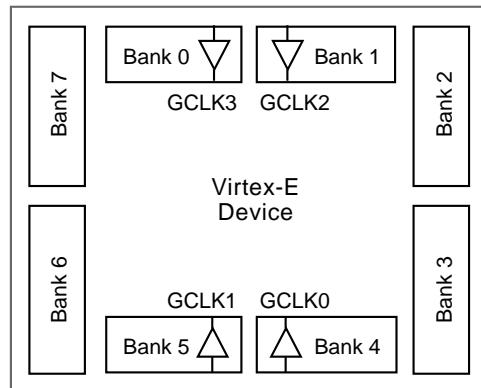
IBUF placement restrictions require that any differential amplifier input signals within a bank be of the same standard. How to specify a specific location for the IBUF via the LOC property is described below. [Table 19](#) summarizes the Virtex-E input standards compatibility requirements.

An optional delay element is associated with each IBUF. When the IBUF drives a flip-flop within the IOB, the delay element by default activates to ensure a zero hold-time requirement. The NODELAY=TRUE property overrides this default.

When the IBUF does not drive a flip-flop within the IOB, the delay element de-activates by default to provide higher performance. To delay the input signal, activate the delay element with the DELAY=TRUE property.

Table 19: Xilinx Input Standards Compatibility Requirements

Rule 1	Standards with the same input V_{CCO} , output V_{CCO} , and V_{REF} can be placed within the same bank.
--------	--



ds022_42_012100

Figure 38: Virtex-E I/O Banks

IBUFG

Signals used as high fanout clock inputs to the Virtex-E device should drive a global clock input buffer (IBUFG) via an external input port in order to take advantage of one of the four dedicated global clock distribution networks. The output of the IBUFG symbol can drive only a CLKDLL, CLKDLLHF, or BUFG symbol. The generic Virtex-E IBUFG symbol appears in [Figure 39](#).

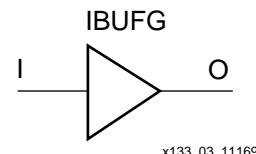


Figure 39: Virtex-E Global Clock Input Buffer (IBUFG) Symbol

The extension to the base name determines which I/O standard is used by the IBUFG. With no extension specified for the generic IBUFG symbol, the assumed standard is LVTTL.

The following list details variations of the IBUFG symbol.

- IBUFG
- IBUFG_LVCMOS2
- IBUFG_PCI33_3
- IBUFG_PCI66_3
- IBUFG_GTL
- IBUFG_GTLP
- IBUFG_HSTL_I
- IBUFG_HSTL_III
- IBUFG_HSTL_IV
- IBUFG_SSTL3_I
- IBUFG_SSTL3_II
- IBUFG_SSTL2_I
- IBUFG_SSTL2_II
- IBUFG_CTT
- IBUFG_AGPs
- IBUFG_LVCMOS18
- IBUFG_LVDS
- IBUFG_LVPECL

When the IBUFG symbol supports an I/O standard that requires a differential amplifier input, the IBUFG automatically configures as a differential amplifier input buffer. The low-voltage I/O standards with a differential amplifier input require an external reference voltage input V_{REF} .

The voltage reference signal is “banked” within the Virtex-E device on a half-edge basis such that for all packages there are eight independent V_{REF} banks internally. See [Figure 38](#) for a representation of the Virtex-E I/O banks. Within each bank approximately one of every six I/O pins is automatically configured as a V_{REF} input. After placing a differential amplifier input signal within a given V_{REF} bank, the same external source must drive all I/O pins configured as a V_{REF} input.

IBUFG placement restrictions require any differential amplifier input signals within a bank be of the same standard. The LOC property can specify a location for the IBUFG.

As an added convenience, the BUFGP can be used to instantiate a high fanout clock input. The BUFGP symbol represents a combination of the LVTTL IBUFG and BUFG symbols, such that the output of the BUFGP can connect directly to the clock pins throughout the design.

Unlike previous architectures, the Virtex-E BUFGP symbol can only be placed in a global clock pad location. The LOC property can specify a location for the BUFGP.

OBUF

An OBUF must drive outputs through an external output port. The generic output buffer (OBUF) symbol appears in [Figure 40](#).

The extension to the base name defines which I/O standard the OBUF uses. With no extension specified for the generic OBUF symbol, the assumed standard is slew rate limited LVTTL with 12 mA drive strength.

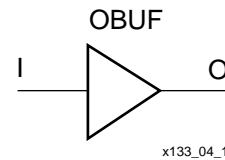


Figure 40: Virtex-E Output Buffer (OBUF) Symbol

The LVTTL OBUF additionally can support one of two slew rate modes to minimize bus transients. By default, the slew rate for each output buffer is reduced to minimize power bus transients when switching non-critical signals.

LVTTL output buffers have selectable drive strengths.

The format for LVTTL OBUF symbol names is as follows:

OBUF_<slew_rate>_<drive_strength>

where <slew_rate> is either F (Fast) or S (Slow), and <drive_strength> is specified in millamps (2, 4, 6, 8, 12, 16, or 24).

The following list details variations of the OBUF symbol.

- OBUF
- OBUF_S_2
- OBUF_S_4
- OBUF_S_6
- OBUF_S_8
- OBUF_S_12
- OBUF_S_16
- OBUF_S_24
- OBUF_F_2
- OBUF_F_4
- OBUF_F_6
- OBUF_F_8
- OBUF_F_12
- OBUF_F_16
- OBUF_F_24
- OBUF_LVCMOS2
- OBUF_PCI33_3
- OBUF_PCI66_3
- OBUF_GTL
- OBUF_GTLP
- OBUF_HSTL_I
- OBUF_HSTL_III
- OBUF_HSTL_IV
- OBUF_SSTL3_I
- OBUF_SSTL3_II
- OBUF_SSTL2_I
- OBUF_SSTL2_II
- OBUF_CTT
- OBUF_AGPs
- OBUF_LVCMOS18
- OBUF_LVDS
- OBUF_LVPECL

The Virtex-E series supports eight banks for the HQ and PQ packages. The CS packages support four V_{CCO} banks.

OBUF placement restrictions require that within a given V_{CCO} bank each OBUF share the same output source drive voltage. Input buffers of any type and output buffers that do not require V_{CCO} can be placed within any V_{CCO} bank. **Table 20** summarizes the Virtex-E output compatibility requirements. The LOC property can specify a location for the OBUF.

Table 20: Output Standards Compatibility Requirements

Rule 1	Only outputs with standards that share compatible V_{CCO} can be used within the same bank.
Rule 2	There are no placement restrictions for outputs with standards that do not require a V_{CCO} .
V_{CCO}	Compatible Standards
3.3	LVTTL, SSTL3_I, SSTL3_II, CTT, AGP, GTL, GTL+, PCI33_3, PCI66_3
2.5	SSTL2_I, SSTL2_II, LVCMOS2, GTL, GTL+
1.5	HSTL_I, HSTL_III, HSTL_IV, GTL, GTL+

OBUFT

The generic 3-state output buffer OBUFT (see [Figure 41](#)) typically implements 3-state outputs or bidirectional I/O.

The extension to the base name defines which I/O standard OBUFT uses. With no extension specified for the generic OBUFT symbol, the assumed standard is slew rate limited LVTTL with 12 mA drive strength.

The LVTTL OBUFT additionally can support one of two slew rate modes to minimize bus transients. By default, the slew rate for each output buffer is reduced to minimize power bus transients when switching non-critical signals.

LVTTL 3-state output buffers have selectable drive strengths.

The format for LVTTL OBUFT symbol names is as follows:

OBUFT_<slew_rate>_<drive_strength>

where <slew_rate> is either F (Fast) or S (Slow), and <drive_strength> is specified in millamps (2, 4, 6, 8, 12, 16, or 24).

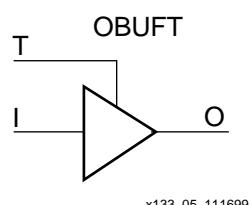


Figure 41: 3-State Output Buffer Symbol (OBUFT)

The following list details variations of the OBUFT symbol.

- OBUFT
- OBUFT_S_2
- OBUFT_S_4
- OBUFT_S_6
- OBUFT_S_8
- OBUFT_S_12
- OBUFT_S_16
- OBUFT_S_24
- OBUFT_F_2
- OBUFT_F_4
- OBUFT_F_6
- OBUFT_F_8
- OBUFT_F_12
- OBUFT_F_16
- OBUFT_F_24
- OBUFT_LVCMOS2
- OBUFT_PCI33_3
- OBUFT_PCI66_3
- OBUFT_GTL
- OBUFT_GTL_P
- OBUFT_HSTL_I
- OBUFT_HSTL_III
- OBUFT_HSTL_IV
- OBUFT_SSTL3_I
- OBUFT_SSTL3_II
- OBUFT_SSTL2_I
- OBUFT_SSTL2_II
- OBUFT_CTT
- OBUFT_AG
- OBUFT_LVCMOS18
- OBUFT_LVDS
- OBUFT_LVPECL

The Virtex-E series supports eight banks for the HQ and PQ packages. The CS package supports four V_{CCO} banks.

The SelectIO OBUFT placement restrictions require that within a given V_{CCO} bank each OBUFT share the same output source drive voltage. Input buffers of any type and output buffers that do not require V_{CCO} can be placed within the same V_{CCO} bank.

The LOC property can specify a location for the OBUFT.

3-state output buffers and bidirectional buffers can have either a weak pull-up resistor, a weak pull-down resistor, or a weak “keeper” circuit. Control this feature by adding the appropriate symbol to the output net of the OBUFT (PULLUP, PULLDOWN, or KEEPER).

The weak “keeper” circuit requires the input buffer within the IOB to sample the I/O signal. So, OBUFTs programmed for an I/O standard that requires a V_{REF} have automatic placement of a V_{REF} in the bank with an OBUFT configured with

a weak “keeper” circuit. This restriction does not affect most circuit design as applications using an OBUFT configured with a weak “keeper” typically implement a bidirectional I/O. In this case the IBUF (and the corresponding V_{REF}) are explicitly placed.

The LOC property can specify a location for the OBUFT.

IOBUF

Use the IOBUF symbol for bidirectional signals that require both an input buffer and a 3-state output buffer with an active high 3-state pin. The generic input/output buffer IOBUF appears in [Figure 42](#).

The extension to the base name defines which I/O standard the IOBUF uses. With no extension specified for the generic IOBUF symbol, the assumed standard is LVTTL input buffer and slew rate limited LVTTL with 12 mA drive strength for the output buffer.

The LVTTL IOBUF additionally can support one of two slew rate modes to minimize bus transients. By default, the slew rate for each output buffer is reduced to minimize power bus transients when switching non-critical signals.

LVTTL bidirectional buffers have selectable output drive strengths.

The format for LVTTL IOBUF symbol names is as follows:

IOBUF_<slew_rate>_<drive_strength>

where <slew_rate> is either F (Fast) or S (Slow), and <drive_strength> is specified in millamps (2, 4, 6, 8, 12, 16, or 24).

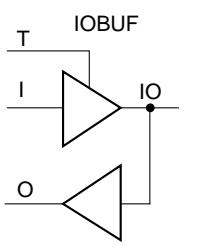


Figure 42: Input/Output Buffer Symbol (IOBUF)

The following list details variations of the IOBUF symbol.

- IOBUF
- IOBUF_S_2
- IOBUF_S_4
- IOBUF_S_6
- IOBUF_S_8
- IOBUF_S_12
- IOBUF_S_16
- IOBUF_S_24
- IOBUF_F_2
- IOBUF_F_4
- IOBUF_F_6

- IOBUF_F_8
- IOBUF_F_12
- IOBUF_F_16
- IOBUF_F_24
- IOBUF_LVCMS2
- IOBUF_PCI33_3
- IOBUF_PCI66_3
- IOBUF_GTL
- IOBUF_GTLP
- IOBUF_HSTL_I
- IOBUF_HSTL_III
- IOBUF_HSTL_IV
- IOBUF_SSTL3_I
- IOBUF_SSTL3_II
- IOBUF_SSTL2_I
- IOBUF_SSTL2_II
- IOBUF_CTT
- IOBUF_AGP
- IOBUF_LVCMS18
- IOBUF_LVDS
- IOBUF_LVPECL

When the IOBUF symbol used supports an I/O standard that requires a differential amplifier input, the IOBUF automatically configures with a differential amplifier input buffer. The low-voltage I/O standards with a differential amplifier input require an external reference voltage input V_{REF} .

The voltage reference signal is “banked” within the Virtex-E device on a half-edge basis such that for all packages there are eight independent V_{REF} banks internally. See [Figure 38, page 31](#) for a representation of the Virtex-E I/O banks. Within each bank approximately one of every six I/O pins is automatically configured as a V_{REF} input. After placing a differential amplifier input signal within a given V_{REF} bank, the same external source must drive all I/O pins configured as a V_{REF} input.

IOBUF placement restrictions require any differential amplifier input signals within a bank be of the same standard.

The Virtex-E series supports eight banks for the HQ and PQ packages. The CS package supports four V_{CCO} banks.

Additional restrictions on the Virtex-E SelectIO IOBUF placement require that within a given V_{CCO} bank each IOBUF must share the same output source drive voltage. Input buffers of any type and output buffers that do not require V_{CCO} can be placed within the same V_{CCO} bank. The LOC property can specify a location for the IOBUF.

An optional delay element is associated with the input path in each IOBUF. When the IOBUF drives an input flip-flop within the IOB, the delay element activates by default to ensure a zero hold-time requirement. Override this default with the NODELAY=TRUE property.

In the case when the IOBUF does not drive an input flip-flop within the IOB, the delay element de-activates by default to provide higher performance. To delay the input signal, activate the delay element with the **DELAY=TRUE** property.

3-state output buffers and bidirectional buffers can have either a weak pull-up resistor, a weak pull-down resistor, or a weak “keeper” circuit. Control this feature by adding the appropriate symbol to the output net of the IOBUF (PULLUP, PULLDOWN, or KEEPER).

SelectI/O Properties

Access to some of the SelectI/O features (for example, location constraints, input delay, output drive strength, and slew rate) is available through properties associated with these features.

Input Delay Properties

An optional delay element is associated with each IBUF. When the IBUF drives a flip-flop within the IOB, the delay element activates by default to ensure a zero hold-time requirement. Use the **NODELAY=TRUE** property to override this default.

In the case when the IBUF does not drive a flip-flop within the IOB, the delay element by default de-activates to provide higher performance. To delay the input signal, activate the delay element with the **DELAY=TRUE** property.

IOB Flip-Flop/Latch Property

The Virtex-E series I/O Block (IOB) includes an optional register on the input path, an optional register on the output path, and an optional register on the 3-state control pin. The design implementation software automatically takes advantage of these registers when the following option for the Map program is specified.

```
map -pr b <filename>
```

Alternatively, the **IOB = TRUE** property can be placed on a register to force the mapper to place the register in an IOB.

Location Constraints

Specify the location of each SelectI/O symbol with the location constraint LOC attached to the SelectI/O symbol. The external port identifier indicates the value of the location constrain. The format of the port identifier depends on the package chosen for the specific design.

The LOC properties use the following form:

LOC=A42

LOC=P37

Output Slew Rate Property

As mentioned above, a variety of symbol names provide the option of choosing the desired slew rate for the output buffers. In the case of the LVTTL output buffers (OBUF, OBUFT, and IOBUF), slew rate control can be alternatively programmed with the **SLEW=** property. By default, the slew rate

for each output buffer is reduced to minimize power bus transients when switching non-critical signals. The **SLEW=** property has one of the two following values.

SLEW=SLOW

SLEW=FAST

Output Drive Strength Property

The desired output drive strength can be additionally specified by choosing the appropriate library symbol. The Xilinx library also provides an alternative method for specifying this feature. For the LVTTL output buffers (OBUF, OBUFT, and IOBUF), the desired drive strength can be specified with the **DRIVE=** property. This property could have one of the following seven values.

DRIVE=2

DRIVE=4

DRIVE=6

DRIVE=8

DRIVE=12 (Default)

DRIVE=16

DRIVE=24

Design Considerations

Reference Voltage (V_{REF}) Pins

Low-voltage I/O standards with a differential amplifier input buffer require an input reference voltage (V_{REF}). Provide the V_{REF} as an external signal to the device.

The voltage reference signal is “banked” within the device on a half-edge basis such that for all packages there are eight independent V_{REF} banks internally. See [Figure 38, page 31](#) for a representation of the Virtex-E I/O banks. Within each bank approximately one of every six I/O pins is automatically configured as a V_{REF} input. After placing a differential amplifier input signal within a given V_{REF} bank, the same external source must drive all I/O pins configured as a V_{REF} input.

Within each V_{REF} bank, any input buffers that require a V_{REF} signal must be of the same type. Output buffers of any type and input buffers can be placed without requiring a reference voltage within the same V_{REF} bank.

Output Drive Source Voltage (V_{CCO}) Pins

Many of the low voltage I/O standards supported by SelectI/O devices require a different output drive source voltage (V_{CCO}). As a result each device can often have to support multiple output drive source voltages.

The Virtex-E series supports eight banks for the HQ and PQ packages. The CS package supports four V_{CCO} banks.

Output buffers within a given V_{CCO} bank must share the same output drive source voltage. Input buffers for LVTTL, LVCMS2, LVCMS18, PCI33_3, and PCI 66_3 use the V_{CCO} voltage for Input V_{CCO} voltage.

Transmission Line Effects

The delay of an electrical signal along a wire is dominated by the rise and fall times when the signal travels a short distance. Transmission line delays vary with inductance and capacitance, but a well-designed board can experience delays of approximately 180 ps per inch.

Transmission line effects, or reflections, typically start at 1.5" for fast (1.5 ns) rise and fall times. Poor (or non-existent) termination or changes in the transmission line impedance cause these reflections and can cause additional delay in longer traces. As system speeds continue to increase, the effect of I/O delays can become a limiting factor and therefore transmission line termination becomes increasingly more important.

Termination Techniques

A variety of termination techniques reduce the impact of transmission line effects.

The following lists output termination techniques.

- None
- Series
- Parallel (Shunt)
- Series and Parallel (Series-Shunt)

Input termination techniques include the following.

- None
- Parallel (Shunt)

These termination techniques can be applied in any combination. A generic example of each combination of termination methods appears in [Figure 43](#).

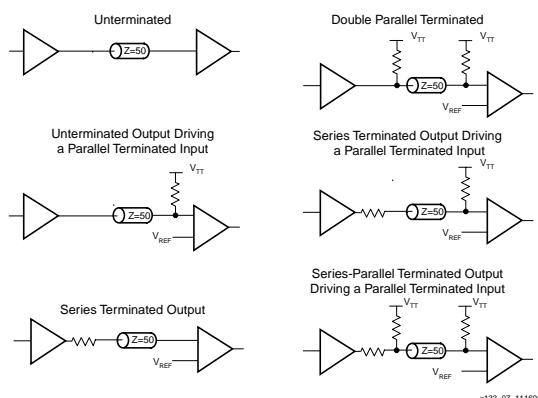


Figure 43: Overview of Standard Input and Output Termination Methods

Simultaneous Switching Guidelines

Ground bounce can occur with high-speed digital ICs when multiple outputs change states simultaneously, causing undesired transient behavior on an output, or in the internal logic. This problem is also referred to as the Simultaneous Switching Output (SSO) problem.

Ground bounce is primarily due to current changes in the combined inductance of ground pins, bond wires, and ground metallization. The IC internal ground level deviates from the external system ground level for a short duration (a few nanoseconds) after multiple outputs change state simultaneously.

Ground bounce affects stable Low outputs and all inputs because they interpret the incoming signal by comparing it to the internal ground. If the ground bounce amplitude exceeds the actual instantaneous noise margin, then a non-changing input can be interpreted as a short pulse with a polarity opposite to the ground bounce.

[Table 21](#) provides guidelines for the maximum number of simultaneously switching outputs allowed per output power/ground pair to avoid the effects of ground bounce. See [Table 22](#) for the number of effective output power/ground pairs for each Virtex-E device and package combination.

Table 21: Guidelines for Max Number of Simultaneously Switching Outputs per Power/Ground Pair

Standard	Package		
	BGA, CS, FGA	HQ	PQ, TQ
LVTTL Slow Slew Rate, 2 mA drive	68	49	36
LVTTL Slow Slew Rate, 4 mA drive	41	31	20
LVTTL Slow Slew Rate, 6 mA drive	29	22	15
LVTTL Slow Slew Rate, 8 mA drive	22	17	12
LVTTL Slow Slew Rate, 12 mA drive	17	12	9
LVTTL Slow Slew Rate, 16 mA drive	14	10	7
LVTTL Slow Slew Rate, 24 mA drive	9	7	5
LVTTL Fast Slew Rate, 2 mA drive	40	29	21
LVTTL Fast Slew Rate, 4 mA drive	24	18	12
LVTTL Fast Slew Rate, 6 mA drive	17	13	9
LVTTL Fast Slew Rate, 8 mA drive	13	10	7
LVTTL Fast Slew Rate, 12 mA drive	10	7	5
LVTTL Fast Slew Rate, 16 mA drive	8	6	4
LVTTL Fast Slew Rate, 24 mA drive	5	4	3
LVCMOS2	10	7	5
PCI	8	6	4
GTL	4	4	4
GTL+	4	4	4
HSTL Class I	18	13	9
HSTL Class III	9	7	5
HSTL Class IV	5	4	3
SSTL2 Class I	15	11	8
SSTL2 Class II	10	7	5
SSTL3 Class I	11	8	6
SSTL3 Class II	7	5	4
CTT	14	10	7
AGP	9	7	5

Note: This analysis assumes a 35 pF load for each output.

Table 22: Virtex-E Equivalent Power/Ground Pairs

Pkg/Part	XCV100E	XCV200E	XCV300E	XCV400E	XCV600E	XCV1000E	XCV1600E	XCV2000E
CS144	12	12						
PQ240	20	20	20	20				
HQ240					20	20		
BG352	20	32	32					
BG432			32	40	40			
BG560				40	40	56	58	60
FG256 ⁽¹⁾	20	24	24					
FG456		40	40					
FG676				54	56			
FG680 ⁽²⁾					46	56	56	56
FG860						58	60	64
FG900					56	58		60
FG1156						96	104	120

Notes:

1. Virtex-E devices in FG256 packages have more V_{CCO} than Virtex series devices.
2. FG680 numbers are preliminary.

Application Examples

Creating a design with the SelectI/O features requires the instantiation of the desired library symbol within the design code. At the board level, designers need to know the termination techniques required for each I/O standard.

This section describes some common application examples illustrating the termination techniques recommended by each of the standards supported by the SelectI/O features.

Termination Examples

Circuit examples involving typical termination techniques for each of the SelectI/O standards follow. For a full range of accepted values for the DC voltage specifications for each standard, refer to the table associated with each figure.

The resistors used in each termination technique example and the transmission lines depicted represent board level components and are not meant to represent components on the device.

GTL

A sample circuit illustrating a valid termination technique for GTL is shown in Figure 44. Table 23 lists DC voltage specifications.

Table 23: GTL Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	-	N/A	-
$V_{REF} = N \times V_{TT}^1$	0.74	0.8	0.86
V_{TT}	1.14	1.2	1.26
$V_{IH} = V_{REF} + 0.05$	0.79	0.85	-
$V_{IL} = V_{REF} - 0.05$	-	0.75	0.81
V_{OH}	-	-	-
V_{OL}	-	0.2	0.4
I_{OH} at V_{OH} (mA)	-	-	-
I_{OL} at V_{OL} (mA) at 0.4V	32	-	-
I_{OL} at V_{OL} (mA) at 0.2V	-	-	40

Notes:

1. N must be greater than or equal to 0.653 and less than or equal to 0.68.

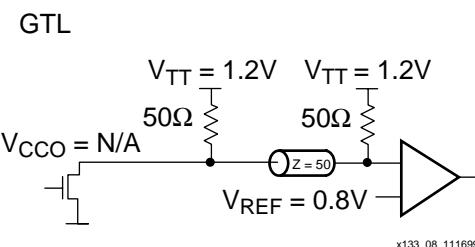


Figure 44: Terminated GTL

GTL+

A sample circuit illustrating a valid termination technique for GTL+ appears in Figure 45. DC voltage specifications appear in Table 24.

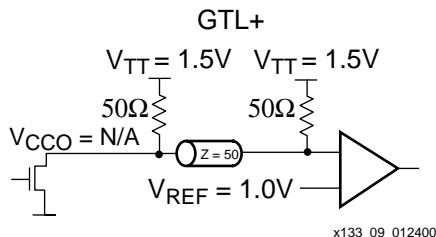


Figure 45: Terminated GTL+

Table 24: GTL+ Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	-	-	-
$V_{REF} = N \times V_{TT}^1$	0.88	1.0	1.12
V_{TT}	1.35	1.5	1.65
$V_{IH} = V_{REF} + 0.1$	0.98	1.1	-
$V_{IL} = V_{REF} - 0.1$	-	0.9	1.02
V_{OH}	-	-	-
V_{OL}	0.3	0.45	0.6
I_{OH} at V_{OH} (mA)	-	-	-
I_{OL} at V_{OL} at 0.6V (mA)	36	-	-
I_{OL} at V_{OL} at 0.3V (mA)	-	-	48

Notes:

1. N must be greater than or equal to 0.653 and less than or equal to 0.68.

HSTL

A sample circuit illustrating a valid termination technique for HSTL_I appears in Figure 46. A sample circuit illustrating a valid termination technique for HSTL_III appears in Figure 47.

Table 25: HSTL Class I Voltage Specification

Parameter	Min	Typ	Max
V_{CCO}	1.40	1.50	1.60
V_{REF}	0.68	0.75	0.90
V_{TT}	-	$V_{CCO} \times 0.5$	-
V_{IH}	$V_{REF} + 0.1$	-	-
V_{IL}	-	-	$V_{REF} - 0.1$
V_{OH}	$V_{CCO} - 0.4$	-	-
V_{OL}			0.4
I_{OH} at V_{OH} (mA)	-8	-	-
I_{OL} at V_{OL} (mA)	8	-	-

HSTL Class I

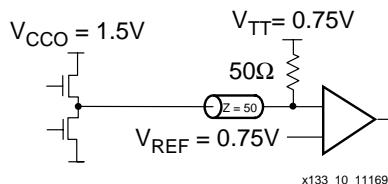


Figure 46: Terminated HSTL Class I

Table 26: HSTL Class III Voltage Specification

Parameter	Min	Typ	Max
V_{CCO}	1.40	1.50	1.60
$V_{REF}^{(1)}$	-	0.90	-
V_{TT}	-	V_{CCO}	-
V_{IH}	$V_{REF} + 0.1$	-	-
V_{IL}	-	-	$V_{REF} - 0.1$
V_{OH}	$V_{CCO} - 0.4$	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	-8	-	-
I_{OL} at V_{OL} (mA)	24	-	-

Note: Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user."

HSTL Class III

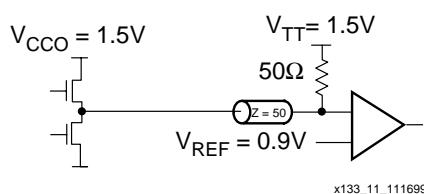


Figure 47: Terminated HSTL Class III

A sample circuit illustrating a valid termination technique for HSTL_III appears in [Figure 49](#). DC voltage specifications appear in [Table 28](#).

Table 27: HSTL Class IV Voltage Specification

Parameter	Min	Typ	Max
V_{CCO}	1.40	1.50	1.60
V_{REF}	-	0.90	-
V_{TT}	-	V_{CCO}	-
V_{IH}	$V_{REF} + 0.1$	-	-
V_{IL}	-	-	$V_{REF} - 0.1$
V_{OH}	$V_{CCO} - 0.4$	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	-8	-	-
I_{OL} at V_{OL} (mA)	48	-	-

Note: Per EIA/JESD8-6, "The value of V_{REF} is to be selected by the user to provide optimum noise margin in the use conditions specified by the user.

HSTL Class IV

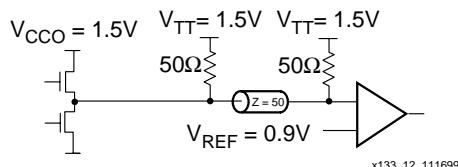


Figure 48: Terminated HSTL Class IV

SSTL3_I

A sample circuit illustrating a valid termination technique for SSTL3_I appears in [Figure 49](#). DC voltage specifications appear in [Table 28](#).

Table 28: SSTL3_I Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.6
$V_{REF} = 0.45 \times V_{CCO}$	1.3	1.5	1.7
$V_{TT} = V_{REF}$	1.3	1.5	1.7
$V_{IH} = V_{REF} + 0.2$	1.5	1.7	3.9 ⁽¹⁾
$V_{IL} = V_{REF} - 0.2$	-0.3 ⁽²⁾	1.3	1.5
$V_{OH} = V_{REF} + 0.6$	1.9	-	-
$V_{OL} = V_{REF} - 0.6$	-	-	1.1
I_{OH} at V_{OH} (mA)	-8	-	-
I_{OL} at V_{OL} (mA)	8	-	-

Notes:

1. V_{IH} maximum is $V_{CCO} + 0.3$
2. V_{IL} minimum does not conform to the formula

SSTL3 Class I

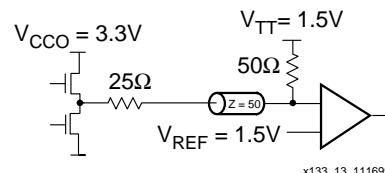


Figure 49: Terminated SSTL3 Class I

SSTL3_II

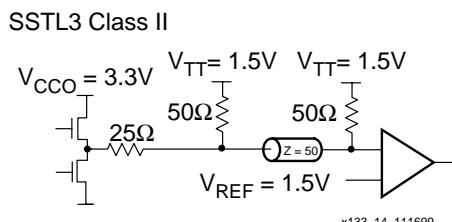
A sample circuit illustrating a valid termination technique for SSTL3_II appears in [Figure 50](#). DC voltage specifications appear in [Table 29](#).

Table 29: SSTL3_II Voltage Specifications

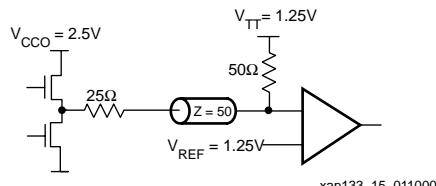
Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.6
$V_{REF} = 0.45 \times V_{CCO}$	1.3	1.5	1.7
$V_{TT} = V_{REF}$	1.3	1.5	1.7
$V_{IH} = V_{REF} + 0.2$	1.5	1.7	3.9 ⁽¹⁾
$V_{IL} = V_{REF} - 0.2$	-0.3 ⁽²⁾	1.3	1.5
$V_{OH} = V_{REF} + 0.8$	2.1	-	-
$V_{OL} = V_{REF} - 0.8$	-	-	0.9
I_{OH} at V_{OH} (mA)	-16	-	-
I_{OL} at V_{OL} (mA)	16	-	-

Notes:

1. V_{IH} maximum is $V_{CCO} + 0.3$
2. V_{IL} minimum does not conform to the formula

**Figure 50: Terminated SSTL3 Class II****SSTL2_I**

A sample circuit illustrating a valid termination technique for SSTL2_I appears in [Figure 51](#). DC voltage specifications appear in [Table 30](#).

SSTL2 Class I**Figure 51: Terminated SSTL2 Class I****Table 30: SSTL2_I Voltage Specifications**

Parameter	Min	Typ	Max
V_{CCO}	2.3	2.5	2.7
$V_{REF} = 0.5 \times V_{CCO}$	1.15	1.25	1.35
$V_{TT} = V_{REF} + N^{(1)}$	1.11	1.25	1.39
$V_{IH} = V_{REF} + 0.18$	1.33	1.43	3.0 ⁽²⁾
$V_{IL} = V_{REF} - 0.18$	-0.3 ⁽³⁾	1.07	1.17
$V_{OH} = V_{REF} + 0.61$	1.76	-	-
$V_{OL} = V_{REF} - 0.61$	-	-	0.74
I_{OH} at V_{OH} (mA)	-7.6	-	-
I_{OL} at V_{OL} (mA)	7.6	-	-

Notes:

1. N must be greater than or equal to -0.04 and less than or equal to 0.04.
2. V_{IH} maximum is $V_{CCO} + 0.3$.
3. V_{IL} minimum does not conform to the formula.

SSTL2_II

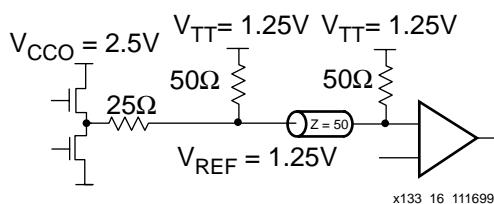
A sample circuit illustrating a valid termination technique for SSTL2_II appears in [Figure 52](#). DC voltage specifications appear in [Table 31](#).

Table 31: SSTL2_II Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	2.3	2.5	2.7
$V_{REF} = 0.5 \times V_{CCO}$	1.15	1.25	1.35
$V_{TT} = V_{REF} + N^{(1)}$	1.11	1.25	1.39
$V_{IH} = V_{REF} + 0.18$	1.33	1.43	3.0 ⁽²⁾
$V_{IL} = V_{REF} - 0.18$	-0.3 ⁽³⁾	1.07	1.17
$V_{OH} = V_{REF} + 0.8$	1.95	-	-
$V_{OL} = V_{REF} - 0.8$	-	-	0.55
I_{OH} at V_{OH} (mA)	-15.2	-	-
I_{OL} at V_{OL} (mA)	15.2	-	-

Notes:

1. N must be greater than or equal to -0.04 and less than or equal to 0.04.
2. V_{IH} maximum is $V_{CCO} + 0.3$.
3. V_{IL} minimum does not conform to the formula.

SSTL2 Class II**Figure 52: Terminated SSTL2 Class II**

CTT

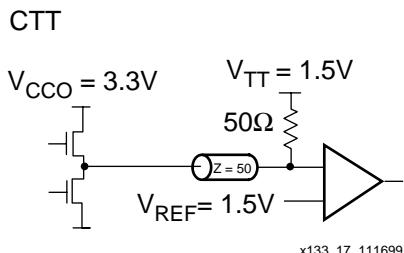
A sample circuit illustrating a valid termination technique for CTT appear in [Figure 53](#). DC voltage specifications appear in [Table 32](#).

Table 32: CTT Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	2.05 ⁽¹⁾	3.3	3.6
V_{REF}	1.35	1.5	1.65
V_{TT}	1.35	1.5	1.65
$V_{IH} = V_{REF} + 0.2$	1.55	1.7	-
$V_{IL} = V_{REF} - 0.2$	-	1.3	1.45
$V_{OH} = V_{REF} + 0.4$	1.75	1.9	-
$V_{OL} = V_{REF} - 0.4$	-	1.1	1.25
I_{OH} at V_{OH} (mA)	-8	-	-
I_{OL} at V_{OL} (mA)	8	-	-

Notes:

- Timing delays are calculated based on V_{CCO} min of 3.0V.

**Figure 53: Terminated CTT**

PCI33_3 & PCI66_3

PCI33_3 or PCI66_3 require no termination. DC voltage specifications appear in [Table 33](#).

Table 33: PCI33_3 and PCI66_3 Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.6
V_{REF}	-	-	-
V_{TT}	-	-	-
$V_{IH} = 0.5 \times V_{CCO}$	1.5	1.65	$V_{CCO} + 0.5$
$V_{IL} = 0.3 \times V_{CCO}$	-0.5	0.99	1.08
$V_{OH} = 0.9 \times V_{CCO}$	2.7	-	-
$V_{OL} = 0.1 \times V_{CCO}$	-	-	0.36

Table 33: PCI33_3 and PCI66_3 Voltage Specifications

Parameter	Min	Typ	Max
I_{OH} at V_{OH} (mA)	Note 1	-	-
I_{OL} at V_{OL} (mA)	Note 1	-	-

Notes:

- Tested according to the relevant specification.

LVTTL

LVTTL requires no termination. DC voltage specifications appears in [Table 34](#).

Table 34: LVTTL Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.6
V_{REF}	-	-	-
V_{TT}	-	-	-
V_{IH}	2.0	-	3.6
V_{IL}	-0.5	-	0.8
V_{OH}	2.4	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	-24	-	-
I_{OL} at V_{OL} (mA)	24	-	-

Notes:

- Note: V_{OL} and V_{OH} for lower drive currents sample tested.

LVCMOS2

LVCMOS2 requires no termination. DC voltage specifications appear in [Table 35](#).

Table 35: LVCMOS2 Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	2.3	2.5	2.7
V_{REF}	-	-	-
V_{TT}	-	-	-
V_{IH}	1.7	-	3.6
V_{IL}	-0.5	-	0.7
V_{OH}	1.9	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	-12	-	-
I_{OL} at V_{OL} (mA)	12	-	-

LVC MOS18

LVC MOS18 does not require termination. **Table 36** lists DC voltage specifications.

Table 36: LVC MOS18 Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	1.70	1.80	1.90
V_{REF}	-	-	-
V_{TT}	-	-	-
V_{IH}	$0.7 \times V_{CCO}$	-	1.95
V_{IL}	-0.5	-	$0.2 \times V_{CCO}$
V_{OH}	$V_{CCO} - 0.4$	-	-
V_{OL}	-	-	0.4
I_{OH} at V_{OH} (mA)	-8	-	-
I_{OL} at V_{OL} (mA)	8	-	-

AGP-2X

The specification for the AGP-2X standard does not document a recommended termination technique. DC voltage specifications appear in **Table 37**.

Table 37: AGP-2X Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.6
$V_{REF} = N \times V_{CCO}^{(1)}$	1.17	1.32	1.48
V_{TT}	-	-	-
$V_{IH} = V_{REF} + 0.2$	1.37	1.52	-
$V_{IL} = V_{REF} - 0.2$	-	1.12	1.28
$V_{OH} = 0.9 \times V_{CCO}$	2.7	3.0	-
$V_{OL} = 0.1 \times V_{CCO}$	-	0.33	0.36
I_{OH} at V_{OH} (mA)	Note 2	-	-
I_{OL} at V_{OL} (mA)	Note 2	-	-

Notes:

1. N must be greater than or equal to 0.39 and less than or equal to 0.41.
2. Tested according to the relevant specification.

LVDS

Depending on whether the device is transmitting an LVDS signal or receiving an LVDS signal, there are two different circuits used for LVDS termination. A sample circuit illustrating a valid termination technique for transmitting LVDS signals appears in **Figure 54**. A sample circuit illustrating a valid termination for receiving LVDS signals appears in **Figure 55**. **Table 38** lists DC voltage specifications. Further information on the specific termination resistor packs shown can be found on **Table 40**.

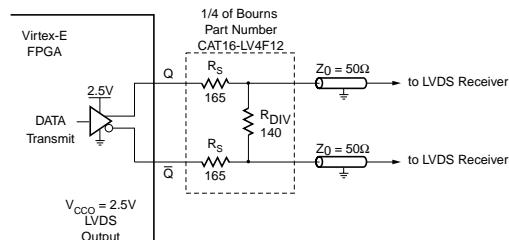


Figure 54: Transmitting LVDS Signal Circuit

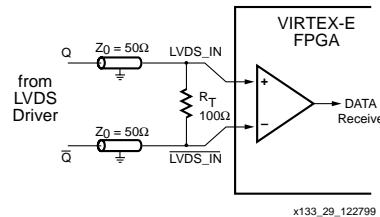


Figure 55: Receiving LVDS Signal Circuit

Table 38: LVDS Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	2.375	2.5	2.625
$V_{ICM}^{(2)}$	0.2	1.25	2.2
$V_{OCM}^{(1)}$	1.125	1.25	1.375
$V_{IDIFF}^{(1)}$	0.1	0.35	-
$V_{ODIFF}^{(1)}$	0.25	0.35	0.45
$V_{OH}^{(1)}$	1.25	-	-
$V_{OL}^{(1)}$	-	-	1.25

Notes:

1. Measured with a 100 ohm resistor across Q and \bar{Q} .
2. Measured with a differential input voltage = \pm 350 mV.

LVPECL

Depending on whether the device is transmitting or receiving an LVPECL signal, two different circuits are used for LVPECL termination. A sample circuit illustrating a valid termination technique for transmitting LVPECL signals appears in [Figure 56](#). A sample circuit illustrating a valid termination for receiving LVPECL signals appears in [Figure 57](#). [Table 39](#) lists DC voltage specifications. Further information on the specific termination resistor packs shown can be found on [Table 40](#).

Table 39: LVPECL Voltage Specifications

Parameter	Min	Typ	Max
V_{CCO}	3.0	3.3	3.6
V_{REF}	-	-	-
V_{TT}	-	-	-
V_{IH}	1.49	-	2.72
V_{IL}	0.86	-	2.125
V_{OH}	1.8	-	-
V_{OL}	-	-	1.57

Notes:

- For more detailed information, see [LVPECL DC Specifications](#)

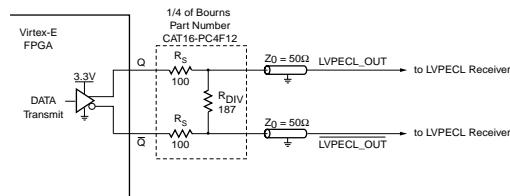


Figure 56: Transmitting LVPECL Signal Circuit

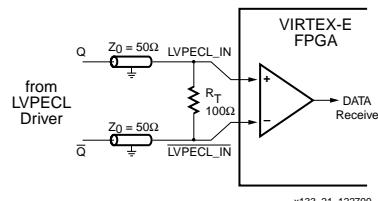


Figure 57: Receiving LVPECL Signal Circuit

Termination Resistor Packs

Resistor packs are available with the values and the configuration required for LVDS and LVPECL termination from Bourns, Inc., as listed in Table. For pricing and availability, please contact Bourns directly at <http://www.bourns.com>.

Table 40: Bourns LVDS/LVPECL Resistor Packs

Part Number	I/O Standard	Term. for:	Pairs/Pack	Pins
CAT16-LV2F6	LVDS	Driver	2	8
CAT16-LV4F12	LVDS	Driver	4	16
CAT16-PC2F6	LVPECL	Driver	2	8
CAT16-PC4F12	LVPECL	Driver	4	16
CAT16-PT2F2	LVDS/LVPECL	Receiver	2	8
CAT16-PT4F4	LVDS/LVPECL	Receiver	4	16

LVDS Design Guide

The SelectI/O library elements have been expanded for Virtex-E devices to include new LVDS variants. At this time all of the cells might not be included in the Synthesis libraries. The 2.1i-Service Pack 2 update for Alliance and Foundation software includes these cells in the VHDL and Verilog libraries. It is necessary to combine these cells to create the P-side (positive) and N-side (negative) as described in the input, output, 3-state and bidirectional sections.

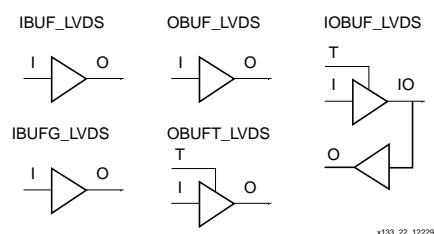


Figure 58: LVDS elements

Creating LVDS Global Clock Input Buffers

Global clock input buffers can be combined with adjacent IOBs to form LVDS clock input buffers. P-side is the GCLKPAD location; N-side is the adjacent IO_LVDS_DLL site.

Table 41: Global Clock Input Buffer Pair Locations

Pkg	GCLK 3		GCLK 2		GCLK 1		GCLK 0	
	P	N	P	N	P	N	P	N
CS144	A6	C6	A7	B7	M7	M6	K7	N8
PQ240	P213	P215	P210	P209	P89	P87	P92	P93
HQ240	P213	P215	P210	P209	P89	P87	P92	P93
BG352	D14	A15	B14	A13	AF14	AD14	AE13	AC13
BG432	D17	C17	A16	B16	AK16	AL17	AL16	AH15
BG560	A17	C18	D17	E17	AJ17	AM18	AL17	AM17
FG256	B8	A7	C9	A8	R8	T8	N8	N9
FG456	C11	B11	A11	D11	YII	AA11	W12	U12
FG676	E13	B13	C13	F14	AB13	AF13	AA14	AC14
FG680	A20	C22	D21	A19	AU22	AT22	AW19	AT21
FG860	C22	A22	B22	D22	AY22	AW21	BA22	AW20
FG900	C15	A15	E15	E16	AK16	AH16	AJ16	AF16
FG115 6	E17	C17	D17	J18	AI19	AL17	AH18	AM18

HDL Instantiation

Only one global clock input buffer is required to be instantiated in the design and placed on the correct GCLKPAD location. The N-side of the buffer is reserved and no other IOB is allowed to be placed on this location.

In the physical device, a configuration option is enabled that routes the pad wire to the differential input buffer located in the GCLKIOB. The output of this buffer then drives the output of the GCLKIOB cell. In EPIC it appears that the second buffer is unused. Any attempt to use this location for another purpose leads to a DRC error in the software.

VHDL Instantiation

```
gclk0_p : IBUFG_LVDS port map
(I=>clk_external, O=>clk_internal);
```

Verilog Instantiation

```
IBUFG_LVDS gclk0_p (.I(clk_external),
.O(clk_internal));
```

Location constraints

All LVDS buffers must be explicitly placed on a device. For the global clock input buffers this can be done with the following constraint in the .ucf or .ncf file.

```
NET clk_external LOC = GCLKPAD3;
```

GCLKPAD3 can also be replaced with the package pin name such as D17 for the BG432 package.

Optional N-side

Some designers might prefer to also instantiate the N-side buffer for the global clock buffer. This allows the top-level net list to include net connections for both PCB layout and system-level integration. In this case, only the output P-side IBUFG connection has a net connected to it. Since the N-side IBUFG does not have a connection in the EDIF net list, it is trimmed from the design in MAP.

VHDL Instantiation

```
gclk0_p : IBUFG_LVDS port map
(I=>clk_p_external, O=>clk_internal);
gclk0_n : IBUFG_LVDS port map
(I=>clk_n_external, O=>clk_internal);
```

Verilog Instantiation

```
IBUFG_LVDS gclk0_p (.I(clk_p_external),
.O(clk_internal));
IBUFG_LVDS gclk0_n (.I(clk_n_external),
.O(clk_internal));
```

Location Constraints

All LVDS buffers must be explicitly placed on a device. For the global clock input buffers this can be done with the following constraint in the .ucf or .ncf file.

```
NET clk_p_external LOC = GCLKPAD3;
```

```
NET clk_n_external LOC = C17;
```

GCLKPAD3 can also be replaced with the package pin name, such as D17 for the BG432 package.

Creating LVDS Input Buffers

An LVDS input buffer can be placed in a wide number of IOB locations. The exact location is dependent on the package that is used. The Virtex-E package information lists the possible locations as IO_L#P for the P-side and IO_L#N for the N-side where # is the pair number.

HDL Instantiation

Only one input buffer is required to be instantiated in the design and placed on the correct IO_L#P location. The N-side of the buffer is reserved and no other IOB is allowed to be placed on this location. In the physical device, a configuration option is enabled that routes the pad wire from the IO_L#N IOB to the differential input buffer located in the IO_L#P IOB. The output of this buffer then drives the output of the IO_L#P cell or the input register in the IO_L#P IOB. In EPIC it appears that the second buffer is unused. Any attempt to use this location for another purpose leads to a DRC error in the software.

VHDL Instantiation

```
data0_p : IBUF_LVDS port map (I=>data(0),
O=>data_int(0));
```

Verilog Instantiation

```
IBUF_LVDS data0_p (.I(data[0]),
.O(data_int[0]));
```

Location Constraints

All LVDS buffers must be explicitly placed on a device. For the input buffers this can be done with the following constraint in the .ucf or .ncf file.

```
NET data<0> LOC = D28; # IO_L0P
```

Optional N-side

Some designers might prefer to also instantiate the N-side buffer for the input buffer. This allows the top-level net list to include net connections for both PCB layout and system-level integration. In this case, only the output P-side IBUF connection has a net connected to it. Since the N-side IBUF does not have a connection in the EDIF net list, it is trimmed from the design in MAP.

VHDL Instantiation

```
data0_p : IBUF_LVDS port map
(I=>data_p(0), O=>data_int(0));
data0_n : IBUF_LVDS port map
(I=>data_n(0), O=>open);
```

Verilog Instantiation

```
IBUF_LVDS data0_p (.I(data_p[0]),
.O(data_int[0]));
IBUF_LVDS data0_n (.I(data_n[0]), .O());
```

Location Constraints

All LVDS buffers must be explicitly placed on a device. For the global clock input buffers this can be done with the following constraint in the .ucf or .ncf file.

```
NET data_p<0> LOC = D28; # IO_L0P
```

```
NET data_n<0> LOC = B29; # IO_L0N
```

Adding an Input Register

All LVDS buffers can have an input register in the IOB. The input register is in the P-side IOB only. All the normal IOB register options are available (FD, FDE, FDC, FDCE, FDP, FDPE, FDR, FDRE, FDS, FDSE, LD, LDE, LDC, LDCE, LDP, LDPE). The register elements can be inferred or explicitly instantiated in the HDL code.

The register elements can be packed in the IOB using the IOB property to TRUE on the register or by using the “map-pr [i|o|b]” where “i” is inputs only, “o” is outputs only and “b” is both inputs and outputs.

To improve design coding times VHDL and Verilog synthesis macro libraries available to explicitly create these structures.

The input library macros are listed in [Table 42](#). The I and IB inputs to the macros are the external net connections.

Table 42: Input Library Macros

Name	Inputs	Outputs
IBUFDS_FD_LVDS	I, IB, C	Q
IBUFDS_FDE_LVDS	I, IB, CE, C	Q
IBUFDS_FDC_LVDS	I, IB, C, CLR	Q
IBUFDS_FDCE_LVDS	I, IB, CE, C, CLR	Q
IBUFDS_FDP_LVDS	I, IB, C, PRE	Q
IBUFDS_FDPE_LVDS	I, IB, CE, C, PRE	Q
IBUFDS_FDR_LVDS	I, IB, C, R	Q
IBUFDS_FDRE_LVDS	I, IB, CE, C, R	Q
IBUFDS_FDS_LVDS	I, IB, C, S	Q
IBUFDS_FDSE_LVDS	I, IB, CE, C, S	Q
IBUFDS_LD_LVDS	I, IB, G	Q
IBUFDS_LDE_LVDS	I, IB, GE, G	Q
IBUFDS_LDC_LVDS	I, IB, G, CLR	Q
IBUFDS_LDCE_LVDS	I, IB, GE, G, CLR	Q
IBUFDS_LDP_LVDS	I, IB, G, PRE	Q
IBUFDS_LDPE_LVDS	I, IB, GE, G, PRE	Q

Creating LVDS Output Buffers

LVDS output buffers can be placed in a wide number of IOB locations. The exact locations are dependent on the package used. The Virtex-E package information lists the possible locations as IO_L#P for the P-side and IO_L#N for the N-side, where # is the pair number.

HDL Instantiation

Both output buffers are required to be instantiated in the design and placed on the correct IO_L#P and IO_L#N locations. The IOB must have the same net source the following pins, clock (C), set/reset (SR), output (O), output clock enable (OCE). In addition, the output (O) pins must be inverted with respect to each other, and if output registers are used, the INIT states must be opposite values (one HIGH and one LOW). Failure to follow these rules leads to DRC errors in software.

VHDL Instantiation

```
data0_p : OBUF_LVDS port map
(I=>data_int(0), O=>data_p(0));
data0_inv: INV      port map
(I=>data_int(0), O=>data_n_int(0));
data0_n : OBUF_LVDS port map
(I=>data_n_int(0), O=>data_n(0));
```

Verilog Instantiation

```

OBUF_LVDS data0_p (.I(data_int[0]),
.O(data_p[0]));

INV      data0_inv (.I(data_int[0]),
.O(data_n_int[0]));

OBUF_LVDS data0_n (.I(data_n_int[0]),
.O(data_n[0]));

```

Location Constraints

All LVDS buffers must be explicitly placed on a device. For the output buffers this can be done with the following constraint in the .ucf or .ncf file.

```

NET data_p<0> LOC = D28; # IO_L0P
NET data_n<0> LOC = B29; # IO_L0N

```

Synchronous vs. Asynchronous Outputs

If the outputs are synchronous (registered in the IOB) then any IO_L#P|N pair can be used. If the outputs are asynchronous (no output register), then they must use one of the pairs that are part of the same IOB group at the end of a ROW or COLUMN in the device.

The LVDS pairs that can be used as asynchronous outputs are listed in the Virtex-E pinout tables. Some pairs are marked as asynchronous-capable for all devices in that package, and others are marked as available only for that device in the package. If the device size might change at some point in the product lifetime, then only the common pairs for all packages should be used.

Adding an Output Register

All LVDS buffers can have an output register in the IOB. The output registers must be in both the P-side and N-side IOBs. All the normal IOB register options are available (FD, FDE, FDC, FDCE, FDP, FDPE, FDR, FDRE, FDS, FDSE, LD, LDE, LDC, LDCE, LDP, LDPE). The register elements can be inferred or explicitly instantiated in the HDL code.

Special care must be taken to insure that the D pins of the registers are inverted and that the INIT states of the registers are opposite. The clock pin (C), clock enable (CE) and set/reset (CLR/PRE or S/R) pins must connect to the same source. Failure to do this leads to a DRC error in the software.

The register elements can be packed in the IOB using the IOB property to TRUE on the register or by using the “map -pr [i|o|b]” where “i” is inputs only, “o” is outputs only and “b” is both inputs and outputs.

To improve design coding times VHDL and Verilog synthesis macro libraries have been developed to explicitly create these structures. The output library macros are listed in

Table 43. The O and OB inputs to the macros are the external net connections.

Table 43: Output Library Macros

Name	Inputs	Outputs
OBUFDS_FD_LVDS	D, C	O, OB
OBUFDS_FDE_LVDS	DD, CE, C	O, OB
OBUFDS_FDC_LVDS	D, C, CLR	O, OB
OBUFDS_FDCE_LVDS	D, CE, C, CLR	O, OB
OBUFDS_FDP_LVDS	D, C, PRE	O, OB
OBUFDS_FDPE_LVDS	D, CE, C, PRE	O, OB
OBUFDS_FDR_LVDS	D, C, R	O, OB
OBUFDS_FDRE_LVDS	D, CE, C, R	O, OB
OBUFDS_FDS_LVDS	D, C, S	O, OB
OBUFDS_FDSE_LVDS	D, CE, C, S	O, OB
OBUFDS_LD_LVDS	D, G	O, OB
OBUFDS_LDE_LVDS	D, GE, G	O, OB
OBUFDS_LDC_LVDS	D, G, CLR	O, OB
OBUFDS_LDCE_LVDS	D, GE, G, CLR	O, OB
OBUFDS_LDP_LVDS	D, G, PRE	O, OB
OBUFDS_LDPE_LVDS	D, GE, G, PRE	O, OB

Creating LVDS Output 3-State Buffers

LVDS output 3-state buffers can be placed in a wide number of IOB locations. The exact locations are dependent on the package used. The Virtex-E package information lists the possible locations as IO_L#P for the P-side and IO_L#N for the N-side, where # is the pair number.

HDL Instantiation

Both output 3-state buffers are required to be instantiated in the design and placed on the correct IO_L#P and IO_L#N locations. The IOB must have the same net source the following pins, clock (C), set/reset (SR), 3-state (T), 3-state clock enable (TCE), output (O), output clock enable (OCE). In addition, the output (O) pins must be inverted with respect to each other, and if output registers are used, the INIT states must be opposite values (one High and one Low). If 3-state registers are used, they must be initialized to the same state. Failure to follow these rules leads to DRC errors in the software.

VHDL Instantiation

```

data0_p:    OBUFT_LVDS port map
(I=>data_int(0), T=>data_tri,
O=>data_p(0));

data0_inv: INV port map
(I=>data_int(0), O=>data_n_int(0));

data0_n:    OBUFT_LVDS port map
(I=>data_n_int(0), T=>data_tri,
O=>data_n(0));

```

Verilog Instantiation

```

OBUFT_LVDS data0_p (.I(data_int[0]),
.T(data_tri), .O(data_p[0]));

INV      data0_inv (.I(data_int[0],
.O(data_n_int[0]));

OBUFT_LVDS data0_n  (.I(data_n_int[0]),
.T(data_tri), .O(data_n[0]));

```

Location Constraints

All LVDS buffers must be explicitly placed on a device. For the output buffers this can be done with the following constraint in the .ucf or .ncf file.

```

NET data_p<0> LOC = D28; # IO_L0P
NET data_n<0> LOC = B29; # IO_L0N

```

Synchronous vs. Asynchronous 3-State Outputs

If the outputs are synchronous (registered in the IOB), then any IO_L#P|N pair can be used. If the outputs are asynchronous (no output register), then they must use one of the pairs that are part of the same IOB group at the end of a ROW or COLUMN in the device. This applies for either the 3-state pin or the data out pin.

LVDS pairs that can be used as asynchronous outputs are listed in the Virtex-E pinout tables. Some pairs are marked as "asynchronous capable" for all devices in that package, and others are marked as available only for that device in the package. If the device size might be changed at some point in the product lifetime, then only the common pairs for all packages should be used.

Adding Output and 3-State Registers

All LVDS buffers can have an output register in the IOB. The output registers must be in both the P-side and N-side IOBs. All the normal IOB register options are available (FD, FDE, FDC, FDCE, FDP, FDPE, FDR, FDRE, FDS, FDSE, LD, LDE, LDC, LDCE, LDP, LDPE). The register elements can be inferred or explicitly instantiated in the HDL code.

Special care must be taken to insure that the D pins of the registers are inverted and that the INIT states of the registers are opposite. The 3-state (T), 3-state clock enable (CE), clock pin (C), output clock enable (CE) and set/reset

(CLR/PRE or S/R) pins must connect to the same source. Failure to do this leads to a DRC error in the software.

The register elements can be packed in the IOB using the IOB property to TRUE on the register or by using the "map -pr [i|o|b]" where "i" is inputs only, "o" is outputs only and "b" is both inputs and outputs.

To improve design coding times VHDL and Verilog synthesis macro libraries have been developed to explicitly create these structures. The input library macros are listed below. The 3-state is configured to be 3-stated at GSR and when the PRE,CLR,S or R is asserted and shares it's clock enable with the output register. If this is not desirable then the library can be updated by the user for the desired functionality. The O and OB inputs to the macros are the external net connections.

Creating a LVDS Bidirectional Buffer

LVDS bidirectional buffers can be placed in a wide number of IOB locations. The exact locations are dependent on the package used. The Virtex-E package information lists the possible locations as IO_L#P for the P-side and IO_L#N for the N-side, where # is the pair number.

HDL Instantiation

Both bidirectional buffers are required to be instantiated in the design and placed on the correct IO_L#P and IO_L#N locations. The IOB must have the same net source the following pins, clock (C), set/reset (SR), 3-state (T), 3-state clock enable (TCE), output (O), output clock enable (OCE). In addition, the output (O) pins must be inverted with respect to each other, and if output registers are used, the INIT states must be opposite values (one HIGH and one LOW). If 3-state registers are used, they must be initialized to the same state. Failure to follow these rules leads to DRC errors in the software.

VHDL Instantiation

```

data0_p: IOBUF_LVDS port map
(I=>data_out(0), T=>data_tri,
IO=>data_p(0), O=>data_int(0));

data0_inv: INV      port map
(I=>data_out(0), O=>data_n_out(0));

data0_n : IOBUF_LVDS port map
(I=>data_n_out(0), T=>data_tri,
IO=>data_n(0), O=>open);

```

Verilog Instantiation

```

IOBUF_LVDS data0_p(.I(data_out[0]),
.T(data_tri), .IO(data_p[0]),
.O(data_int[0]));

INV      data0_inv (.I(data_out[0],
.O(data_n_out[0]));

IOBUF_LVDS
data0_n(.I(data_n_out[0]),.T(data_tri),
.IO(data_n[0]).O());

```

Location Constraints

All LVDS buffers must be explicitly placed on a device. For the output buffers this can be done with the following constraint in the .ucf or .ncf file.

```
NET data_p<0> LOC = D28; # IO_L0P
NET data_n<0> LOC = B29; # IO_L0N
```

Synchronous vs. Asynchronous Bidirectional Buffers

If the output side of the bidirectional buffers are synchronous (registered in the IOB), then any IO_L#P|N pair can be used. If the output side of the bidirectional buffers are asynchronous (no output register), then they must use one of the pairs that is a part of the asynchronous LVDS IOB group. This applies for either the 3-state pin or the data out pin.

The LVDS pairs that can be used as asynchronous bidirectional buffers are listed in the Virtex-E pinout tables. Some pairs are marked as asynchronous capable for all devices in that package, and others are marked as available only for that device in the package. If the device size might change at some point in the product's lifetime, then only the common pairs for all packages should be used.

Adding Output and 3-State Registers

All LVDS buffers can have an output and input registers in the IOB. The output registers must be in both the P-side and

N-side IOBs, the input register is only in the P-side. All the normal IOB register options are available (FD, FDE, FDC, FDCE, FDP, FDPE, FDR, FDRE, FDS, FDSE, LD, LDE, LDC, LDCE, LDP, LDPE). The register elements can be inferred or explicitly instantiated in the HDL code. Special care must be taken to insure that the D pins of the registers are inverted and that the INIT states of the registers are opposite. The 3-state (T), 3-state clock enable (CE), clock pin (C), output clock enable (CE), and set/reset (CLR/PRE or S/R) pins must connect to the same source. Failure to do this leads to a DRC error in the software.

The register elements can be packed in the IOB using the IOB property to TRUE on the register or by using the “map -pr [i|o|b]” where “i” is inputs only, “o” is outputs only and “b” is both inputs and outputs. To improve design coding times VHDL and Verilog synthesis macro libraries have been developed to explicitly create these structures. The bidirectional I/O library macros are listed in [Table 44](#). The 3-state is configured to be 3-stated at GSR and when the PRE, CLR, S or R is asserted and shares its clock enable with the output and input register. If this is not desirable then the library can be updated by the user for the desired functionality. The I/O and IOB inputs to the macros are the external net connections.

Table 44: Bidirectional I/O Library Macros

Name	Inputs	Bidirectional	Outputs
IOBUFDS_FD_LVDS	D, T, C	IO, IOB	Q
IOBUFDS_FDE_LVDS	D, T, CE, C	IO, IOB	Q
IOBUFDS_FDC_LVDS	D, T, C, CLR	IO, IOB	Q
IOBUFDS_FDCE_LVDS	D, T, CE, C, CLR	IO, IOB	Q
IOBUFDS_FDP_LVDS	D, T, C, PRE	IO, IOB	Q
IOBUFDS_FDPE_LVDS	D, T, CE, C, PRE	IO, IOB	Q
IOBUFDS_FDR_LVDS	D, T, C, R	IO, IOB	Q
IOBUFDS_FDRE_LVDS	D, T, CE, C, R	IO, IOB	Q
IOBUFDS_FDS_LVDS	D, T, C, S	IO, IOB	Q
IOBUFDS_FDSE_LVDS	D, T, CE, C, S	IO, IOB	Q
IOBUFDS_LD_LVDS	D, T, G	IO, IOB	Q
IOBUFDS_LDE_LVDS	D, T, GE, G	IO, IOB	Q
IOBUFDS_LDC_LVDS	D, T, G, CLR	IO, IOB	Q
IOBUFDS_LDCE_LVDS	D, T, GE, G, CLR	IO, IOB	Q
IOBUFDS_LDP_LVDS	D, T, G, PRE	IO, IOB	Q
IOBUFDS_LDPE_LVDS	D, T, GE, G, PRE	IO, IOB	Q

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/7/99	1.0	Initial Xilinx release.
1/10/00	1.1	Re-released with spd.txt v. 1.18, FG860/900/1156 package information, and additional DLL, Select RAM and SelectI/O information.
1/28/00	1.2	Added Delay Measurement Methodology table, updated SelectI/O section, Figures 30, 54, & 55, text explaining Table 5, T_{BYP} values, buffered Hex Line info, p. 8, I/O Timing Measurement notes, notes for Tables 15, 16, and corrected F1156 pinout table footnote references.
2/29/00	1.3	Updated pinout tables, V_{CC} page 20, and corrected Figure 20.
5/23/00	1.4	Correction to table on p. 22.
7/10/00	1.5	<ul style="list-style-type: none"> • Numerous minor edits. • Data sheet upgraded to Preliminary. • Preview -8 numbers added to Virtex-E Electrical Characteristics tables.
8/1/00	1.6	<ul style="list-style-type: none"> • Reformatted entire document to follow new style guidelines. • Changed speed grade values in tables on pages 35-37.

Date	Version	Revision
9/20/00	1.7	<ul style="list-style-type: none"> Min values added to Virtex-E Electrical Characteristics tables. XCV2600E and XCV3200E numbers added to Virtex-E Electrical Characteristics tables (Module 3). Corrected user I/O count for XCV100E device in Table 1 (Module 1). Changed several pins to "No Connect in the XCV100E" and removed duplicate V_{CCINT} pins in Table ~ (Module 4). Changed pin J10 to "No connect in XCV600E" in Table 74 (Module 4). Changed pin J30 to "VREF option only in the XCV600E" in Table 74 (Module 4). Corrected pair 18 in Table 75 (Module 4) to be "AO in the XCV1000E, XCV1600E".
11/20/00	1.8	<ul style="list-style-type: none"> Upgraded speed grade -8 numbers in Virtex-E Electrical Characteristics tables to Preliminary. Updated minimums in Table 13 and added notes to Table 14. Added note 2 to Absolute Maximum Ratings. Changed speed grade -8 numbers for T_{SHCKO32}, T_{REG}, T_{BCCS}, and T_{ICKOF}. Changed all minimum hold times to -0.4 under Global Clock Set-Up and Hold for LVTTL Standard, with DLL. Revised maximum T_{DLLPW} in -6 speed grade for DLL Timing Parameters. Changed GCLK0 to BA22 for FG860 package in Table 46.
2/12/01	1.9	<ul style="list-style-type: none"> Revised footnote for Table 14. Added numbers to Virtex-E Electrical Characteristics tables for XCV1000E and XCV2000E devices. Updated Table 27 and Table 78 to include values for XCV400E and XCV600E devices. Revised Table 62 to include pinout information for the XCV400E and XCV600E devices in the BG560 package. Updated footnotes 1 and 2 for Table 76 to include XCV2600E and XCV3200E devices.
4/02/01	2.0	<ul style="list-style-type: none"> Updated numerous values in Virtex-E Switching Characteristics tables. Converted data sheet to modularized format. See the Virtex-E Data Sheet section.
4/19/01	2.1	<ul style="list-style-type: none"> Modified Figure 30 "DLL Generation of 4x Clock in Virtex-E Devices."

Virtex-E Data Sheet

The Virtex-E Data Sheet contains the following modules:

- DS022-1, Virtex-E 1.8V FPGAs:
[Introduction and Ordering Information \(Module 1\)](#)
- DS022-2, Virtex-E 1.8V FPGAs:
[Functional Description \(Module 2\)](#)
- DS022-3, Virtex-E 1.8V FPGAs:
[DC and Switching Characteristics \(Module 3\)](#)
- DS022-4, Virtex-E 1.8V FPGAs:
[Pinout Tables \(Module 4\)](#)

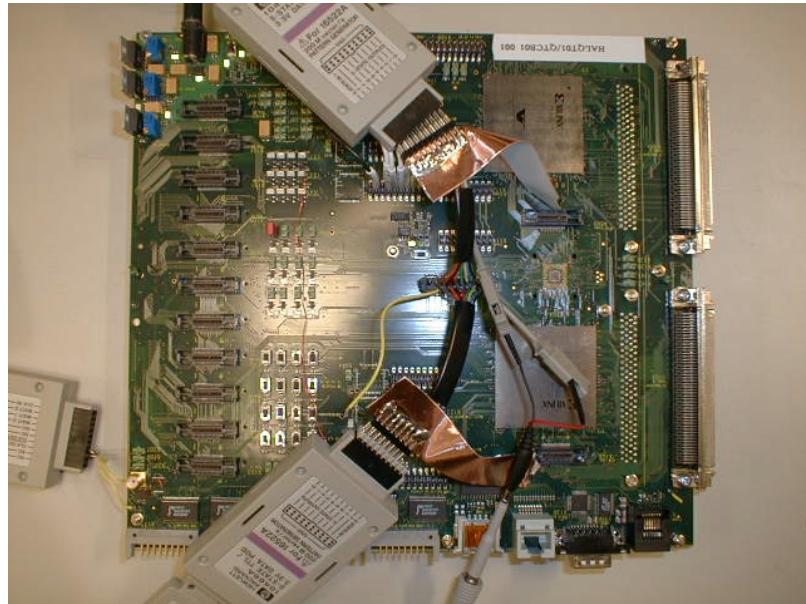


Figure E.1 Custom-Built Xilinx FPGA Test Board

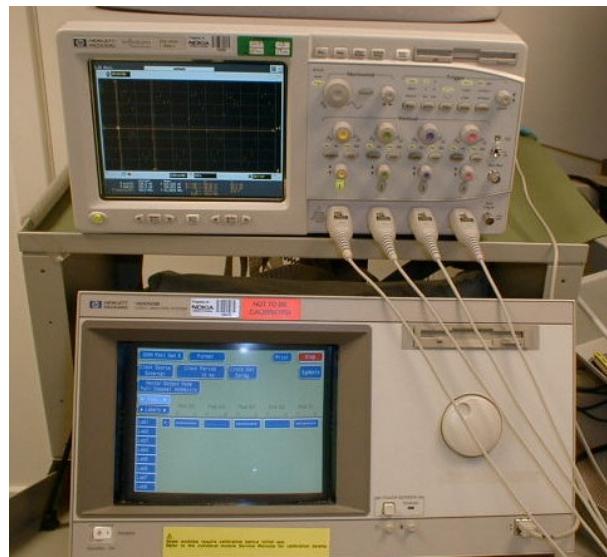


Figure E.2 Xilinx FPGA Test Setup

APPENDIX-G

REFERENCES

BOOKS

- [1] *Bluetooth Revealed — An Insider's Guide to the Open Specification for Global Wireless Communications*, Brent A. Miller, Chatschick Bisdikian, Prentice-Hall Inc., ©2000.
- [2] *Digital Communications*, 4th Ed., John G. Proakis, McGraw-Hill Inc., ©2001.
- [3] *Digital Communications — Fundamentals and Applications*, 2nd Ed., Bernard Sklar, Prentice-Hall Inc., ©2001.
- [4] *Communication Systems*, 4th Ed., Simon Haykin, John Wiley & Sons Inc., ©2001.
- [5] *Modern Digital and Analog Communication Systems*, B. P. Lathi, Oxford University Press, ©1998.
- [6] *Analog & Digital Communication Systems*, 4th Ed., Martin S. Roden, Prentice-Hall Inc., ©1995.
- [7] *Introduction to Communication Systems*, 2nd Ed., Ferrel G. Stremler, Addison-Wesley Publishing Company, ©1990.

- [8] *Communications Receivers — DSP, Software Radios, and Design*, 3rd Ed., Ulrich Rohde, Jerry Whitaker, McGraw-Hill Inc., ©2001.
- [9] *RF and Microwave Circuit Design for Wireless Communications*, Lawrence E. Larson, Artech House Publishers, 1996.
- [10] *Integrated Converters — D/A & A/D Architectures, Analysis & Simulation*, Paul G. A. Jespers, Oxford University Press, ©2001.
- [11] *Analog Integrated Circuits*, David Johns, Kenneth W. Martin, John Wiley & Sons Inc., ©1996.
- [12] *Analog-Digital ASICs — Circuit Techniques, Design Tools & Applications*, R. Soin, F. Maloberti, J. Franca, Peter Peregrinus Ltd., ©1991.
- [13] *Discrete-Time Signal Processing*, Alan V. Oppenheim, Ronald W. Schafer, John R. Buck, Prentice-Hall Inc., ©1999.
- [14] *Digital Signal Processing — Principles, Architectures, and Applications*, John G. Proakis, Dimitris G. Manolakis, Prentice-Hall, Inc., 1996.
- [15] *Digital Filters — Analysis, Design & Applications*, Andreas Antoniou, McGraw-Hill, Inc., 1993.
- [16] *Digital Signal Processing — A Practical Approach*, Emmanuel C. Ifeachor, Barrie W. Jervis, Addison-Wesley Longman Limited, 1993.
- [17] *DSP Integrated Circuits*, Lars Wanhammar, Academic Press, ©1999.
- [18] *VLSI Digital Signal Processing Systems*, Keshab K. Parhi, John Wiley & Sons Inc., ©1999.
- [19] *The Designer's Guide to VHDL*, Peter J. Ashenden, Morgan-Kaufmann Publishers Inc., ©1996.
- [20] *Analysis and Design of Digital Systems with VHDL*, Allen Dewey, PWS Publishing Company, ©1997.
- [21] *System-on-a-Chip Design and Test*, Rochit Rajsuman, Artech House, ©2000.
- [22] *Digital Systems Testing and Testable Design*, Miron Abramovici, Melvin A. Breuer, Arthur D. Friedman, IEEE Press, ©1993.
- [23] *Digital Circuit Testing & Testability*, Parag K. Lala, Academic Press, ©1995.

ARTICLES

- [23a] *Theory of Spread Spectrum Communications — A Tutorial*, D. L. Schilling, L. B. Milstein, R. L. Pickholtz, IEEE Transactions on Communication, vol. COM-30, May 1982.
- [24] *Digital Filter Sets for Frequency Shift Keyed Modems*, A. P. Breen, C. C. Goodyear, IEEE Proceedings, 1989.
- [25] *Design of Digital Discriminator Filters for Voiceband FSK Data Modems*, C. C. Goodyear, P. M. Hughes, M. Rahim, IEEE Proceedings, 1989.
- [26] *Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review*, Stanley A. White, IEEE ASSP Magazine, © July 1989.

- [27] *An All-CMOS Architecture for a Low-Power, Frequency-Hopped 900 MHz, Spread Spectrum Transceiver*, Jonathan Min, Henry Samueli, Ahmad Reza Rofougaran, Asad A. Abidi, IEEE Custom Integrated Circuits Conference, 1994.
- [28] *A Low-Power Baseband Receiver I.C. for Frequency-Hopped Spread Spectrum Applications*, Jonathan Min, Henry Samueli, Huan-Chang Liu, IEEE Custom Integrated Circuits Conference, 1995.
- [29] *A Novel, Pure Digital Signal Processing, NonCoherent Receiver based on Filter Bank Realization for Frequency Shift Keying*, Tsai-Pao Lee, Kwang-Cheng Chen, IEEE, 1995.
- [30] *Minimizing Power Consumption in Digital CMOS Circuits*, Anantha P. Chandrakasan & Robert W. Brodersen, Proceedings of the IEEE, April, 1995.
- [31] *FPGA-based Rapid Prototyping: An Overview*, Helena Krupnova, Gabriele Saucier (Institut National Polytechnique de Grenoble, France), Proceedings of BEC, 2000.
- [32] *Bluetooth's Slow Dawn*, Ron Schneiderman, IEEE Spectrum, ©2000
- [33] *Bluetooth Radio Architectures*, James P. K. Gilb, IEEE Radio Frequency Integrated Circuits Symposium 2000.
- [34] *BlueCore™01 Single Chip Bluetooth System — Product Overview Data Sheet*, Cambridge Silicon Radio Ltd., UK, ©2001.
- [35] *A Fully-Integrated Single-Chip SOC for Bluetooth*, Frank Op't Eynde et al.(Alcatel Microelectronics, Belgium & Turkey & TTP Communications, UK), IEEE International Solid-State Circuits Conference, 2001.
- [36] *A 2.4 GHz CMOS Transceiver for Bluetooth*, H. Darabi et al. (Broadcom Corporation, USA), IEEE International Solid-State Circuits Conference, 2001.
- [37] *A 22mW Bluetooth RF Transceiver with Direct RF Modulation and On-Chip IF Filtering*, Norm Filliol et al. (Conexant Systems Inc., Canada & Theta Microelectronics, Greece & University of Oulu, Finland), IEEE International Solid-State Circuits Conference, 2001.

MANUALS

- [38] *Bluetooth Specification*, Bluetooth Special Interest Group.
- [39] *MATLAB User Guide*, MathWorks Inc., ©2000
- [40] *Simulink User Guide*, MathWorks Inc., ©2000
- [41] *Communication Toolbox User Guide*, MathWorks Inc., ©2000
- [42] *Signal Processing Toolbox User Guide*, MathWorks Inc., ©2000
- [43] *Filter Design Toolbox User Guide*, MathWorks Inc., ©2000
- [44] *Communication Blockset User Guide*, MathWorks Inc., ©2000
- [45] *DSP Blockset User Guide*, MathWorks Inc., ©2000
- [46] *Fixed-Point Blockset User Guide*, MathWorks Inc., ©2000
- [47] *Xilinx Blockset Reference Guide*, Xilinx Inc., ©2000.
- [48] *Xilinx System Generator Quick Start Guide*, Xilinx Inc., ©2000.

- [49] *Xilinx CORE Generator User Guide*, Xilinx Inc., ©1999.
- [50] *Xilinx LogiBLOX Guide*, Xilinx Inc., ©2000.
- [51] *Synopsys FPGA Compiler II/FPGA Express VHDL Reference Manual*, Synopsys Inc., ©1999.
- [52] *Xilinx Design Manager/Flow Engine Guide*, Xilinx Inc., ©2000.
- [53] *Xilinx Chip Scope Guide*, Xilinx Inc., ©2000.
- [54] *Xilinx FPGA Editor Guide*, Xilinx Inc., ©2000.
- [55] *Xilinx Hardware Debugger Guide*, Xilinx Inc., ©2000.
- [56] *Xilinx Hardware User Guide*, Xilinx Inc., ©2000.
- [57] *Getting Started with the MultiLINUX Cable*, Application Note: HW-MultiLINUX, Carl Carmichael, XAPP168 (v 1.0), October 6, 1999

WEB SITES

- [58] *Bluetooth Website* www.bluetooth.com
- [59] *Berkeley Design Technology Inc. Website* www.bdti.com
- [60] *Synopsys Inc. Website* www.synopsys.com
- [61] *Cadence Inc. Website* www.cadence.com
- [62] *Mentor Graphics Inc. Website* www.mentorgraphics.com
- [63] *Hewlett-Packard Inc. Website* www.agilent.com
- [64] *Hyperception Inc. Website* www.hypersignal.com
- [65] *Signalogic Inc. Website* www.signalogic.com
- [66] *Elanix Inc. Website* www.elanix.com
- [67] *Mathworks Inc. Website* www.mathworks.com
- [68] *Xilinx Inc. Website* www.xilinx.com