P- 3451

# A RECONFIGURABLE APPLICATION SPECIFIC INSTRUCTION PROCESSOR IN SDR ENVIRONMENT


**By**

**P.BALAKRISHNAN**

**Reg No. 0920106003**

of

**KUMARAGURU COLLEGE OF TECHNOLOGY**

(An Autonomous Institution affiliated to Anna University, Coimbatore)

**COIMBATORE - 641049**


**A PROJECT REPORT**

*Submitted to the*

**FACULTY OF ELECTRONICS AND COMMUNICATION ENGINEERING**

*In partial fulfillment of the requirements*

*for the award of the degree*

of

**MASTER OF ENGINEERING**

**IN**

**APPLIED ELECTRONICS**

**APRIL 2011**

## BONAFIDE CERTIFICATE

Certified that this project report entitled **"A RECONFIGURABLE APPLICATION SPECIFIC INSTRUCTION PROCESSOR IN SDR ENVIRONMENT"** is the bonafide work of **Mr.P.BALAKRISHNAN (Reg. no. 0920106003)** who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Project Guide

**Ms. S.Sasikala**

Head of the Department

**Dr. Rajeswari Mariappan**

The candidate with university Register no. 0920106003 is examined by us in the project viva-voce examination held on ...21.4.2011...

Internal Examiner

External Examiner

# ACKNOWLEDGEMENT

A project of this nature needs co-operation and support from many for successful completion. In this regard, I am fortunate to express my heartfelt thanks to Chairman **Padmabhusan Arutselvar Dr.N.Mahalingam B.Sc.,F.I.E.,** and Co-Chairman **Dr.B.K.Krishnaraj Vanavarayar B.Com,B.L.,** for providing necessary facilities throughout the course.

I would like to express my thanks and appreciation to many people who have contributed to the successful completion of this project. First I thank **Dr.J.Shanmugam Ph.D,** Director, for providing us an opportunity to carry out this project work.

I would like to thank **Dr.S.Ramachandran Ph.D,** Principal, who gave his continual support and opportunity for completing the project work successfully.

I would like to thank **Dr.Rajeswari Mariappan Ph.D,** Prof of Head, Department of Electronics and Communication Engineering, who gave her continual support for us throughout the course of study.

I would like to thank **Ms. S.Sasikala M.Tech.,** Asst. Professor, Project guide for his technical guidance, constructive criticism and many valuable suggestions provided throughout the project work.

My heartfelt thanks to **Ms.R.Latha M.E (Ph.D),** Associate Professor, Project coordinator, for her contribution and innovative ideas at various stages of the project to successfully complete this work.

I express my sincere gratitude to my family members, friends and to all my staff members of Electronics and Communication Engineering Department for their support throughout the course of my project.

# ABSTRACT

The software defined radio is to produce seamless communication devices which can support different services. Reconfigurable Application Specific Instruction Set Processors (RASIP) provides complete sets of instructions for reconfiguring hardware for user specific communication interface. The terminal must adapt their hardware structure in function of the wireless network such as CDMA IS-95, GSM etc. In the reconfiguration flow, to validate RASIP architecture where Partial Reconfiguration is used to dynamically reconfigure the requested block.

The instruction set design for the RASIP for achieving SDR functionality has been described and modeled for proposed architecture. The new concept of the user defined instructions that are used for upgrade and modification of existing instruction set after designing the RASIP to incorporate new functionality. This flexibility is essential for any SDR to incorporate new standards in future.

## TABLE OF CONTENTS


| CHAPTER NO | TITLE | PAGE NO |
|---|---|---|
| | **ABSTRACT** | **ii** |
| | **LIST OF FIGURES** | **v** |
| | **LIST OF TABLES** | **vi** |
| | **LIST OF ABBREVIATIONS** | **vii** |
| 1 | **INTRODUCTION** | **1** |
| | 1.1 Software Defined Radio | 1 |
| | 1.2 Application Specific Instruction Processor | 3 |
| 2 | **ASIP ARCHITECTURE** | **8** |
| | 2.1 ASIP Implementation | 8 |
| | 2.2 Architecture Design of ASIP for SDR | 9 |
| 3 | **HARDWARE DESCRIPTION LANGUAGE** | **11** |
| | 3.1 Introduction | 11 |
| | 3.2 Advantages of HDL | 11 |
| | 3.3 VHDL | 12 |
| | 3.3.1 Structural Descriptions | 13 |
| | 3.3.2 Data Flow Descriptions | 15 |
| | 3.3.3 Behavioral Descriptions | 16 |
| | 3.4 Software Used | 17 |
| | 3.5 EDA Tool | 17 |

| | | |
|---|---|---|
| 4 | **DESIGN OF RASIP** | **18** |
| | 4.1 Instruction Set Design | 18 |
| | 4.2 Instruction Set Format | 19 |
| | 4.3 Design of DDC | 21 |
| | 4.4 Design of CIC Filter | 24 |
| | 4.5 Design of DUC | 25 |
| | 4.6 Design of Digital Filter | 27 |
| 5 | **VHDL DESIGN AND IMPLEMENTATION** | **32** |
| | 5.1 Communication System Design Instruction | 32 |
| | 5.2 Numerical Controlled Oscillator | 34 |
| 6 | **DESIGN OF FFT** | **36** |
| | 6.1 Implementation of 8 Point FFT | 36 |
| | 6.2 Design of General Radix-2 FFT using VHDL | 37 |
| | 6.3 FFT Implementation States | 38 |
| 7 | **DESIGN OF MODULATION** | **40** |
| 8 | **RESULTS** | **42** |
| | 8.1 Design Software | 42 |
| | 8.2 Simulation Results Obtained for ALU | 42 |
| | 8.3 Simulation Results Obtained for MAC | 43 |
| | 8.4 Simulation Results Obtained for Sin and Cos | 43 |
| | 8.5 Simulation Result Obtained for Start State | 44 |
| | 8.6 Simulation Result Obtained for Load State | 44 |
| | 8.7 Simulation Result Obtained for Run State | 45 |
| | 8.8 Simulation Result Obtained for AM | 45 |

## LIST OF FIGURES


| FIGURE NO | CAPTION | PAGE NO |
|---|---|---|
| 1.1 | Block Diagram of a Generic Digital Transceiver | 2 |
| 1.2 | Application Instruction Processor Architecture | 5 |
| 2.1 | Proposed Architecture of ASIP for ADR | 10 |
| 3.1 | Schematic SR Latch | 14 |
| 3.2 | Data flow approach in SR Latch | 15 |
| 4.1 | Instruction Set format of general instruction | 19 |
| 4.2 | Digital down Converter Block diagram | 22 |
| 4.3 | CIC Filter Block Diagram | 24 |
| 4.4 | Digital up Converter Block Diagram | 26 |
| 4.5 | Block Diagram of Pulse Shaping Block in DUC | 27 |
| 4.6 | Various Realizations of FIR Filters | 28 |
| 4.7 | FIR Filter Realization Using MAC | 30 |
| 4.8 | Modified Canonical Form Realization of IIR Filters | 31 |
| 5.1 | Numerical Controlled Oscillator | 34 |
| 6.1 | Pipeline architecture of 8 point FFT | 36 |
| 6.2 | Scheduling Diagram of stage one - FFT | 37 |
| 7.1 | Universal Modulator using CORDIC | 40 |
| 8.1 | Simulation Result Obtained For ALU | 42 |
| 8.2 | Simulation Results Obtained for MAC | 43 |
| 8.3 | Simulation Results Obtained For Sin and Cos | 43 |
| 8.4 | Simulation Results Obtained for START State | 44 |
| 8.5 | Simulation Results Obtained for LOAD State | 44 |
| 8.6 | Simulation Results Obtained for Run State | 45 |
| 8.7 | Simulation Results Obtained for AM | 45 |

9 **CONCLUSION AND FUTURE SCOPE** **46**

9.1 Conclusion 46

9.2 Future Scope 46

**REFERENCES** **47**

**APPENDIX** **49**

## LIST OF TABLES


| TABLE NO | CAPTION | PAGE NO |
|---|---|---|
| 4.1 | Encoding of op-code field in the first, second and third group instructions | 20 |
| 5.1 | Sin/Cos outputs for common angles | 33 |

## LIST OF ABBREVATIONS

| | | |
|---|---|---|
| SDR | ---- | Software Defined Radio |
| ASIP | ---- | Application Specific Instruction Processor |
| DDC | ---- | Digital Down Conversion |
| DUC | ---- | Digital Up Conversion |
| ASIC | ---- | Application Specific Integrated Circuits |
| NCO | ---- | Numerical Controlled Oscillator |
| ALU | ---- | Arithmetic and Logic Unit |
| HDL | ---- | Hardware Description Language |
| FPGA | ---- | Field Programmable Gate Array |
| RTL | ---- | Register Transfer Level |
| EDA | ---- | Electronic Design Automation |

# CHAPTER 1

# INTRODUCTION

## 1.1 SOTWARE DEFINED RADIO

Software-Defined Radio (SDR) is a rapidly evolving technology that is receiving enormous recognition and generating widespread interest in the telecommunication industry. The analog radio systems are being replaced by digital radio systems for various radio applications in military, civilian and commercial spaces. In addition to this programmable hardware modules are increasingly being used in digital radio systems at different functional levels. SDR technology aims to take advantage of these programmable hardware modules to build open-architecture based radio system software.

SDR technology facilitates implementation of some of the functional modules in a radio system such as modulation, demodulation, signal generation, coding and link-layer protocols in software. This helps in building reconfigurable software radio systems where dynamic selection of parameters for each of the above-mentioned functional a module is possible. A complete hardware based radio system has limited utility since parameters for each of the functional modules are fixed. A radio system built using SDR technology extends the utility of the system for a wide range of applications that use different link-layer protocols and modulation/demodulation techniques.

Commercial wireless communication industry is currently facing problems due to constant evolution of link-layer protocol standards of 2.5G, 3G, and 4G, existence of incompatible wireless network technologies in different countries inhibiting deployment of global roaming facilities and problems in rolling-out new services due to wide-spread presence of legacy subscriber handsets. SDR technology promises to solve these problems by implementing the radio functionality as software modules running on a generic hardware platform. Further, multiple software modules implementing different standards can be present in the radio system.

The system can take up different personalities depending on the software module being used. Also, the software modules that implement new services/features can be downloaded over-the-air onto the handsets. This kind of flexibility offered by SDR systems helps in dealing with problems due to differing standards and issues related to deployment of new service.

The overview of a basic conventional digital radio system and then how SDR technology can be used to implement radio functions in software. It then explains the software architecture of SDR. The various functional blocks in a generic digital radio transceiver system is depicted in figure 1.1. The digital radio system consists of three main functional blocks: RF section, IF section and baseband section. The RF section consists of essentially analog hardware modules while IF and baseband sections contain digital hardware modules.

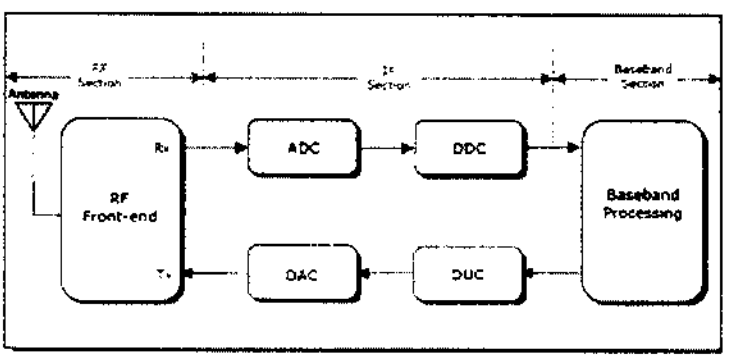



**Figure 1.1: Block Diagram of a Generic Digital Transceiver**

The RF section is responsible for transmitting or receiving the radio frequency signal from the antenna through a coupler and converting the RF signal to an intermediate frequency signal. The RF front-end on the receive path performs RF amplification and analog down conversion from RF to IF. On the transmit path, RF front-end performs analog up conversion and RF power amplification.

The ADC/DAC blocks perform analog-to-digital conversion on receive path and digital-to analog conversion on transmit path respectively. ADC/DAC blocks interface between the analog and digital sections of the radio system. DDC/DUC blocks perform digital-down conversion on receive path and digital-up-conversion on transmit path

respectively. DUC/DDC blocks essentially perform modem operations, modulation of the signal on transmit path and demodulation of the signal on receive path.

The baseband section performs baseband operations of connection setup, equalization, frequency hopping, timing recovery and correlation. Also implements the link layer protocol. The DDC/DUC and baseband processing operations require large computing power and these modules are generally implemented using ASICs or stock DSPs. Implementation of the digital sections using ASICs results in fixed-function digital radio systems. If DSPs are used for baseband processing, a programmable digital radio (PDR) system can be realized.

The limitation of this system is that any change made to the RF section of the system will impact the DDC/DUC operations and will require non-trivial changes to be made in DDC/DUC ASICs. A software-defined radio system is one in which the baseband processing as well as DDC/DUC modules are programmable. Availability of smart antennas, wideband RF front-end, wideband ADC/DAC technologies and ever increasing processing capacity of DSPs and general-purpose microprocessors have fostered the development of multi-band, multi-standard, multi-mode radio systems using SDR technology.

In an SDR system, the link-layer protocols and modulation/demodulation operations are implemented in software. If the programmability is further extended to the RF section, ideal software radio systems that support programmable RF bands can be implemented. However, the current state-of-the-art ADC/DAC devices cannot support the digital bandwidth, dynamic range and sampling rate required to implement this in a commercially viable manner.

## 1.2 APPLICATION SPECIFIC INSTRUCTION PROCESSOR

ASIP is Application Specific Instruction-set Processor dedicated designed for an application domain. ASIP instruction set is specifically designed to accelerate heavy and most used functions. ASIP architecture is designed to implement the assembly instruction set with minimum hardware cost. ASIP DSP is an application specific digital signal processor for iterative data manipulation and transformation extensive applications.

The instruction set must be general enough to support general applications. The compiler should offer compilation for all programs and to adapt all programmers coding behaviors. The biggest challenges for ASIP design are the silicon cost and power consumption. Based on the carefully specified function coverage, the goal of an ASIP design is to reach the highest performance over silicon, over power consumption, as well over the design cost. The requirement on flexibility should be sufficient instead of ultimate. The performance is application specific instead of the highest one.

The first step in constructing an ASIP is to analyze the application that is to be executed. A simple analysis can yield characteristics such as the amount of memory required and the number and type of operations. Designing the ASIP architecture is an iterative process whereby the processor architecture and the algorithm are progressively refined. The cost of candidate processor architecture may be estimated relatively easily by counting the resources used. The performance of the algorithm can be assessed by a more detailed analysis of implementation of the key sections of the application. Reaching the optimal implementation using this approach may, in general, require a number of iterations. However, we find in practice that an acceptable architecture can be designed for many applications relatively quickly. For applications that have been implemented in software or in previous hardware generations, modern FPGAs often provide vastly more computational power than is required.

ASIPs represent a hybrid approach between the paradigms of general-purpose processor and Application-Specific Integrated Circuits (ASICs) and achieve performance levels significantly higher than the performance obtained with ASIC and same flexibility that offered by general-purpose processors like Digital Signal Processors (DSPs), Microprocessors and Micro-controllers. ASIPs combine a software programmable processor and application-specific hardware component that can be used in a number of times in an instruction pipeline passion. The instruction-set aims complete operation of SDR, which is the implementation of wireless network protocols, whose operation modes can be changed after design and development. This requires an inherent parallelism of functional HW.

The recent trend in mobile communication is to realize a common worldwide communication interface by which user may access network operators and services based on their universal profile in any country of the world using single device. The main problem for a user to access different network is due to non-flexibility of terminals. SDR system can provide flexibility to control the same hardware resources via software for multi communication protocol. To decide, more suitable station for reconfiguration, downloading methodology of reconfigurable bit streams is still a problem for designers.

Today's design is third generation (3G) technology, in which base station provides flexibility for selection via software of companion second generation (2G) and third generation (3G) technology. The base station can benefit from more accelerated and enriched adoption of SDR with any initial implementations. The new technology is required due to demand of functionality by users on single device, power

Available resources are suitably considered so that they could be effectively utilized in the proposed architecture shown in Figure 1.2. The available chipset is Xilinx Virtex-II Pro (XCV2PI30/50). It is embedded with two hard-core power PCs (PPCs) on the chip.

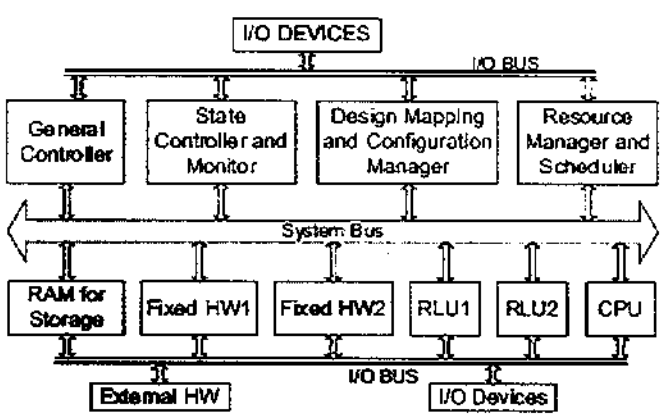



**Figure 1.2: Application Instruction Processor Architecture**

The proposed architecture is pure general with reconfiguration facility contains the following sub modules:

A. *State Controller and Monitor*

This module controls the FSM execution of the system and monitors each state status and their controlling part. Its basic function is monitoring each status of every register that will be used in the scheduling, resource manager and configuration manager.

B. *Resource manager and scheduler*

Depending upon the application, computation requirement and status register contents displaced by State Controller, scheduler makes scheduled task list.

C. *Design Mapping and Configuration Manager*

This module maps the design library onto the RLUs according to the list made by the scheduler.

D. *General Controller*

In this module the entire controlling part implemented, which is necessary for the system level controlling and communication controlling among the different modules in the system.

*E. Fixed hardware*

This module implements the floating-point Arithmetic Logic Unit (ALU). Example: float point adder, subtraction, multiplier, comparator and logical functions.

*F. RLU1 and RLU2*

These are two different locations in a Reconfigurable Logic Unit on a FPGA (Virtex-II Pro chip), which will be implemented basically performance oriented functionality in terms of the silicon area, fast execution, by implementing one of the best among the available design library for the same functionality.

*G. RAM for Data Storage and User program*

This is basically a memory block that stores different data (I/O and intermediate data) and configuration's bit streams for user defined instructions. This block also stores constants required for the computations and look-up table.

*H. PROM for functional library Storage*

In this memory, the entire functional libraries in the form of configuration's bit streams are stored.

# CHAPTER 2

# ASIP ARCHITECTURE

## 2.1 ASIP IMPLEMENTATION

As the SDR functionality is real-time task, due to complexity involved in its algorithm and to improve flexibility and enhance ability to implement multiple and new coming standards an ASIP based approach has been chosen to give as flexibility as general-purpose processor and almost same performance as ASIC.ASIPs represent a hybrid approach between the paradigms of general-purpose processor and Application-Specific Integrated Circuits (ASICs) and achieve performance levels significantly higher than the performance obtained with ASIC and same flexibility that offered by general-purpose processors like Digital Signal Processors (DSPs), Microprocessors and Micro-controllers.

ASIPs combine a software programmable processor and application-specific hardware component that can be used in a number of times in an instruction pipeline passion. Our instruction-set aims complete operation of SDR, which is the implementation of wireless network protocols, whose operation modes can be changed after design and development. The recent trend in mobile communication is to realize a common worldwide communication interface by which user may access network operators and services based on their universal profile in any country of the world using single device.

The main problem for a user to access different network is due to non-flexibility of terminals. SDR system can provide flexibility to control the same hardware resources via software for multi communication protocol. Today's design is third generation (3G) technology, in which base station provides flexibility for selection via software of companion second generation (2G) and third generation (3G) technology.

## 2.2 ARCHITECTURE DESIGN OF ASIP FOR SDR

As the SDR functionality is real-time task, due to complexity involved in its algorithm and to improve flexibility and enhance ability to implement multiple and new coming standards an ASIP based approach has been chosen to give as flexibility as general-purpose processor and almost same performance as ASIC.

The architecture of the proposed block diagram is shown in Figure 2.1. Our proposed architecture has single-bus architecture, for data and some dedicated paths used wherever necessary. The diagram shows the two kinds of special functional block diagrams: (i) Block diagrams calling them as base-band and Channel-coding sub-functionalities such as Modulation and Demodulation schemes, Channel coding and decoding, DUC, DDC, NOC, FIR/IIR/CIC, and FFT. These functional block diagrams are useful for the base-band coding and channel coding (ii) Special functions but general to any communication system are shown as EU in the architecture, such as COS/SINE, EXP, Random, SETP (set protocol type), FORS (for loop starting point), FORE (for loop ending point), Floating point divider, multiplier and format converters. (iii) Logic block, address generation, Program counter logic, PCL and other control logic blocks.

**The functional definitions of sub-blocks in architectural diagram**

- **Execution Unit (EU):** It will implement other than functional units mentioned in the architecture that is COS/SINE, EXP, FORS, FORE, COPY, FADS, etc. I/P register and O/P register may be clocked.

- **Address Generation Unit. (AGU) and Memory Units**

Seven separate memory units are proposed; out of which four of 256 X 32-bit size memories are for program memory floating-point look-up table, dynamic data and constants storage. Three memory units of size 256 X 16-bit for integers look-up table, dynamic data and constants storage. Interface to and from other I/O blocks are either doing the computations or to store the I/O and computation results. The entire memory unit's addresses and their corresponding data should be displayed in binary format only. The reason behind this condition is compatible to any kind of crosschecking and independent of the instruction-set design.

*The following units are included their IP/OP registers in the block itself.*

• **Protocol Stack Engine**:

It will run any one of chosen protocol depending on the IP of the ASIP and decided by Reset and Program start pointer. That is it will load one protocol library among the available based on its IP. Ex: CDMA-2000, GSM, and so on.

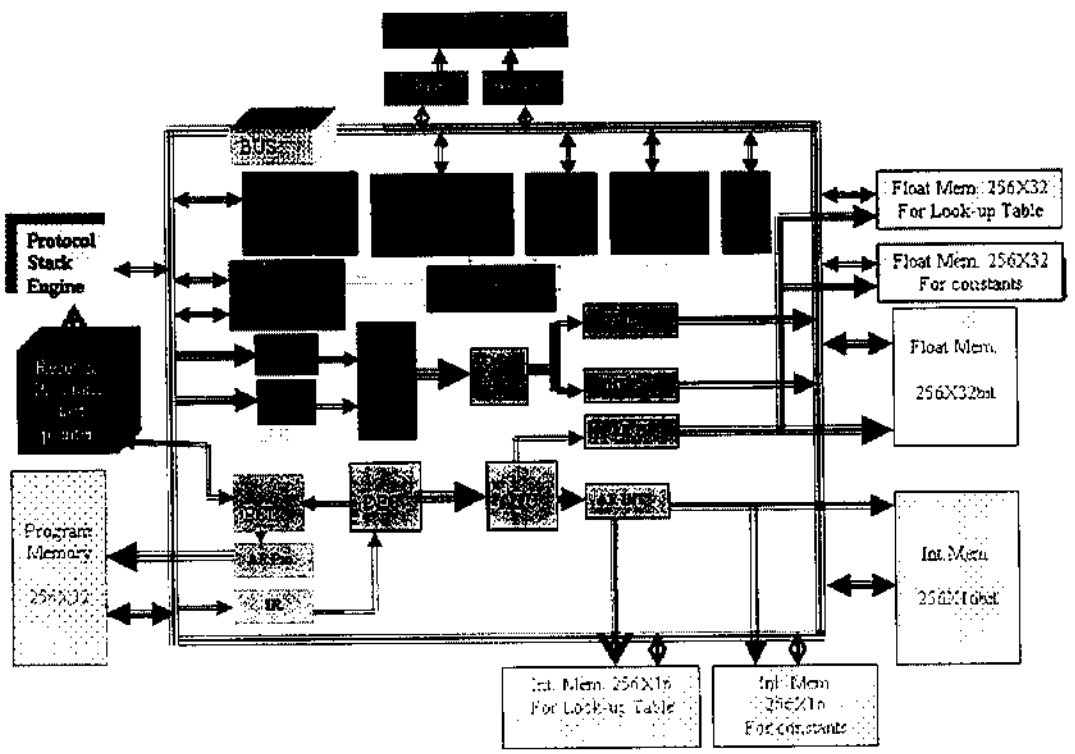



**Figure 2.1: Proposed Architecture of ASIP for ADR**

• **Channel Coding and decoding:** Depending upon the protocol chosen, any one or combination of the channel coding will be used. 1. Viterbi Encoding and decoding 2. Turbo Encoding and decoding 3. Reed-Solomon (RS) Encoding and decoding 4. Convolution Encoding and decoding and 5. Trellis-coded encoding and decoding

• **Modulation and demodulation schemes**: Ex: GMSK, 64-QAM.

# CHAPTER 3

# HARDWARE DESCRIPTION LANGUAGE

## 3.1 INTRODUCTION

Hardware Description Language (HDL) is a language that can describe the behavior and structure of electronic system, but it is particularly suited as a language to describe the structure and the behavior of the digital electronic hardware design, such as ASICs and FPGAs as well as conventional circuits. HDL can be used to describe electronic hardware at many different levels of abstraction such as Algorithm, Register transfer level (RTL) and Gate level. Algorithm is un synthesizable, RTL is the input to the synthesis, and Gate Level is the input from the synthesis. It is often reported that a large number of ASIC designs meet their specification first time, but fail to work when plunged into a system. HDL allows this issue to be addressed in two ways; a HDL specification can be executed in order to achieve a high level of confidence in its correctness before commencing design and may simulate one specification for a part in the wider system context. This depends upon how accurately the specialization handles aspects such as timing and initialization.

## 3.2 ADVANTAGES OF HDL

A design methodology that uses HDLs has several fundamental advantages over traditional Gate Level Design Methodology. The following are some of the advantages:

- One can verify functionality early in the design process and immediately simulate the design written as a HDL description. Design simulation at this high level, before implementation at the Gate Level allows testing architectural and designing decisions.
- FPGA synthesis provides logic synthesis and optimization, so one can automatically convert a VHDL description to gate level implementation in a given technology.

- HDL descriptions provide technology independent documentation of a design and its functionality. A HDL description is more easily read and understood than a net-list or schematic description.
- HDLs typically support a mixed level description where structural or net-list constructs can be mixed with behavioral or algorithmic descriptions. With this mixed level capabilities one can describe system architectures at a high level or gate level implementation.

## 3.3 VHDL

VHDL is a hardware description language. It describes the behavior of an electronic circuit or system, from which the physical circuit or system can then be attained.

VHDL stands for VHSIC Hardware Description Language. VHSIC is itself an abbreviation for Very High Speed Integrated Circuits, an initiative funded by United States Department of Defense in the 1980s that led to creation of VHDL. Its first version was VHDL 87, later upgraded to the VHDL 93. VHDL was the original and first hardware description language to be standardized by Institute of Electrical and Electronics Engineers, through the IEEE 1076 standards. An additional standard, the IEEE 1164, was later added to introduce a multi-valued logic system.

VHDL is intended for circuit synthesis as well as circuit simulation. However, though VHDL is simulating fully, not all constructs are synthesizable. The two main immediate applications of VHDL are in the field of Programmable Logic Devices and in the field of ASICs (Application Specific Integrated Circuits). Once the VHDL code has been written, it can be used either to implement the circuit in a programmable device or can be submitted to a foundry for fabrication of an ASIC chip.

VHDL is a fairly general-purpose language, and it doesn't require a simulator on which to run the code. There are many VHDL compilers, which build executable binaries. It can read and write files on the host computer, so a VHDL program can be written that generates another VHDL program to be incorporated in the design being developed. Because of this general-purpose nature, it is possible to use VHDL to write a

test bench that verifies the functionality of the design using files on the host computer to define stimuli, interacts with the user, and compares results with those expected.

The key advantage of VHDL when used for systems design is that it allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires). The VHDL statements are inherently concurrent and the statements placed in a PROCESS, FUNCTION or PROCEDURES are executed sequentially. The different approaches in VHDL are structural, data flow, and behavioral methods of hardware description.

### 3.3.1 STRUCTURAL DESCRIPTIONS

The structural descriptions are explained below with examples. Every portion of a VHDL design is considered a block. A VHDL design may be completely described in a single block, or it may be decomposed in several blocks. Each block in VHDL is analogous to an off-the-shelf part and is called an entity. The entity describes the interface to that block and a separate part associated with the entity describes how that block operates. The interface description is like a pin description in a data book, specifying the inputs and outputs to the block. The following is an example of an entity declaration in VHDL.

```
entity latch is
    port (s,r: in bit;
      q,nq: out bit);
end latch;
```

The first line indicates a definition of a new entity, whose name is latch. The last line marks the end of the definition. The lines in between, called the port clause, describe the interface to the design. The port clause contains a list of interface declarations. Each interface declaration defines one or more signals that are inputs or outputs to the design. Each interface declaration contains a list of names, a mode, and a type. The following is an example of an architecture declaration for the latch entity.

```
Architecture dataflow of latch is
    signal q0 : bit := '0';
```

```
    signal nq0 : bit := '1';
    begin
      q0<=r nor nq0;
      nq0<=s nor q0;
      nq<=nq0;
     q<=q0;
end dataflow;
```

The first line of the declaration indicates that this is the definition of a new architecture called dataflow and it belongs to the entity named latch. So this architecture describes the operation of the latch entity. The schematic for the latch might be

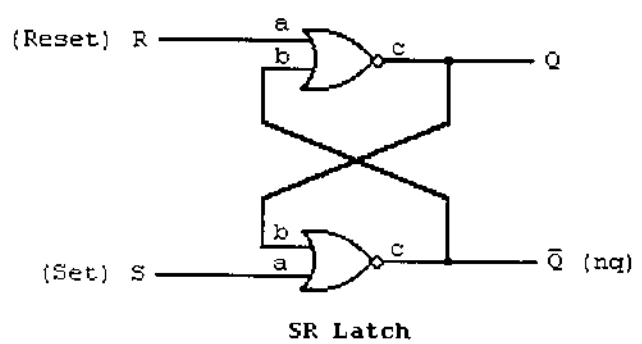



**Figure 3.1: Schematic SR Latch**

The same connections that occur in the schematic using VHDL with the following architecture declaration:

```
Architecture structure of latch is
     component nor_gate
      port (a,b: in bit;  c: out bit);
    end component;
    begin
      n1: nor_gate
      port map (r,nq,q);  n2: nor_gate
      port map (s,q,nq);
    end structure;
```

The lines between the first and the keyword begin are a component declaration. A list of components and there connections in any language is sometimes called a net list. The structural description of a design in VHDL is one of many means of specifying net lists.

### 3.3.2 Data Flow Descriptions

In the data flow approach, circuits are described by indicating how the inputs and outputs of built-in primitive components are connected together. To describe the following SR latch using VHDL as in the following schematic.

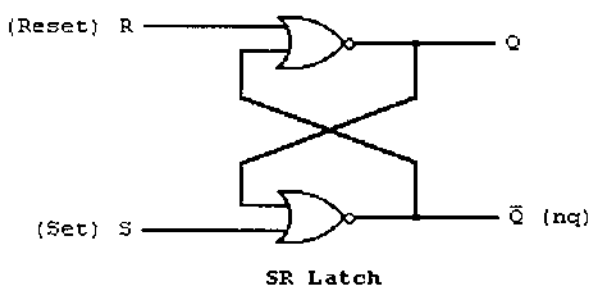



**Figure 3.2: Data flow approach in SR Latch**

The entity part is written in the following manner.

```
entity latch is
   port (s,r : in bit;
   q,nq : out bit);
end latch;
architecture dataflow of latch is
   begin
   q<=r nor nq;
   nq<=s nor q;
end dataflow;
```

The signal assignment operator in VHDL specifies a relationship between signals, not a transfer of data as in programming languages. The architecture part describes the internal operation of the design. The scheme used to model a VHDL design is called discrete event time simulation. In this the values of signals are only updated when certain

events occur and events occur at discrete instances of time. The above mentioned SR latch works with this type of simulation.

There are two models of delay are used in VHDL. The first is called the inertial delay model. The inertial delay model is specified by adding an after clause to the signal assignment statement. The second is called transport delay model.

### 3.3.3 Behavioral Descriptions

The behavioral approach to modeling hardware components is different from the other two methods in that it does not necessarily in any way reflect how the design is implemented. It is basically the black box approach to modeling. It accurately models what happens on the inputs and outputs of the black box, but what is inside the box is irrelevant. The behavioral description is usually used in two ways in VHDL. First, it can be used to model complex components.

Behavioral descriptions are supported with the process statement. The process statement can appear in the body of an architecture declaration just as the signal assignment statement does. The process statement can also contain signal assignments in order to specify the outputs of the process. A variable is used to hold data and also it behaves like you would expect in a software programming language, which is much different than the behavior of a signal. Although variables represent data like the signal, they do not have or cause events and are modified differently. Variables are modified with the variable assignment

There are several statements that may only be used in the body of a process. These statements are called sequential statements because they are executed sequentially. The types of statements used here are if, if else, for and loop. A signal assignment, if anything, merely schedules an event to occur on a signal and does not have an immediate effect. When a process is resumed, it executes from top to bottom and no events are processed until after the process is complete.

In most programming languages there is a mechanism for printing text on the monitor and getting input from the user through the keyboard. It can able to give output certain information during simulation. Every VHDL language system will contain a standard library. In VHDL, common code can be put in a separate file to be used by

many designs. This common code is called a library. The write statement can be used to append constant values and the value of variables and signals of the types bit, bit_vector, time, integer, and real.

### 3.4 SOFTWARES USED

- ModelSim PE 6.1e
- Xilinx ISE 9.2i

## 3.5 EDA Tools

There are several EDA (Electronic Design Automation) tool available for circuit synthesis, implementation and simulation using VHDL. Some tools are offered as part of a vendor's design suite such as Altera's Quatus II which allows the synthesis of VHDL code onto Altera's CPLD/FPGA chips, or Xilinx's ISE suite, for Xilinx's CPLD/FPGA chips. The tools used were either ISE combined with ModelSim.

# CHAPTER 4

# DESIGN OF RASIP

## 4.1 INSTRUCTION SET DESIGN

The general architecture can be viewed as a different-tire algorithms implementation to achieve the complete functionality. Hence, we are providing two-tier level of functionality implementation of SDR algorithm modules. The novelty of our instruction-set is that the user can define the own instructions with required functionality using format FMT7. The other interesting achievement is to embed all the controlling and decoding part in the microinstruction architecture to reduce the fetching and decoding sequentially.

The internal registers for each functional unit, which implements or executes instructions in parallel, are provided. Thus, the VLIW architecture philosophy with CISC instruction making code generation scheme much simple are achieved. The instruction-set format use 32-bit and implements all the required functionality of the SDR environment. The different instructions are required to implement complete functionality of the SDR these are classified as follows:

- General computation instructions used for the general computation and logical functions.
- General Instructions for the communication system Design.
- Application specific instructions to achieve SDR functionality implemented in RLU.
- Stream-based (block based) instructions, which are reconfigurable by the user implemented in RLU.

## 4.2 INSTRUCTION-SET FORMATS

To implement the above type of instructions, 7-types of different formats are proposed with simple decoding. Based on previous experience and requirements of the SDR implementation we have decided to have the following different instruction formats, which are shown in Figure 4.1 for designing the instruction-set. The decoding logic of instruction-format is shown in table 4.1. Here for simplicity we have given only op-code decoding and rest of the fields are self-explained.

Reconfigurable instructions are identified by a special op-code "111111". The next 5-bits field will be a function (25 = 32 functions can be accommodated, but we are not using all those, in future we can provide them with the remaining slots so that more flexibility can be obtained) or name of the instruction this will point the location of the configuration data for the instruction is specified in the instruction word similar to DIS. The FMT6 is application specific instructions for the SDR functionality where as FMT7 supports user definitions to simplify his logic in addition to application specific instructions for SDR

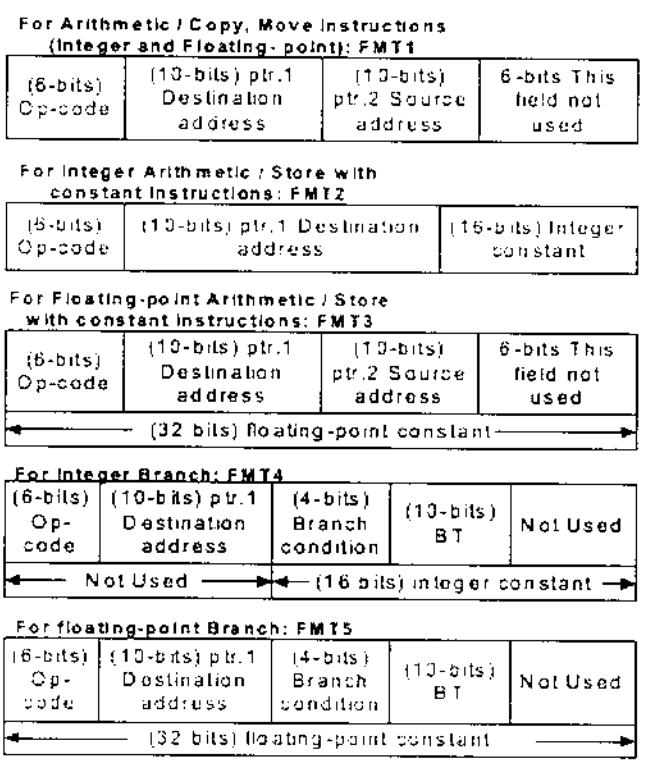



**Figure 4.1: Instruction Set format of general instruction**

The total instructions set are classified as follows:

*Data move and program control:* Total instructions in this category are 13 and are shown in table 4.1

*Arithmetic:* Total instructions in this category are 14 and are shown in table 4.1

| Code | Cat. code | Mem code | Instruction | Group code | Cat. Code | Mem. code | Instruction |
|---|---|---|---|---|---|---|---|
| 00 Data move and program control instructions | 0 | 000 | COPY I,I | 01 Arithmetic instruction Groups | 0 | 000 | ADD I,I |
| | 0 | 001 | COPY I,C | | 0 | 001 | ADD I,C |
| | 0 | 010 | NEG I | | 0 | 010 | SUB I,I |
| | 0 | 011 | BRANCH I,I | | 0 | 011 | SUB I,C |
| | 0 | 100 | BRANCH I,C | | 0 | 100 | MULT I, I |
| | 0 | 101 | COMPR I,I | | 0 | 101 | MULT I,C |
| | 1 | 000 | CALL ( ) | | 1 | 000 | ADD F,F |
| | 1 | 001 | RETURN ( ) | | 1 | 001 | ADD F,C |
| | 1 | 010 | COPY F,F | | 1 | 010 | SUB F,F |
| | 1 | 011 | COPY F,C | | 1 | 011 | SUB F,C |
| | 1 | 100 | NEG F | | 1 | 100 | MULT F,F |
| | 1 | 101 | Not defined | | 1 | 101 | MULT F,C |
| | 1 | 110 | BRANCH I,C | | 1 | 110 | DIVD F,F |
| | 1 | 111 | COMPR I,I | | 1 | 111 | DIVD F,C |
| | 0 | 000 | COPY I,I | | | | |
| 10 Application specific but general to communication system design instructions | 0 | 000 | RND I,F | 11 Application specific instructions for base band and channel coding | 0000 | | CH_CODE() |
| | 0 | 001 | TRUNC I,F | | 0001 | | CH_DECOD( ) |
| | 0 | 010 | FOR_S( ) | | 0010 | | DDC( ) |
| | 0 | 011 | FOR_E( ) | | 0011 | | DUC( ) |
| | 0 | 100 | CONVT I,F | | 0100 | | NCO( ) |
| | 0 | 101 | SETP( ) | | 0101 | | FIR( ) |
| | 1 | 000 | Fn_EXP F,F | | 0110 | | IIR( ) |
| | 1 | 001 | Fn_COS F,F | | 0111 | | CIC( ) |
| | 1 | 010 | Fn_SINE F,F | | 1000 | | FFT( ) |
| | 1 | 011 | MAC F,F | | 1001 | | MOD( ) |

**Table 4.1: Encoding of op-code field in the first, second and third group instructions**

*Application Specific but general to communication system modeling:* Total instructions in this category are 14 and are shown in table 4.1

*Application Specific to base-band and channel coding:* Total instructions in this category are 11 and are shown in table 4.1

## 4.3 DESIGN OF DDC

In many applications the signal of interest may not be at the optimum part of the spectrum for processing. In a communications system the signal band may only be narrow, KHz wide, but the signal band could be centered at RF frequencies, at many MHz, if the signal is sampled according to the Nyquist criteria, twice the highest frequency. The data rate for the RF signal will be very high and processing the data at this high rate is both difficult and expensive in terms of the amount of hardware required. A DDC will select the frequency band of interest and shift it down in frequency, which allows the data rate required to retain all the information to be much lower, and consequently reduces the complexity of any further processing.

A DDC can be split into two main sections, first a digital mixer to frequency shift the spectrum of the signal, and then filters to remove unwanted spectral components and allow reduction, or decimation, of the data rate. The processing in the example DDC is based on quadrature sampling, the amount of hardware is doubled, as there is now both Phase and Quadrature channels, but the data rate required to retain all the information is halved. Quadrature sampling has the advantage that the negative frequency components cancel. The Phase and Quadrature outputs of the DDC could be linked to other CoreGen blocks such as a complex FFT to analyze the spectrum, or a CORDIC block to extract the magnitude and phase from the complex signal.

The object of the signal generator is to generate a composite signal centered on 10MHz. There are two banks of RAM inside the FPGA that can be loaded over the HERON interface, with values describing the required signal waveforms. The two blocks of RAM are loaded in the demonstration with an Amplitude Modulated (AM) waveform and a Frequency Modulated (FM) waveform respectively. These RAMs are read at a rate of 1.5625 MHz with the address incrementing after each read, when the maximum address value is reached the next address starts again at zero.

The adder combines the waveforms stored in the RAMs to give a composite signal with both AM and FM. The block diagram in figure 4.2 shows the basic structure of the DDC. The analogue input signal to the ADC is the AM and FM signal modulated up to 10MHz in the signal generator section of this demonstration. The ADC samples the 10MHz analogue signal at 210MSPS. The object of the DDC is to shift the frequency band of interest, which is at 10MHz, down to baseband. The first step is to mix the digitized input signal with 10MHz to produce sum and difference frequency components at 20MHz and baseband respectively. In this example the input signal is mixed with both a 10MHz sine and cosine to produce a quadrature and in phase signal respectively.

The Direct Digital Synthesizers (DDS) generates a 10MHz digitized sine and cosine waves to mix, or multiply, the digitized input signal. The same sample frequency for the ADC and for the output of the DDS has to be the same and is 210MSPS in this example.

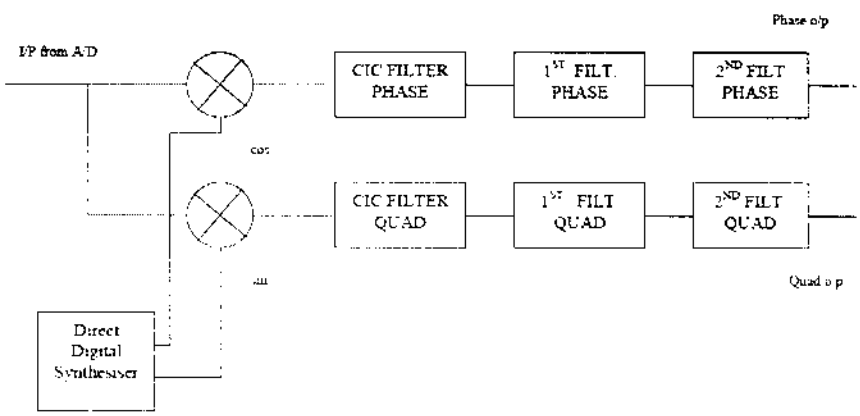



**Figure 4.2: Digital down Converter Block diagram**

The low pass filters that follow the mixers are there to remove the sum frequency component and also to ensure that the bandwidth of the signal is suitable for the new sample frequencies after decimation. In this example the sample frequency has been kept to the Nyquist rate or better as the signal progresses through the filter stages, even though the sample rate could be halved because of the in phase and quadrature sampling. This is because only the in phase component of the DDC output signal is transferred to the host program to be displayed and FFT.

The DDC needs the data to be multiplexed onto a single bus running at the full sample clock frequency. A Digital Clock Manager (DCM) is used in the FPGA to double the data clock frequency while retaining the timing relationship to the data busses. The ADC sample clock should not be used to clock the ADC data. This new data clock is now at the sample frequency, and has the correct timing relationship to the data busses to allow the sample data to be multiplexed onto a single bus. This new clock should be used for clocking all the data that requires the full sample rate frequency.

The DDS generates the digitized 10MHz sine and cosine signals required to mix with the ADC data giving both sum and difference frequencies. The cosine signal is used to generate the in Phase component, and the sine generates the quadrature component of the complex signal. The DDS has to be clocked at the sample frequency so that sine and cosine data are presented to the mixers at the same rate as the multiplexed ADC data. The digitized 10MHz sine and cosine waves have a sample frequency of 210MHz so there are 21 samples per cycle.

CIC filters are low pass filters used to realize sample rate changes, especially in systems that have large excess sample rates. At the outputs of the DDC mixers the difference frequency band of interest is only in the order of 100 kHz while the sample frequency is still 210MHz. CIC filters also have the advantage that they do not require multipliers, consisting only of adders, subtractors and registers. The sample rate has been reduced from 210MSPS a lower clock frequency can be used which will relax the required timing constraints. The change in clock frequency is achieved using a fifo. At the input to the fifo data is clocked in using the 210MHz clock at 6.5625MSPS and at the output the data is clocked out using a 100MHz clock at 6.5625MSPS.

The filters are standard Finite Impulse Response (FIR) filters both with 35 taps that are used to filter and to further decimate the sample frequency. The low pass frequency response of FILT1 has a cut off frequency at (0.048x6.5625) = 315 kHz and the sample frequency at the output is decimated by 8 giving (6.5625/8) = 820kSPS. The sample frequency required to retain all the information in a purely real signal, no phase and quadrature components, was defined by Nyquist to be twice the signal bandwidth.

In a DDC the sample frequencies can be much higher than required for the bandwidth of the signal of interest, but before the sample frequency is decimated

unwanted frequency components must be removed by filtering. Decimating the sample frequency makes signal processing simpler to implement as the timing constraints can be relaxed, it can also lead to a reduction in the amount of FPGA area required to perform the processing as functions can be performed serially rather than in parallel.

At the input to the filter or decimator there could well be signals outside the frequency band of interest, but because of the higher sample frequency do not interfere with the required signals. If the sample frequency was just decimated without filtering the unwanted signals would be overlaid onto the required signal frequency band. This is why the signal is filtered first, to remove any frequency components that could be overlaid onto the required signal, and then the sample frequency is decimated. In this example the Host software takes just the phase component for an FFT, so the sample frequency at the output of the filter/decimators has been kept to the Nyquist rate. If the full complex signal was used the sample frequency could be halved.

## 4.4 DESIGN OF CIC FILTER

A very efficient architecture for a high decimation-rate filter is the "cascade integrator comb" filter. The Cascade Integrator Comb (CIC) has proven to be an effective element in high-decimation or interpolation systems. One application is in wireless communications, where signals, sampled at RF or IF rates, need to be reduced to baseband. For narrow band applications, decimation rates excess of 1000 are routinely required. Such a system is sometimes referred to as channelizers.

CIC filters are based on the fact that perfect pole/zero canceling can be achieved. This is only possible with exact integer arithmetic. Both two's complement and the residue number system have the ability to support error free arithmetic. In the case of two's complement, arithmetic is performed modulo $2^b$ and in the case of modulo M.

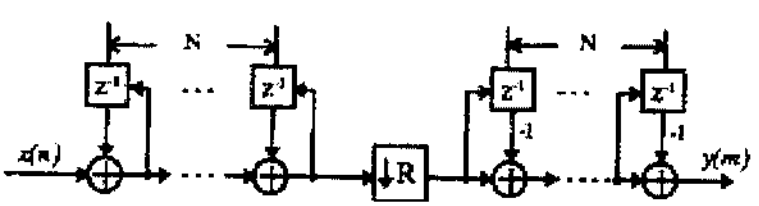



**Figure 4.3: CIC Filter Block Diagram**

The first-order CIC filter without decimation is shown in Figure 4.3. The filter consists of a integrator followed by a differentiator or comb. The filter is recursive, the impulse response is finite. The impulse response shows that the filter computes the sum.

$$y[n] = \left[\frac{1 - Z^{-R}}{1 - Z^{-1}}\right]^n$$

Hogenauer's CIC decimation filter shown in Fig. 4.3, consists of N cascaded digital integrators operating at a high input sampling rate, $f_s$, and N cascaded differentiators at a low rate, $f_s$/R,where R is the integer down conversion factor, Its transfer functions which is an N-stage sinc function. Due to the droop around the cutoff frequency and aliasing within the band of interest, the choices of N and *R* are made to provide acceptable pass band characteristics over the range from zero to the cutoff frequency. Generally speaking, the higher the first down conversion rate, $f_s$ /R, relative to the Nyquist rate, $f_s$, the smaller the droop and the higher aliasing attenuation. However, this leaves more filtering to be done with a less efficient transversal or IIR decimator. Hogenauer gave two tables for the design trade-off, For nth-order low pass modulators, the compromise is to take the first down conversion rate to be four times the Nyquist rate. In doing so, one has to compensate only small droop later in the low frequency stage.

## 4.5 DESIGN OF DUC

The Digital up Converter is a digital circuit which implements the conversion of a complex digital base band signal to a real pass band signal. The input complex base band signal is sampled at a relatively low sampling rate, typically the digital modulation symbol rate. The base band signal is filtered and converted to a higher sampling rate before being modulated onto a direct digitally synthesized (DDS) carrier frequency. The DUC typically performs pulse shaping and modulation of an intermediate carrier frequency appropriate for driving a final analog up converter and is used extensively in wireless and wire line communication systems.

A DUC consists of a series of cascaded interpolation finite impulse response (FIR) filters, a mixer, and a direct digital synthesizer (DDS). The DUC is a cascade of two FIR, Interpolation Filters and one Cascaded Integrator Comb (CIC) Interpolation Filter. The first FIR Interpolation Filter is a pulse shaping FIR filter that increases the

sampling rate by 2 and performs transmitter Nyquist pulse shaping. The second FIR Interpolation Filter is a compensation FIR filter that increases the sampling rate by 2 and compensates for the distortion of the following CIC filter. The CIC Interpolation Filter is a programmable filter increases the sampling rate by 4 to 1448. Mixer is designed with an area efficient high-speed algorithm for variable multiplication and generation of carrier waves with a wide range of frequencies from the Direct Digital Synthesizer.

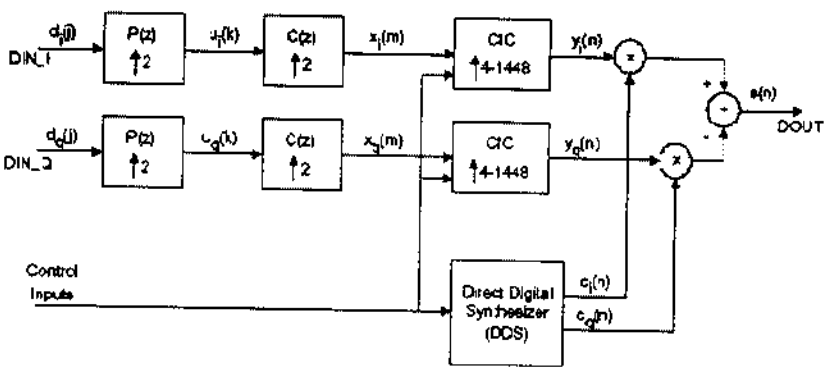



**Figure 4.4: Digital up Converter Block Diagram**

A block diagram of the Digital up Converter is shown in Figure 4.4. Spectral shaping of the complex input signal is performed by the PFIR filter. Typically this filter would be performing a Nyquist transmit filter operation with a rate-change of 2. Bias-free convergent rounding or truncation is employed between each processing stage to limit the bit growth through the DUC. Output from each of the PFIR filters is input to each of the CFIR filters, which is used to compensate for the droop within the CIC filter and performs the second rate-change. The CFIR also performs a rate-change of 2.

The complex data stream from the CIC filter is mixed with a local oscillator generated by the DDS. Results from the mixers are combined, forming the final DUC result. The DUC result is often used as the input to a digital-to-analog converter (DAC) to generate an intermediate frequency analog signal. When the programmable CIC rate change option is selected, and the current rate change is less than the maximum rate change, then the output from the DUC will have times when VOUT is not valid and the last valid output data will be maintained on the DOUT port.

The DUC consists of following blocks Pulse shaping FIR filter, Compensation FIR filter, Cascaded Integrator Comb (CIC) filter, Direct Digital Synthesizer (DDS), Multiplier. The Pulse Shaping FIR filter P (z) provides a sampling rate increase of 2 as shown in figure 4.5, and typically performs transmitter Nyquist pulse shaping. The programmable finite impulse response (PFIR) filter usually performs spectral shaping.

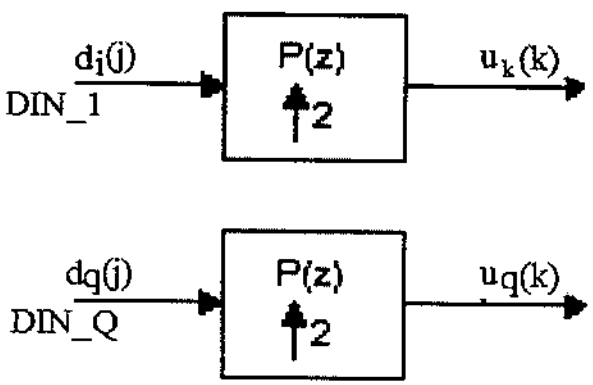



**Figure 4.5: Block Diagram of Pulse Shaping Block in DUC**

FIR filters P(z) operate on the sequences $d_i(j)$ and $d_q(j)$ applied to the DUC input ports DIN_I and DIN_Q. The sequences $d_i(j)$ and $d_q(j)$ are up sampled by a factor of 2 by filtering with the Pulse Shaping Filter coefficients p(k), with the resulting sequences u(k) sampled at a rate of $F_{sk} = 2F_{s\_in}$. The pulse shaping finite impulse response filter P(z) typically conditions the transmitter signal's channel response and inter-symbol interference characteristics. Pulse shaping techniques are intended to decrease the profile of the transmitted signal, without compromising its information bearing properties, so that all the system creates less overall interference with one another.

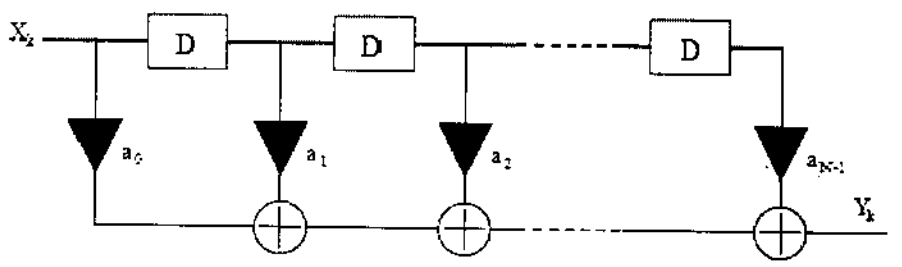
## 4.6 DESIGN OF DIGITAL FILTERS

Digital filters can be categorized into two classes known as FIR (finite-length impulse response) and IIR (infinite-length impulse response filters). Advantages of FIR filters over IIR filters are that they are guaranteed to be stable and to have a linear-phase response. Linear-phase FIR filters are widely used in digital communication systems, in speech and image processing systems, in spectral analysis, and particularly in

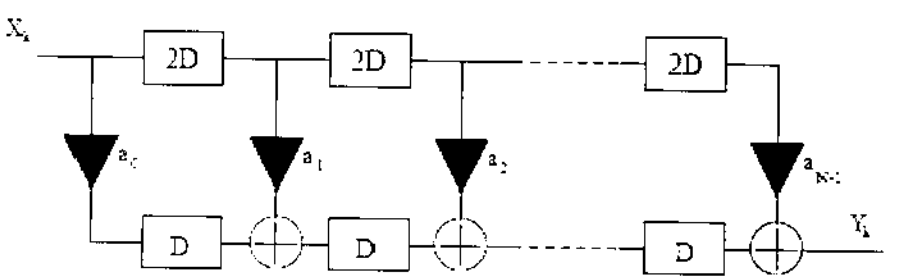
applications where nonlinear-phase distortion cannot tolerate. The transfer function of an N tap FIR filter is given by

$$H(Z) = a_0 + a_1 Z^{-1} + \cdots + a_{N-1} Z^{-(N-1)}$$

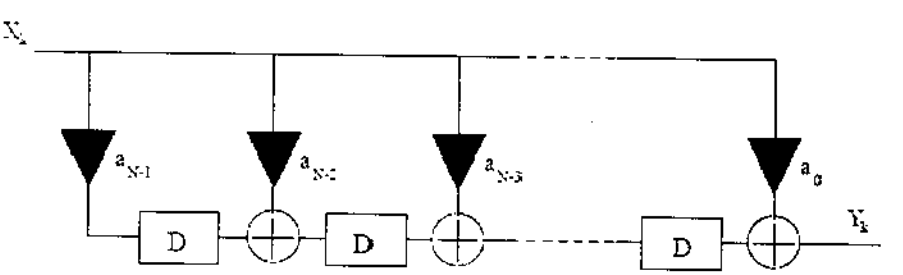
This structure can be realized in many ways, such as the canonical form, pipelined form, and inverted form as depicted in Figure 4.6.




(a) Canonic Form




(b) Pipelined Form




(c) Inverted Form

**Figure 4.6: Various Realizations of FIR Filters**

The inverted form shown in Figure 4.6(c) is well-suited for achieving a high sampling rate even for higher order filters. This is possible because the throughput does not depend strongly on the number of taps due to extensive pipelining. The fact that the multipliers occupy a large area, however, might render the implementation of higher order filters impractical. It has been shown in that a high performance FIR filter with substantial number of taps can be implemented on FPGAs by approximating the filter coefficients to a sum or difference of two powers-of- two terms. Implementation of digital filters may be simplified by using only a limited number of power-of-two terms so that only a small number of shift and add operations is required. A variety of techniques have been proposed to minimize the deterioration of the frequency response due to these constraints. Such coefficient optimization techniques yield performance sufficient for most practical applications.

When the size of the chip is a constraint, the arithmetic resources need to be shared at the expense of speed. The structure shown in Figure 4.7 is suitable for sharing of arithmetic resources. This is a multiply/accumulate (MAC) unit with four multipliers and an adder tree. The inputs and the corresponding filter coefficients are fed to the MAC unit as shown in Figure 4.7. With the insertion of pipeline registers, the clock speed is increased. The delay in the multiplier is greater than that in the adder and hence the clock frequency is dependent on the delay in the multiplier. As there are four multipliers in this MAC unit, summation of four terms is computed every clock cycle.

Hence a four tap filter can be made to operate at a sampling rate equal to the clock rate, and an eight tap filter to operate at a sampling rate half that of the clock rate. An implementation based on the multiple-input MAC unit, as shown in Figure 4.7, was used to evaluate this moderate performance approach to the realization of a filter with an arbitrary number of taps. The four multipliers are arranged in the four corners of the 20 by 20 array of CLBs to reduce the delay from the input pins to the multipliers. Inputs to the multipliers are fed in at right angles, as explained previously, and the arrays are oriented in such a way that the routing delays from pads are minimized.

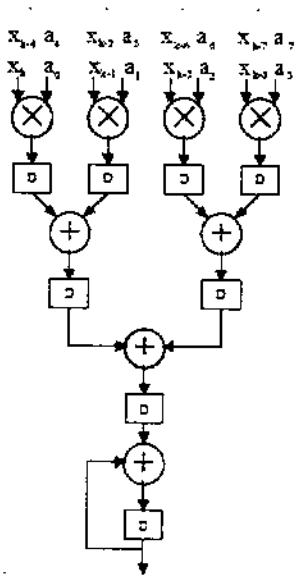


**Figure 4.7: FIR Filter Realization Using MAC**

The transfer function of an $N^{th}$ order IIR filter is given by

$$H(Z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_{N-1} z^{N-1}}{1 - a_1 z^{-1} - a_2 z^{-2} \ldots + a_{N-1} z^{N-1}}$$

Some of the realizations possible are direct form I and direct form II. For reducing the delay in the paths between registers, however, the realization shown in Figure 9 is used in this paper. This realization, like direct form II, is a cascade of an autoregressive (AR) filter and a moving average (MA) filter, but with a pipeline register in between. The delay elements are also rearranged in such a way that there is only one path with a multiplier and two adders. The others have only one multiplier and one or no adder. This realization allows easier placement of the multipliers and adders in the array of CLBs to achieve minimal routing delays.

Second order IIR filters with general purpose multipliers, which can take coefficients as inputs from outside the chip, can be used as building blocks for cascade or parallel realizations of higher order IIR filters. The first term in the denominator of the transfer function may be scaled according to the number of bits in the coefficients for fixed point implementation. This implies that a scaling module is needed before the pipeline register between the AR and MA sections shown in Figure 4.8. This divider can be implemented with a shifter without considerably increasing the area and delay by

constraining this coefficient to be equal to the nearest allowable power-of-two number during the discrete space optimization of the quantized coefficients. In this implementation, the multipliers have 8 bit inputs and the adders have 16 bit inputs.

The 16 bit output of the multipliers is fed to the adders and the most significant 8 bits of the output of the adders are fed back to the multipliers. The two adders in the autoregressive part of the filter referred to in Figure 4.8 are implemented as a cascade of two dedicated carry logic adders. This adder has a total combinatorial delay of 23 ns (2.5 + 20.5), which is close to that of a single dedicated carry logic adder. Thus the dominant delay is not necessarily in the logic path with two adders as it may seem, but could very well be in other paths with one multiplier and adder, unless placement is carefully considered.

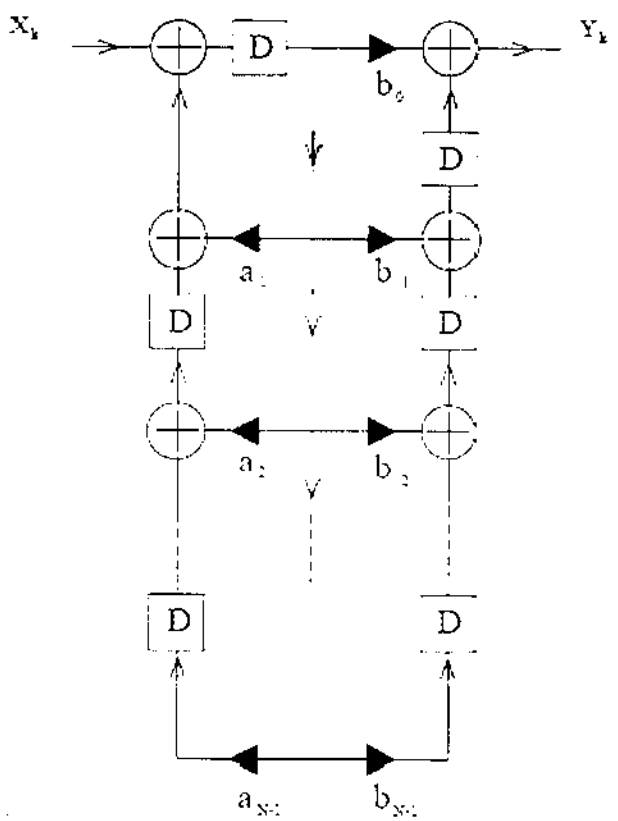



**Figure 4.8: Modified Canonical Form Realization of IIR Filters**

# CHAPTER 5

# VHDL DESIGN AND IMPLMENTATION

## 5.1 COMMUNICATION SYSTEM DESIGN INSTRUCTIONS

The Sine and Cosine values can be calculated using the first CORDIC scheme which calculates:

$$[X_j, Y_j, Z_j] = [P(X_i \cos(Z_i) - Y_i \sin(Z_i)), P(Y_i \cos(Z_i) + X_i \sin(Z_i)), o]$$

By using the following values as inputs

$$X_i = \frac{1}{P} = \frac{1}{1.6467} = 0.60725$$

$$Y_i = 0$$

$$Z_i = 0$$

The core calculates:

$$[X_j, Y_i, Z_i] = [\cos\theta, \sin\theta, 0]$$

The input Z takes values from –180degrees to +180 degrees where:

0x8000 = - 180degrees

0xEFFF = +80degrees

But the core only converges in the range –90degrees to +90degrees.The other inputs and the outputs are all in the range of –1 to +1. The congregate constant P represented in this format results in:

$$Xi = 215 \cdot P = 19898\ (dec) = 4DBA\ (hex)$$

To calculate sine and cosine of 30 degrees, first the angle has to be calculated:

$$360\ \text{deg} \equiv 2^{16}$$

$$1\ \text{deg} \equiv \frac{2^{16}}{360}$$

$$30\ \text{deg} \equiv \frac{2^{16}}{360} * 30 = 5462\ (\text{dec}) = 1555(\text{hex})$$

The core calculates the following sine and cosine values for Zi = 5461;

Sin : 16380 (dec) = 3FFC (hex)

Cos : 28384(dec) = 6EDD (hex)

The output represents values in the -1 to +1 range. The results can be derived as follows

$2^{15} \equiv 1.0$

$16380 \equiv \frac{1.0}{2^{15}} * 16380 = 0.4999$

The values for the various angles are shown the table 5.1.

| | 0 deg | 30 deg | 45deg | 60 deg | 90 deg |
|---|---|---|---|---|---|
| Sin | 0x01CC | 0x3FFC | Ox5A82 | 0x6EDC | 0x8000 |
| Cos | 0x8000 | 0x6EDD | 0x5A83 | 0x4000 | 0x01CC |
| Sin | 0.01403 | 0.49998 | 0.70709 | 0.86609 | 1.00000 |
| Cos | 1.00000 | 0.86612 | 0.70712 | 0.50000 | 0.01403 |

**Table 5.1: Sin/Cos outputs for common angles**

The exponential function $e^x$ can be characterized in a variety of equivalent ways. In particular it may be defined by the following power series.

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots,$$

In computing, especially digital signal processing, multiply-accumulate is a common operation that computes the product of two numbers and adds that product to an accumulator, A = A + (B*C).

Multiplication followed by accumulation is a common operation in many digital systems particularly those highly interconnected, like digital filters, neural networks, data quantizer, etc., The MAC consists of multiplying two values, then adding the result to the previously accumulate value, which must then be re-stored in the registers for future

accumulations. Another feature of a MAC circuit is that it must check for overflow, which might happen when the number of MAC operation is large.

## 5.2 NUMERICAL CONTROLLED OSCILLATOR (NCO)

A numerical controlled oscillator is the digital counterpart of an analog voltage controlled oscillator. Based on the error voltage, the output frequency is changed in an analog VCO. Similarly, in a digital NCO, based on the error input word the output frequency is altered. An NCO consists of an accumulator, to which an incoming error signal is added. This error signal decides the output frequency of the NCO. If $f_c$ is the clock frequency,$\phi$, the incoming error signal magnitude, n, the number of bits of the accumulator, the free running output frequency of the NCO is given as:

$$f_{out} = f_c\phi/2^n$$

The NCO generates a square wave whose frequency is controlled by the error. The NCO forms an important part of digital receiver as it is used in the timing recovery. As described above, it is used to change the frequency of the clock, thereby, the clock timing.

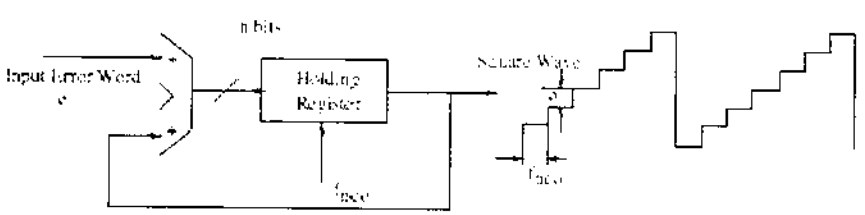



**Figure 5.1: Numerical Controlled Oscillator**

A code NCO is used in the code tracking loop. It is used to generate a clock signal, as well as the prompt signal, based upon the Early minus Late signal obtained by correlating the early and late codes. The carrier NCO is used in the carrier tracking loop, to track the phase of the incoming carrier, by changing the phase of the local signal.

The carrier tracking is done using a Costas loop. Based on the carrier discriminator, the error signal is generated. This error signal is used to increase or decrease the phase of the locally generated signal, so that it matches with the incoming signal.

```
Algorithm NCO
    while sum < threshold do
        sum = sum + ϕ
        if sum >  threshold then
            change input error word ϕ^
        else
            sum =  sum + ϕ^
        end if
    end while
```

# CHAPTER 6

# DESIGN OF FFT

## 6.1 IMPLEMENTATION OF 8-POINT FFT

The FFT computation is accomplished in three stages. The x(0) until x(7) variables are denoted as the input values for FFT computation and X(0) until X(7) are denoted as the outputs. The pipeline architecture of the 8 point FFT is shown in the Figure 6.1 consisting of butterfly schemes in it. There are two operations to complete the computation in each stage.

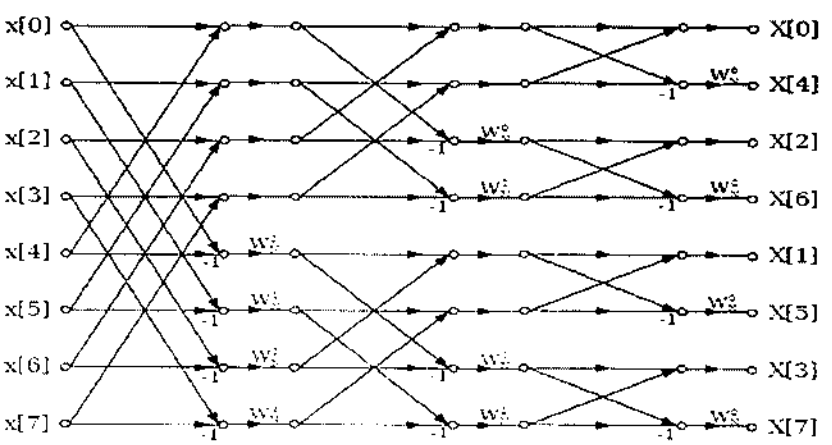



**Figure 6.1: Pipeline Architecture of 8 point FFT.**

The upward arrow will execute addition operation while downward arrow will execute subtraction operation. The subtracted value is multiplied with twiddle factor value before being processed into the next stage. This operation is done concurrently and is known as butterfly process.

The implementation of FFT flow graph in the VHDL requires three stages, final computation is done and the result is sent to the variable Y (0) to Y (7). Equation in each stage is used to construct scheduling diagram. The figure 6.2 shows the scheduling diagram of the first stage of IFFT algorithm.

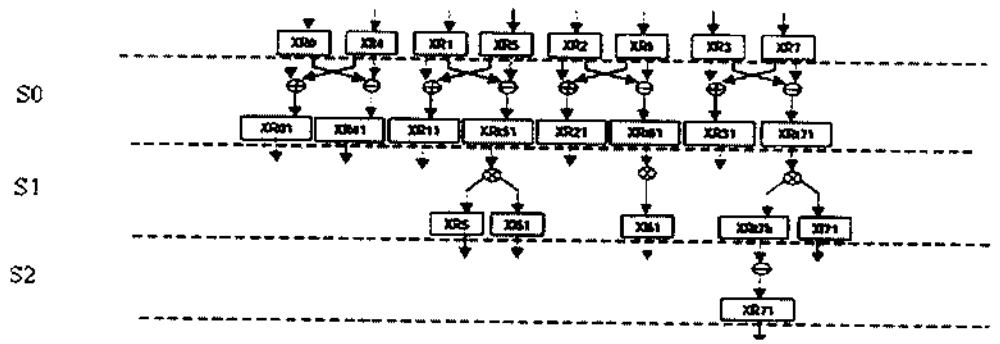



**Figure 6.2: Scheduling Diagram of stage one - FFT**

For stage one, computation is accomplished in three clock cycles denoted as S0 to S2.The operation is much simpler compared with FFT. This is because FFT processed both real and imaginary value. The result from FFT is represented in real and imaginary value because of the multiplication of twiddle factor. Twiddle factor is a constant defined by the number of point used in this transform. This scheduling diagram is derived from the equations obtain in FFT signal flow graph. The rest of the scheduling diagrams can be sketched in the same way as shown in figure 6.2. Thus each stage requires a clock cycle and totally three clock cycles are needed. Scheduling diagrams are a part of behavioral modeling and Synthesis steps to translate the algorithmic description into RTL (register transfer level) in VHDL design.

## 6.2 DESIGN OF GENERAL RADIX-2 FFT USING VHDL

As the lower to higher-point FFTs, the structure for computing the FFT becomes more complex and the need for an efficient complex multiplier to be incorporated within the butterfly structure arises. Hence the algorithm for an efficient complex multiplier that overcomes the complication of using complex numbers throughout the process.

A radix-2 FFT can be efficiently implemented using a butterfly processor which includes, besides the butterfly itself, an additional complex multiplier for the twiddle factors. A radix-2 butterfly processor consists of a complex adder, a complex subtraction, and a complex multiplier for the twiddle factors. The complex multiplication with the

twiddle factor is often implemented with four real multiplications and 2 add / subtract operations.

**Normal Complex Operation:**

$$(X+jY)\,(C+jS) = CX + jSX + jCY - YS$$

$$= CX - YS + j\,(SX + CY)$$

Real Part $R = CX - YS$

Imaginary Part $I = SX + CY$

Using the twiddle factor multiplier that has been developed, it is possible to design a butterfly processor for a radix-2 Cooley-Tukey FFT. Hence this basic structure of radix-2 FFT can be used as a building block to construct higher N-point FFTs. This structure has been developed as an extension to provide for the computation of higher value index FFTs.

The implementations have been carried out using the software, ModSim. The hardware language used is the Very High Speed Integrated Circuit Hardware Description Language (VHDL). VHDL is a widely used language for register transfer level description of hardware. It is used for design entry, compile and simulation of digital systems.

## 6.3 FFT IMPLEMENTATION STATES

The architectural design consist of data inputs, control unit, register, twiddle factor and the data output. The register may be of the array of four or eight variable in the type of real. The FFT implementation in VHDL consists of three states such as start, load and run.

The initial state is start where the eight input of the FFT are given through single clock cycles. The serial input of the input variable is declared as real. The input data is serially sent to the register memory and each input is given through a clock cycle and is used as an array in further stages through the internal register. Hence it required to give

the eight variables for defining the inputs which have to be given serially. The inputs variables are declared as real as it values are of integers.

The second state is load where the butterfly process is carried out. The butterfly process requires two clock cycles for completing a single stage of the butter fly stage. Totally it requires six clock cycles to complete the butter fly process. The internal registers present in the architecture are defined to have the complex number and hence the two variables are used to declare the complex numbers.

In the second stage the complex addition is made on first clock cycle and the multiplication of the result is made with the twiddle factor on the second clock cycles.

In the third state is run, the outputs of FFT are obtained by one by one. The output will also requires a clock cycle and obtained by starting from X (0) up to X (7). As the output consist of real and imaginary values, two registers are used for declare the values. The output of the FFT obtained on the sequential order.

The controlling unit carried in the FFT process is by clock and reset input in the program. The clock input is used to drive the pipeline stage the FFT architecture and reset is carried for resetting the initial state. The twiddle factor value is fixed and it defined inside the code itself. The reset input is used to initialize the stage.

# CHAPTER 7

## DESIGN OF MODULATION

A basic communication system transmits and receives information broadcast over a carrier frequency $f_0$. This carrier is modulated in amplitude, frequency or phase, proportional to the signal x (t) being transmitted. For binary transmission, the modulations are called Amplitude Shift Keying (ASK), Phase Shift Keying (PSK) and Frequency Shift Keying (FSK).

In general, it is more efficient to describe a modulated signal with a projection of a rotating arrow on the horizontal axis according to

}

Where is a random phase offset, A(t) describes the part of the amplitude envelope and describes the frequency or phase modulated component. AM and PM/ FM can be used separately to transmit different signals. The CORDIC algorithm is used in rotation mode, it is co-ordinate convert form (R, Figure 5.1 shows the complete modulator for AM, PM and FM.

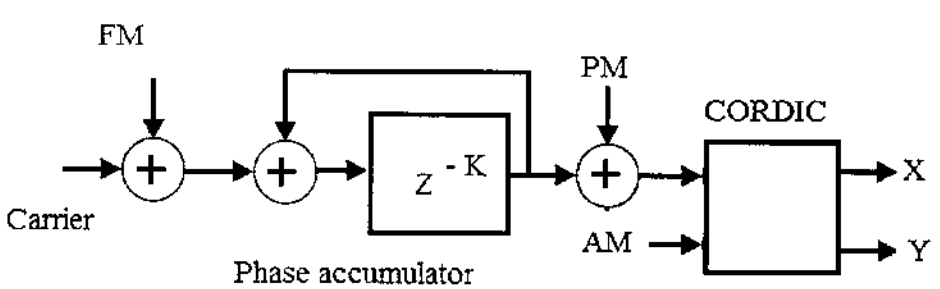



**Figure 7.1: Universal Modulator using CORDIC**

To implement the amplitude modulation, the signal A (t) is directly connected with the radius R input of the CORDIC. In general, the CORDIC algorithm in rotation mode has an attendant linear increase in the radius. This corresponds to a change in the gain of an amplifier and need not be taken into consideration for the AM scheme. When the linear increased radius is not desired, it is possible either to scale the input or the output by 1/ 1.6468 with a constant coefficient multiplier.

# CHAPTER 8

# RESULTS

## 8.1 DESIGN SOFTWARE

The implementations have been carried out using the software, ModSim. The hardware language used is the Very High Speed Integrated Circuit Hardware Description Language (VHDL). VHDL is a widely used language for register transfer level description of hardware. It is used for design entry, compile and simulation of digital systems. Modelsim is a simulation tool for programming {VLSI} {ASIC}s, {FPGA}s, {CPLD}s, and {SoC}s. Modelsim provides a comprehensive simulation and debug environment for complex ASIC and FPGA designs. Support is provided for multiple languages including Verilog, SystemVerilog, VHDL and SystemC.

## 8.2 SIMULATION RESULT OBTAINED FOR ALU

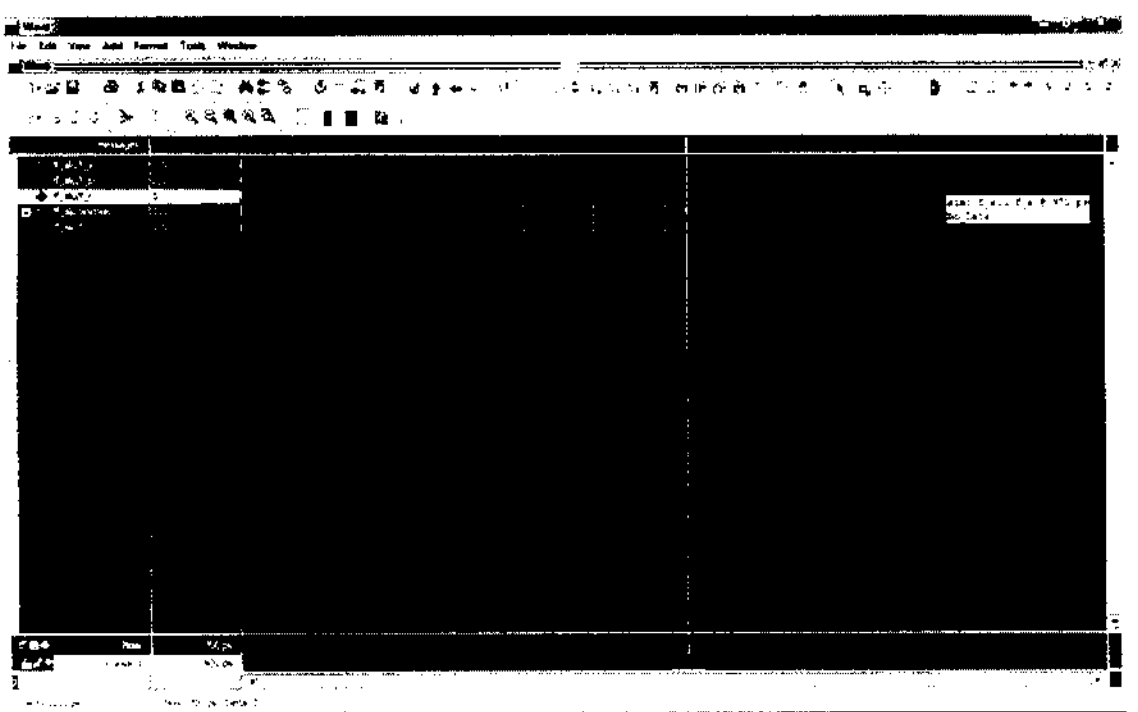


Figure 8.1: Simulation Result Obtained For ALU

## 8.3 SIMULATION RESULTS OBTAINED FOR MAC

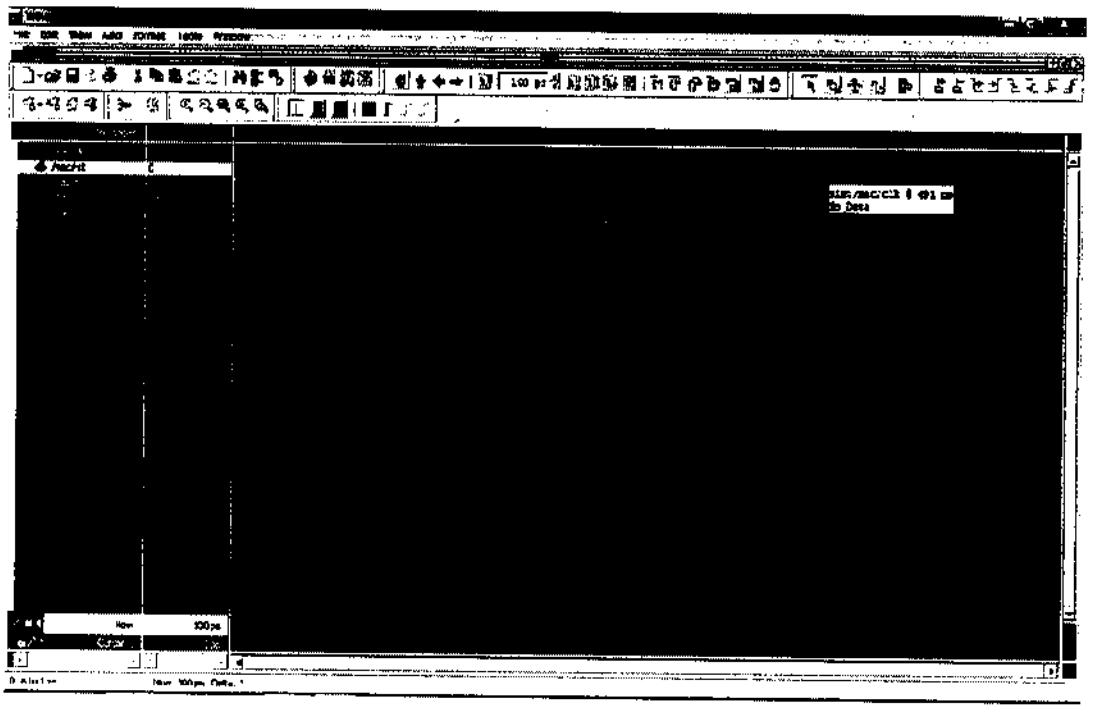


Figure 8.2: Simulation Results Obtained for MAC

## 8.4 SIMULATION RESULTS OBTAINED FOR SIN AND COS FUNCTION

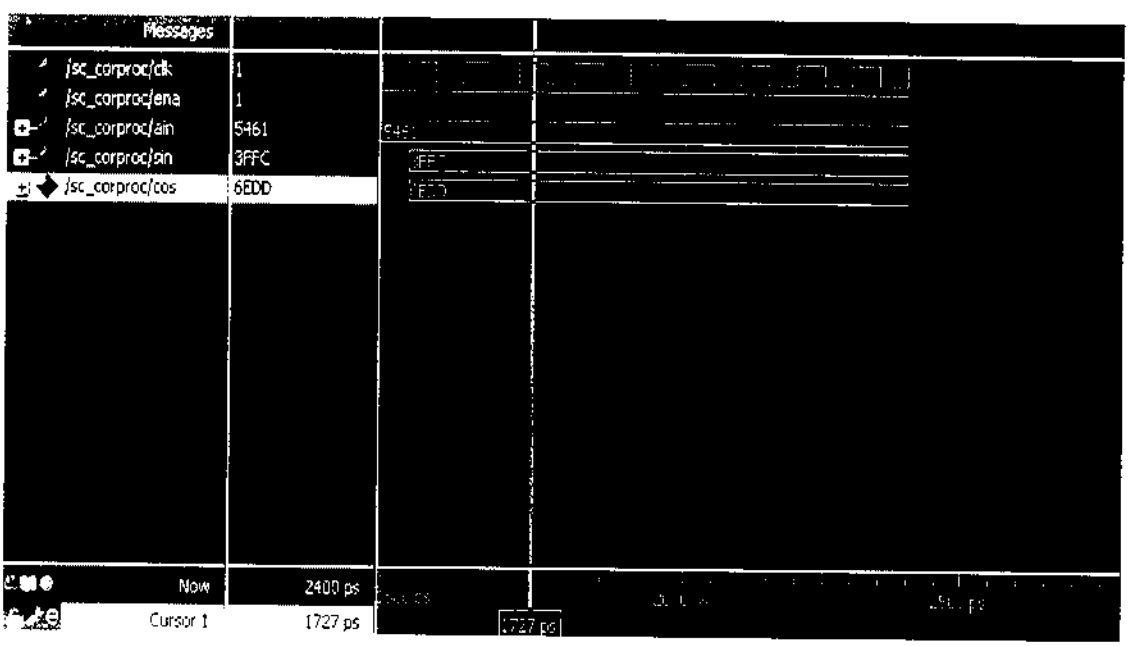



Figure 8.3: Simulation Results Obtained For Sin and Cos

## 8.5 SIMULATION RESULT FOR START STATE IN FFT

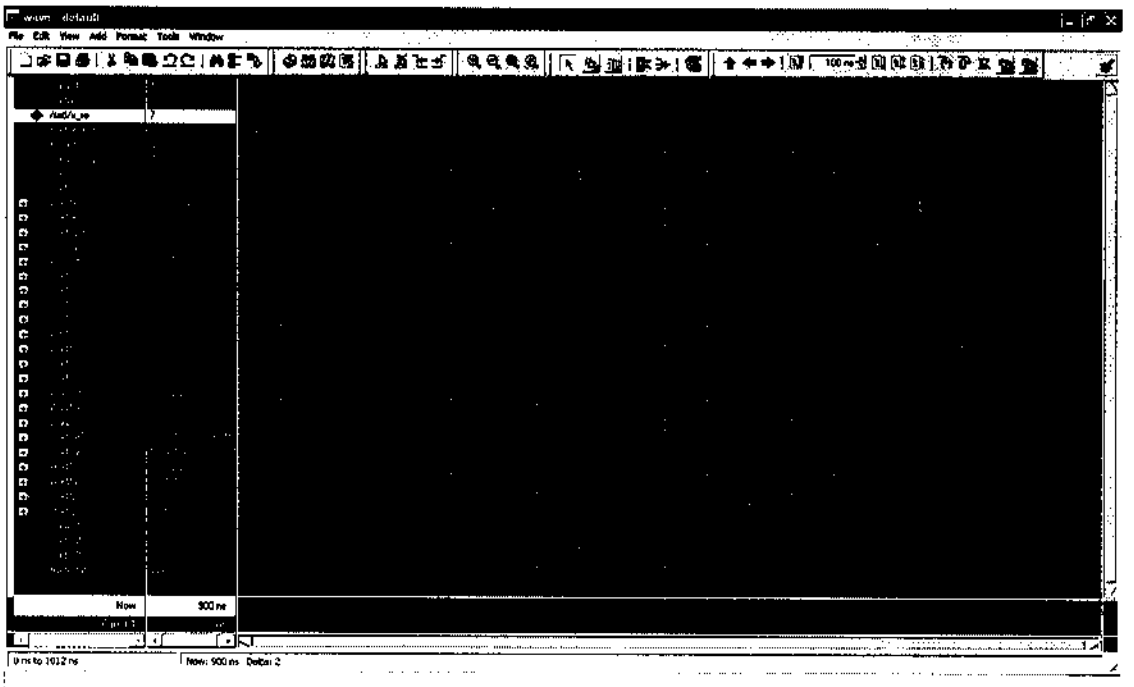


Figure 8.4: Simulation Result Obtained For Start State

## 8.6 SIMULATION RESULTS FOR LOAD STATE IN FFT

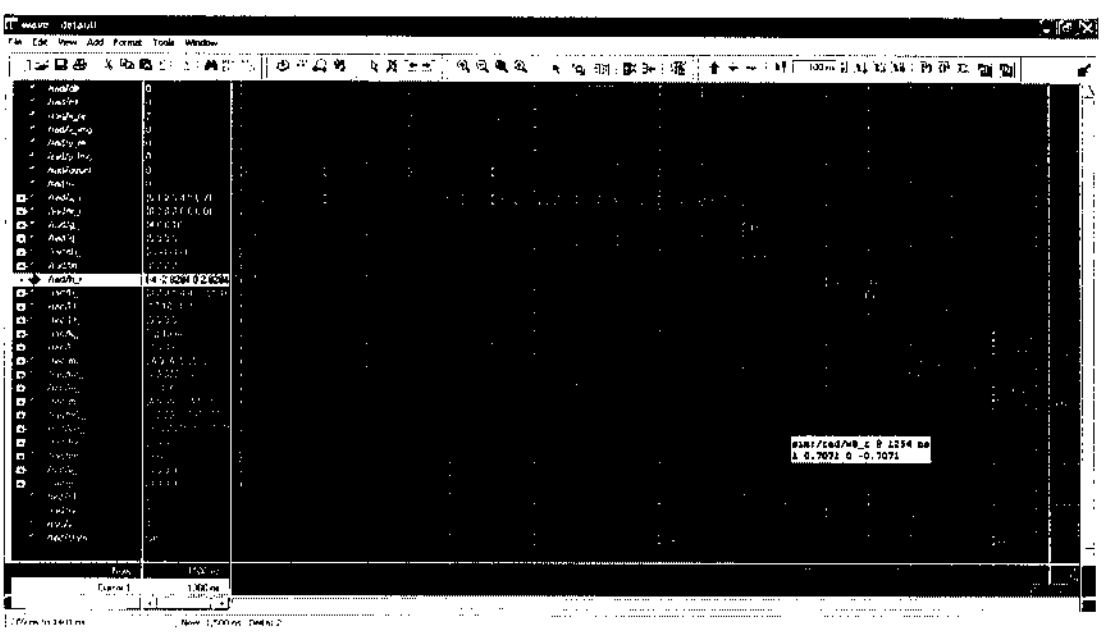


Figure 8.5: Simulation Results Obtained For Load State

## 8.7 SIMULATION RESULTS FOR RUN STATE IN FFT

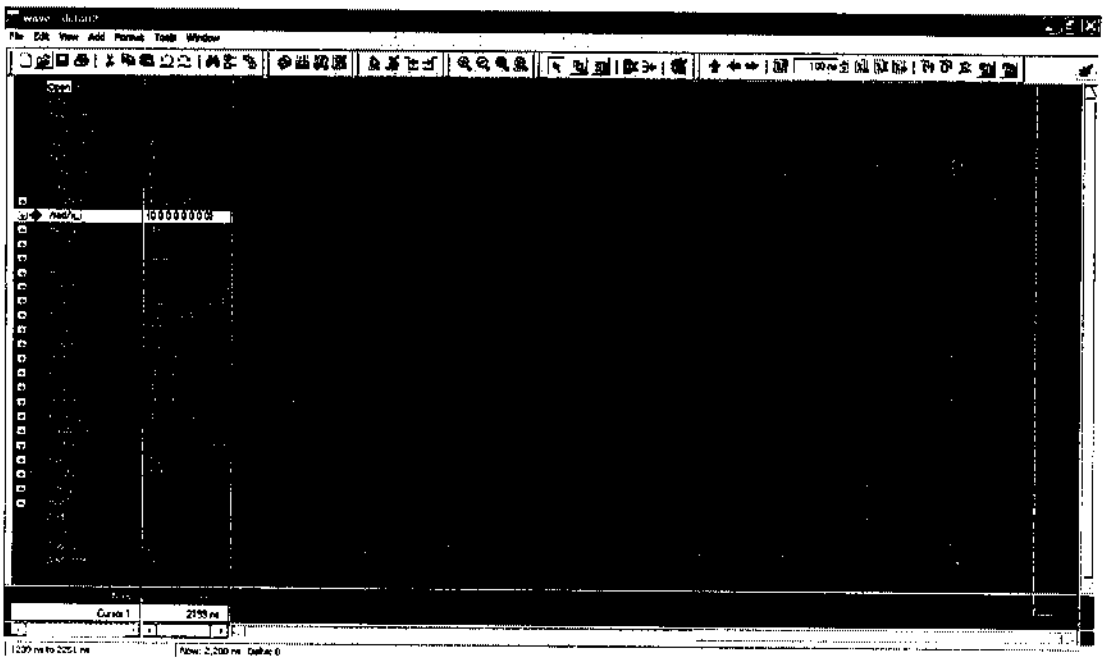


Figure 8.6: Simulation Results Obtained For Run State

## 8.8 SIMULATION RESULT OBTAINED FOR AM

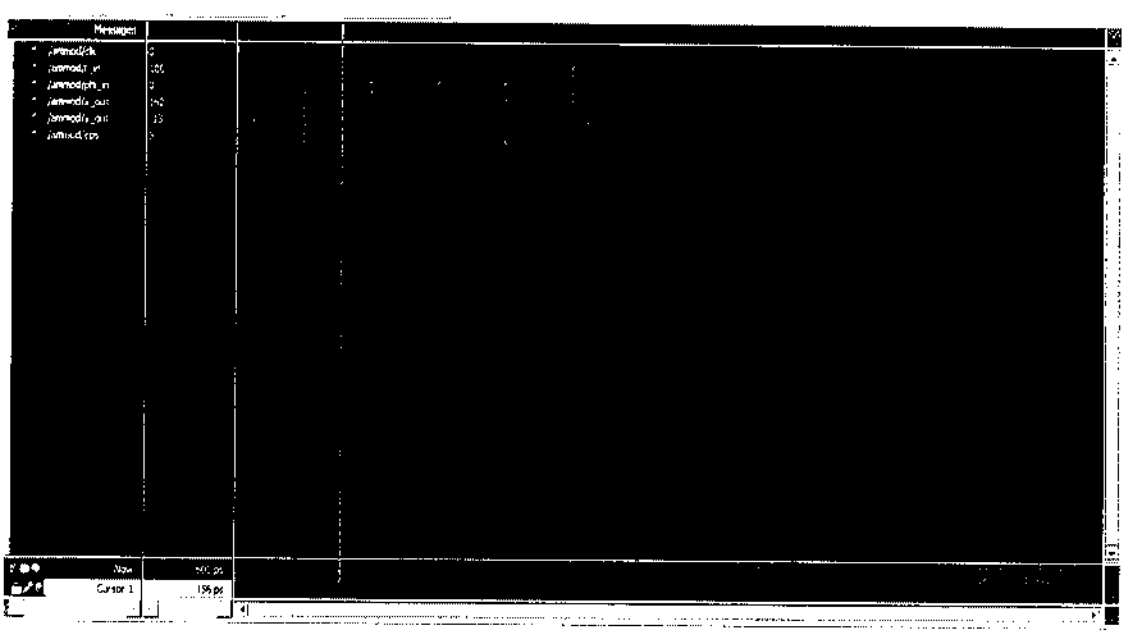


Figure 8.7: Simulation Result Obtained For AM

# CHAPTER 9

# CONCLUSION AND FUTURE SCOPE

## 9.1 CONCLUSION

The architecture and instruction-set design of an ASIP for SDR has been developed and given complete encoding of the op-code of the total instructions. The complete functionality of different modules has been present in the architectural aspects. So far we have modeled the entire data move, arithmetic instructions, program control and application specific general to the communication system and some of base band and channel coding instructions.

## 9.2 FUTURE SCOPE

In future we are going to incorporate these concepts to Reconfigurable ASIP (RASIP) for SDR. The improvements can be made in data transfer rate and some functionality like data encryption, which improve performance and optimize hardware utilization. The CDMA IS-95 and GSM forward and reverse links are considered, as an application to run on our RASIP to show its functionality.

# REFERENCES


[1]. Joe Mitola, "The software radio architecture", in IEEE Communications Magazine, Vol.33, No.5, pp26-37, May 1995.

[2]. K. Solomon Raju, Chandra Shekhar and R. C. Joshi "Behavioral Modeling and Simulation of Instruction-set of an ASIP for Software-Defined Radio", in National Conference on Issues and Trends in Wireless Networks (IT-WiNS), Patiala, pp.130-136, December-2004.

[3]. Walter H.W.Tutlebee, "Software-Defined Radio: Facets of a developing technology", from IEEE Personnel Communications Magazine, Vol.6, No.2, pp. 38-44, April-1999

[4]. M.J. Wirthlin and B.L. Hutchings, "A Dynamic Instruction Set Computer," Proc. Workshop FPGAs and Custom Computing Machines (FCCM '95), pp. 99–107, 1995.

[5]. Massimiliano laddomada ,"Reconfigurable issues of future mobile software radio platforms", wireless communications and mobile computing, pages 815-826,2003.

[6]. Man Young Rhee, "CDMA Cellular Mobile Communication and network security", Prentice Hall Publication, 1998.

[7]. C. Gonzalez-Concejero, V. Rodellar, : 'An FFT/IFFT bllock design versus Altera And Xilinx cores ', in conference on Reconfigurable Computing and FPGAs, 2008. ReConFig '08. International Conference.

[8]. Application note XAPP 290, Two Flows for Partial Reconfiguration: Module Based or Difference Based, available at www.xilinx.com.

[9]. M.D. Galanis, P. Kitsos, G. Kostopoulos, N. skalavos, O. koufopavlou and C.E. Goutis. "Comparision of the Hardware and FPGA Implementations of Stream Ciphers", electronics, circuits and systems proceedings of the 11th International conference, page 571-574, Dec,2004.

[10]. Peter J. Ashenden, "VHDL Standards," IEEE Design and Test of Computers, vol.18, no. 5, pp. 122-123, Sep./Oct. 2001,

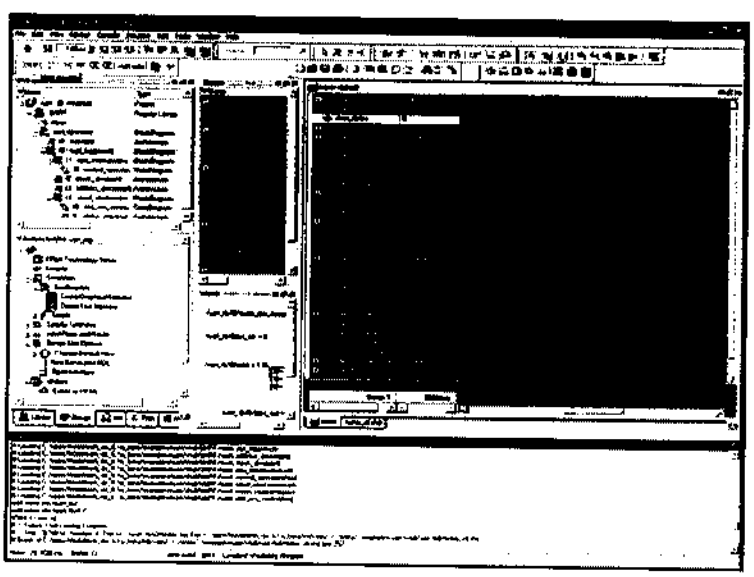
[11]. Timo Vogt, Norbert When, "A Reconfigurable Application Specific Instruction set processor for Convolutional and Turbo Decoding in a SDR Environment", proceedings of the conference on design, automation and test in Europe, 2008.

# APPENDIX

Design Creation to Realization

## ModelSim Designer

D A T A S H E E T



*ModelSim Designer combines easy to use, flexible creation with a powerful verification and debug environment.*

### Major product features:

- **Project manager, version control, and source code templates and wizards**
- **Block and state diagram editors**
- **Intuitive graphical waveform editor**
- **Automated testbench creation**
- **Text to graphic rendering facilitates analysis**
- **HTML Documentation option**
- **VHDL, Verilog, and mixed-language simulation**
- **Optimized native compiled architecture**
- **Single Kernel Simulator technology**
- **Advanced debug including Signal Spy™**
- **Memory window**
- **Easy-to-use GUI with Tcl interface**
- **Support for all major synthesis tools and the Actel, Altera, Lattice, and Xilinx place-and-route flows**
- **Built-in FPGA vendor library compiler**
- **Full support for Verilog 2001**
- **Windows platform support**

***Options:** SWIFT Interface support, graphics-based Dataflow window, Waveform Compare, integrated code coverage, and Profiler (for more details on options, please see the ModelSim SE datasheet)*

### The Easy to Use Solution for the FPGA Designer

ModelSim® Designer is a Windows®-based design environment for FPGAs. It provides an easy to use, advanced-feature tool at an entry-level price. The complete process of creation, management, simulation, and implementation are controlled from a single user interface, facilitating the design and verification flow and providing significant productivity gains.

ModelSim Designer combines the industry-leading capabilities of the ModelSim simulator with a built-in design creation engine. To support synthesis and place-and-route, ModelSim Designer is plug-in ready for the synthesis and place-and-route tools of your choice. Connection of these additional tools is easy, giving FPGA designers control of design creation, simulation, synthesis, and place-and route from a single cockpit.

A flexible methodology conforms to existing design flows. Designers can freely mix text entry of VHDL and Verilog code with graphical entry using block and state diagram editors. A single design unit can be represented in multiple views, and the management of compilation and simulation at all levels of abstraction is a single click away.

www.model.com/modelsimdesigner

Mentor Graphics

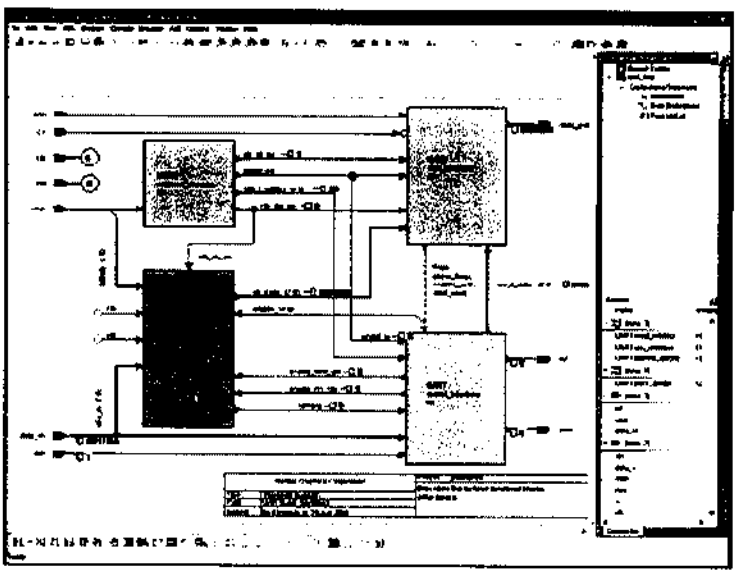
Implementation tasks are achieved through tight integration with the most popular FPGA synthesis and place-and-route tools in the industry. ModelSim Designer can either run these tools directly or externally, via scripts. Either way, the design data is maintained and used in a common and consistent way.

### Project Manager

The flexible and powerful project manager feature allows easy navigation through a design in order to understand design content and execute all necessary tasks. The project manager provides easy access to all design data for each individual design unit, throughout the design process. Synthesis results, place-and-route results, back annotated netlists, and SDF files are associated with the relevant design unit. Project archiving is a few simple clicks away, allowing complete version control management of designs. The project manager also stores user-generated design documents, such as Word, PowerPoint, and PDFs.

### Templates and Wizards

Creation wizards walk users through the creation of VHDL and Verilog design units, using either text or graphics. In the case of the graphical editor, HDL code is generated from the graphical diagrams created. For text-based design, VHDL and Verilog templates and wizards help engineers quickly develop HDL code without having to remember the exact language syntax. The wizards show how to create parameterizable logic blocks, testbench stimuli, and design objects. Novice and advanced HDL developers both benefit from time-saving shortcuts.



*The Block Diagram editor is a convenient way to visually partition your design and have the HDL code automatically generated*

### Intuitive Graphical Editors

ModelSim Designer includes block diagram and state machine editors that ensure a consistent coding style, facilitating design reuse and maintenance. These editors use an intuitive, graphical methodology that flattens the learning curve. This shortens the time to productivity for designers who are migrating to HDL methodologies or changing their primary design language.

Designers can automatically view or render diagrams from HDL code in block diagrams or state machines. When the code changes, the diagram can be updated instantly with its accuracy ensured. This helps designers understand legacy designs and aids in the debugging of current designs.

### Automated Testbench Creation

ModelSim Designer offers an automated mechanism for testbench generation. The testbench wizard generates VHDL or Verilog code through a graphical waveform editor, with output in either HDL or a TCL script. Users can manually define signals in the waveform editor or use the built-in wizard to define the waveforms. Either way, it is intuitive and easy to use and saves considerable time.

### Active Design Visualization Enhances Simulation Debugging

During live simulation, design analysis capabilities are enhanced through graphical design views. From any diagram window, simulations can be fully executed and controlled. Enhanced debugging features include graphical breakpoints, signal probing, graphics-to-text-source cross-highlighting, animation, and cause analysis. The ability to overlay live simulation results in a graphical context speeds up the debug process by allowing faster problem discovery and shorter design iterations.

### HTML Documentation

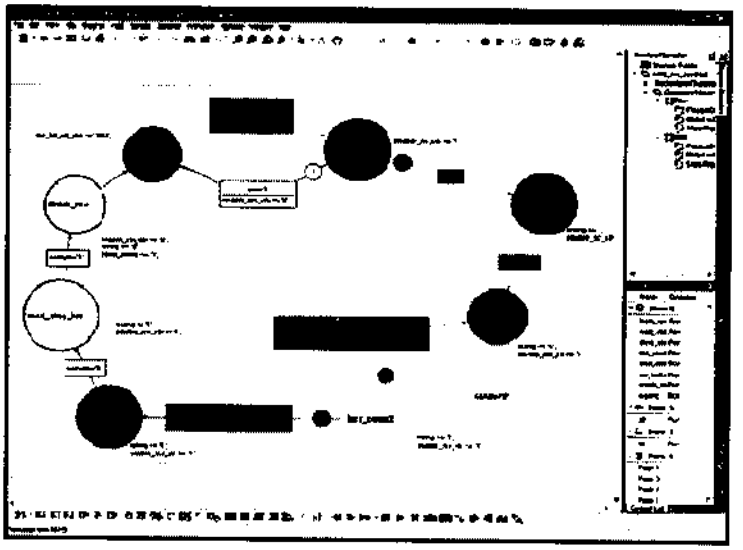
The HTML (HyperText Markup Language) Documentation option allows the user to output the complete design in HTML format. This allows the graphics, HDL, associative design files, and design hierarchy to be shared with anybody that uses an HTML browser. Design hierarchy and details can be viewed and navigated in any HTML browser, aiding communication and documentation.

### Memory Window

The memory window enables flexible viewing and switching of memory locations. VHDL and Verilog memories are auto-extracted in the GUI, delivering powerful search, fill, load, and save functionality. The memory window allows pre-loading of memories, thus saving the time-consuming step of initializing sections of the simulation to load memories. All functions are available via the command line, making them available for scripting.

### Intelligent GUI

An intelligently engineered GUI makes efficient use of desktop real estate. ModelSim Designer's intuitive layout of graphical elements (windows, toolbars, menus, etc.) makes it easy to step through the design flow. There are also wizards in place which help set up the design environment and make going through the design process seamless and efficient.



*The State Diagram Editor user interactive animation to track the simulation through the state machine and to check that all states were exercised.*

### Synthesis and Place-and-Route Integration

The industry's popular FPGA synthesis tools, such as Mentor Graphics Precision-RTL and most FPGA vendor synthesis tools, can be integrated into ModelSim Designer with push button convenience. Actel Designer and Libero, Altera Quartus, Lattice ispLEVEL, and Xilinx ISE place-and-route software are similarly integrated. Place-and-route results, together with SDF information, are automatically managed by ModelSim Designer after the process is complete, making them ready for post-place-and-route, gate-level simulation.

### FPGA Vendor Library Compiler

ModelSim Designer provides an intuitive mechanism to compile the necessary vendor libraries for post-place-and-route simulation. The compiler detects which FPGA vendor tools have been installed and compiles the necessary libraries as soon as the tool is launched, taking care of this step once and for all.

Visit our web site at www.model.com/modelsimdesigner for more information.