Impact of Intellectual Property Cores on Field Programmable Gate Array Designs

by

Lesley Shannon

A Thesis submitted in conformity with the requirements
for the Degree of Master of Applied Science in the
Department of Electrical and Computer Engineering
University of Toronto

Impact of Intellectual Property Cores on Field Programmable Gate Array Designs

Lesley Shannon
Master of Applied Science,
2001
Department of Electrical and Computer Engineering
University of Toronto

Abstract

Intellectual Property (IP) design is a rapidly growing industry and designers and users are being challenged to develop infrastructure and standard interfaces for this new industry. This research studies the impact that Intellectual Property cores have on the design of Systems-on-Chip (SoC) implemented on Field Programmable Gate Arrays (FPGAs). To obtain an understanding of the current state of core technology, a system was built using multiple IP cores. A core was designed to learn about core design issues while the remaining cores were obtained from vendors.

The results were slightly discouraging as it is not a simple process to incorporate third party cores into a design, nor was it easy to obtain cores for the system. Still, the experience has provided much insight as to what problems must be addressed in the IP industry to facilitate a design methodology that includes the use of IP. These challenges include basic concerns such as interfacing difficulties as well as documentation problems that have been grossly underestimated. Yet the issue remains: circuits are becoming too complex to custom design and achieve the desired time-to-market of a product; so how can we interface IP from vendors to a system design in a methodical and timely fashion?

Acknowledgements

I would like to thank my supervisor, Paul Chow, for his unending patience and willingness at any time to answer any of many questions- even if they were not directly related to my thesis. This has been a wonderful learning experience that has convinced me to continue my studies. I am thoroughly indebted to you.

To my parents, my brother Matthew, and Reza thank you for your love and support and belief in my ability to go the distance. To my extended family, thank you for your unconditional love and support. Thanks also to Joanne and Sara for their support, acceptance of my prolonged absences, and insuring that I got a decent meal when things got hectic.

To Alex, Amy, Courtney, Guy, Marcus, Nirmal, Paul K., Paul M., Scott, Ted, Val, and Vince, my friends who took the time to read my drafts, taught me the tricks of the trade, and informed me of any new research information they came across. I can't thank you enough for your time and patience.

To the rest of my friends from the Graduate Labs: Ali, Andy, Dan, Derek, Elias, James, Karen, Ken, Kevin, Neil, Peter, Rob, Robin, Sirish, Steve, Warren, and Yaska; I always knew that school was fun.

To Tim and Eugenia, thanks for making my computer work so that I could do my thesis.

I would also like to thank the many people in industry without whom this work would be significantly less complete: Altera's Maria D'Souza, Joe Hanson, Ralene Marcoccia, and Karen Virk; Lance T. Lehman, Barrister & Solicitor; Memec's Marcia Cunningham and Andrew Goodwin; Altera Toronto Design Centre's Terry Borer and Jonathan Rose; Jim McRoberts from the VCX; Xilinx's Tony Williams and Ralph Wittig.

Finally, the financial support from the Natural Sciences and Engineering Research Council made this work possible.

Contents

List of Figures

List of Tables

Glossary

AHDL	Altera Hardware Design Language
ASIC	Application Specific Integrated Circuit
BER	Bit Error Rate
DAC	Design Automation Conference
EDA	Electronic Design Automation
Erasure	Used to indicate the reception of a signal whose corresponding symbol is uncertain
ESB	Embedded System Block
Firm core	IP cores that are a compromise between hard cores
FPGA	Field Programmable Gate Arrays
GUI	Graphical User Interface
Hard core	IP cores that are delivered as netlists that have been fully placed and routed.
IP core	Intellectual Property core
LCs	Logic Cells
LEs	Logic Elements
puncturing	Method to reduce transmission size by removing a specified fraction of the transmitted data in a predefined format.
RAPID	Reusable Application-Specific Intellectual Property Developers
RTL	Register Transfer Level
SoC	System on Chip
Soft core	IP cores that are delivered in the form of synthesizable HDL and are process independent.
Traceback depth	The depth to which a path is traced back through the trellis to find the point of convergence.
VC	Virtual Component
VCX	Virtual Component Exchange
Verilog	One of the two HDLs used in hardware design and considered an industry standard.
VHDL	VHSIC Hardware Design Language- The other HDL used in hardware design and considered an industry standard.
VHSIC	Very High Speed Integrated Circuit
VSIA	Virtual Socket Interface Alliance

Chapter 1

Introduction

Recently, it became apparent to the members of the chip design community that a "design gap" is forming– a gap between what gates an engineer could design per day and the actual number of gates available [1]. As there has not been a major breakthrough in the Electronic Design Automation (EDA) community's design tools, the number of gates an engineer can design per day has increased minimally while the gate density and speeds of the chips available to the market have been following Moore's law. Since an engineer is only capable of designing so many new gates per day with the tools that are presently available, the idea of design reuse in system designs has become very popular.

Traditionally, systems have been implemented on printed circuit boards using off-the-shelf Integrated Circuits (ICs) from different vendors. These chips contained the Intellectual Property (IP) of the company that produced them and were used to provide a certain functionality to the system into which they were incorporated. ICs were easily mixed because an infrastructure of interface standards had been developed. This infrastructure included testing, selection and verification of components so that mixing ICs from multiple vendors into one system design would not increase the final product's time-to-market. At that time, Field Programmable Gate Arrays (FPGAs) were used to implement glue logic to interface the different ICs and simple logic functions due to the restrictions of their small sizes.

Since that time, the smaller process technologies for ICs have achieved greater densities allowing whole systems to be implemented on one or two chips. Now, System-on-Chip (SoC) design is considered the trend for the future where a chip has at least one programmable microcontroller core and memory. Application Specific Integrated Circuits (ASICs) are becoming multi-million gate devices with more than 80% of their content determined by reusable hardware and software blocks (Virtual Components) supplied as cores [2]. Similarly, FPGAs have also increased in size such that their gate counts are approaching the millions, which allow designers to use FPGAs to implement entire systems.

Designers want the same benefits for their SoC designs as they had for designs implemented on circuit boards. The designers want the cores to hide the detailed functionality of the components so that they can more easily incorporate different components from various vendors to create the system design. FPGAs are also becoming popular as a substitute for ASICs as they allow the vendor to lower Non-Recurring Engineering (NRE) costs while still increasing functionality. They also allow the added benefit of reprogrammability, which can lower re-engineering costs if the design must be changed in the future.

In the competitive electronics industry, shorter product time-to-market may be essential to obtaining a niche in the market. To combat the relative loss of designer productivity due to the design gap, creating circuits from pre-designed blocks of IP, known as IP blocks, cores, or virtual components (VCs), is being explored. The theory is that these pre-designed blocks could be used in SoCs in the same way that off-the-shelf ICs are used to create systems on printed circuit boards. To further the growth of SoC designs and to simplify their implementation on FPGAs, both Altera and Xilinx have recently announced the release of a new FPGA architecture that includes both an embedded processor core and reprogrammable technology on one chip [3].

SoCs designs are presently hindered in their efficient and economical usage in industry by the lack of an infrastructure similar to that which is already in place for ICs. There is no support for the development and verification of cores for the designer and there is no standard way of interfacing cores from multiple vendors. Each core vendor creates cores, even ones with the same functionality, such that they require different logic to interface with the rest of the system design. This creates many difficulties when a designer tries to rapidly integrate these cores from multiple sources into a single design, as it increases the overall design time.

When the concept of reuse took off a few years ago, the buzzwords “intellectual property” (IP) created a whole new market niche. Steve Wolfe, editor of CAD Report newsletter, commented at Design Automation Conference (DAC) ’98, that everyone was talking about IP – it was going to be big business and transform the EDA industry. Ironically, he noted at DAC 2000, that people were not taking such a sunny view of IP– jokingly calling it the “Incredible Pain”. Many of the IP vendor businesses are losing money and the question that seems to be prevalent to the industry is: “Why doesn’t the IP business work?”. It is commonly agreed that it is a waste of valuable design time for a designer to rewrite standard cores, but market numbers suggest that the benefits of design reuse are failing to materialize as envisioned [4].

Obviously, designers realize the present condition of the IP industry is not providing the desired speed up in the time-to-market of a product. The question then becomes, why is this happening and what do vendors and users of IP have to do to make the usage of IP in their design methodology a seamless and painless procedure. This new technology has the potential to simplify and shorten the design process, but the problem issues must be addressed for this methodology to succeed.

1.1 Motivation

The motivation of this study is to explore the issues involved in using IP cores in the design of a circuit. These concerns include:

- What constitutes a core and what are the deliverables that accompany it?
- How difficult is it to use cores in a design?
- From where could a user obtain a core?
- What are the metrics that can be used to measure and quantify the quality of a core?
- Is there a design methodology for using cores in a system design?
- What are the methods and tools for testing cores?

There are multiple reasons that this study focuses on FPGAs as opposed to ASICs. First of all, the number of gates on an FPGA has increased to the point where they may be used to implement SoC designs and so it is possible to use multiple cores in one design. The fact that FPGAs are now large enough for implementing an entire system on chip, as opposed to just glue logic, makes the study of creating systems on this technology worthwhile. IP cores have also become recently available for FPGAs, which should shorten the design time of new products. Finally, the tools and hardware for FPGAs are more easily accessible than ASICs.

1.2 Objective

The objective of this research is to study the impact that Intellectual Property cores have on the design of Systems-on-Chip implemented on Field Programmable Gate Arrays. The first step is to learn more about cores by building several systems using cores. This will provide the design experience to discover the issues related to designing with cores. In conjunction with designing systems using cores, an actual core is designed to better comprehend the challenges and limitations of core design.

Having constructed multiple basic systems using cores, the next stage is to create a set of metrics to qualify and quantify cores and their effects on system design. These metrics are then applied to the systems to obtain a measure of the quality of the cores and the systems that use them. To fully understand the system designs, the parameters of each of the different cores are varied to create multiple versions of each system.

1.3 Contributions

Initially, I had intended to build multiple systems to determine the state of the new technology in industry. Unfortunately, I had naively believed that it would be easy to get a large number of cores from various sources so that it would be possible to construct different industry size systems. This belief was unjustified and it proved impossible to obtain the number of cores required to build multiple systems. Instead, only one system was created using the cores that were available at the time of the design.

Secondly, a convolutional encoder core was designed as a parameterizable module to learn the drawbacks of designing in VHDL and the facility of including a core designed in-house in a system design. Metrics were created to characterize a system design methodology that uses cores as well as a core design methodology. By varying the parameters of the cores in the circuits, multiple versions of the system were implemented. These different versions were then tested against the metrics developed for this study. The results were used to draw conclusions as to what is the state of the present state of IP technology in industry.

1.4 Thesis Organization

This thesis is divided into six chapters. Chapter 2 looks first at the state of IP technology in industry today, outlining the definition of a core, the different individuals involved in the core industry, and the opinion of both industry and researchers as to issues of using cores in designs. Chapter 3 discusses the circuits used in this study. Chapter 4 describes the metrics used to characterize the designs to be tested. Chapter 5 presents the results obtained for each design and Chapter 6 concludes this thesis with suggestions for future work.

Chapter 2

Overview of the Present IP Industry

This chapter provides a summary of the state of IP technology presently used in systems designs. The different types of cores are described as are the different individuals involved in the IP industry. Finally, the different organizations and programs that have been developed by industry to promote the design reuse strategy are described. Before beginning this discussion, it is important to understand the definition of a core and what separates it from a normal custom design.

An IP core is a proprietary module of functionality that meets a certain specification allowing design reusability. The core may or may not be parameterized depending on its functionality, but there must be some documentation and a method of verifying the core functionality provided by the vendor. Unfortunately, the definition is made somewhat ambiguous by the word, “specification”, as there is presently no industry wide standard. Different companies require that their cores meet different specifications, which is one reason why it is so difficult to incorporate cores from multiple vendors into a single design. Although different organizations have suggested standards that cores should meet, unless they become accepted by the IP industry as a whole, the problem of integration remains.

2.1 Types of Cores

IP cores are implemented at different hardware description levels, which has resulted in the creation of three categories of cores: *soft*, *firm*, and *hard* [5]. A core is categorized by the form of its deliverable and each of the three choices offers certain advantages.

Soft cores are delivered in the form of synthesizable HDL and, therefore offer the benefits of being flexible and process independent. Their major disadvantage is the lack of predictability in terms of timing, area, and power. Security may be another issue as the source code must be provided, although potentially encrypted, for integration into the final design. *Hard* cores are delivered as netlists that have been fully placed and routed. These cores have the advantage of being predictable and can potentially be optimized for minimal power and size for a specific process/vendor. Consequently, the lack of flexibility makes them less portable but also easier to protect because there is no requirement to supply RTL. *Firm* cores offer a compromise between these two extremes and are defined as any core that is neither a soft nor hard core by specification, such as a core provided as a netlist.

Most IP cores for FPGAs are provided as soft cores. This provides the possibility for the user to adjust parameters, creating greater reuse potential. The soft cores are often supplied in an encrypted format, so that the actual Register Transfer Level (RTL) is not visible to the user but the placement and routing are still flexible. In these instances, if the core is parameterized, a header file or Graphical User Interface (GUI) is used to provide the user with access to the parameters. For cores with timing-critical aspects, such as PCI interface cores, certain signals may be pre-routed or assigned to specific routing resources to meet timing specifications. These cores may be categorized as firm cores.

Since a core is a pre-designed block of code, it is possible that this will affect the design into which it is being included. The setup and hold times along with the handshaking signals of the core may be unchangeable, which means that the rest of the circuit must be designed to properly interface with the core. If a core has a fixed placement or a partially-fixed placement, then this may affect the placement of the rest of the circuit. The maximum clock rate of the final circuit could be set by the core, which cannot be pipelined any further.

2.2 Stakeholders in the Core Industry

The people who are interested in the IP core industry may be divided into three groups: *third party IP vendors*, *third party IP users*, and *captive IP designers and users*. The *third* *party IP vendors* are those companies who are solely interested in developing cores to sell as a finished product or to complement the sale of silicon. This group of individuals is not concerned with trying to interface the core-product into a bigger design but, instead, works as a vendor. The Reusable Application-Specific Intellectual Property Developers (RAPID) is an association that represents both IP suppliers and integrators, giving the third party IP vendors a voice in industry [6].

The *third party IP users* work for companies who are trying to implement a large design. They wish to leverage the advantage of using a core created by an external source so as to speed the time-to-market of their product. IP users are developing application specific products and are not interested in developing in-house cores as there is a low chance for reuse by the company. Finally, there are the *captive IP designers and users* who work for companies who do extensive in-house design of products for a specific market. These individuals have many opportunities for design reuse as their design focus is on one particular market. Their companies have developed a design reuse culture for cores designed in-house. The company may also purchase cores developed externally as an investment to complement those developed by internal personnel.

2.3 Design Reuse Organizations and Programs

Since IP has become a fixture in the chip design industry, different organizations have been formed to promote design reuse standards. Their goal is to develop a set of industry standards to facilitate the usage of IP in designs and to simplify the interfacing of external IP to a design. The following describes groups interested in the developments of standards and their promotion to industry.

2.3.1 The VSIA

While design reuse appears to be the next innovation for the systems design industry, the innate challenges to creating a new set of industry standards have resulted in the formation of the Virtual Socket Interface Alliance (VSIA) in September of 1996. The Alliance was created with the vision of accelerating SoC development by specifying open standards that facilitate the “mix and match” of IP cores from multiple sources [2]. The membership consists of representatives from the systems, semiconductor, Intellectual Property, and EDA segments of industry.

The basic philosophy is that if physical components can be rapidly mixed and matched on a printed circuit board, IP cores in a standardized “Virtual Component” (VC) form should also be easily mixed and matched within an SoC. The VSIA hopes to create this environment by specifying “open” interface standards so that VCs (the VSIA term for IP cores) can be easily fit into “Virtual Sockets” with minimal (or no) glue logic. They require that this fit be established at both the functional and physical levels [2].

VSIA standards include those that are already industry standards, or open or proprietary data formats. The VSIA only develops new industry standards when none actually exist. Their goal is to create a standard format for core deliverables so that a core is independent of the unique design flow of each customer.

2.3.2 OpenMORE

Synopsys and Mentor Graphics have formed a collaboration known as the Open Measure of Reuse Excellence (OpenMORE) program [7]. It is an assessment program based on the Reuse Methodology Manual (RMM) that was jointly authored by the two founding companies. OpenMORE was released at the IP99 Europe SoC Conference, in November of 1999 [8]. They have chosen to define an IP core as a design that may be viewed as a stand-alone sub-component of a complete SoC design. Furthermore, the RMM defines soft cores are defined as soft macros or a core delivered as synthesizable RTL code, and hard cores as hard macros or a core delivered as a GDSII file. Hard cores are considered to be fully designed, placed and routed [9].

When designers decide to purchase IP cores for their designs, IP evaluation becomes an important stage of the design process. The OpenMORE program is supposed to facilitate the evaluation process by providing a format for a structured assessment of the reuse quality of the core. An IP developer enters data into a worksheet outlining rules and guidelines for both hard and soft cores. The final score obtained from this process allows the user to evaluate the developer’s method of core design.

The worksheet assessment is aimed at improving core reusability, thereby improving the speed and predictability of IP integration into the final SoC design. It must be remembered that while individual companies are developing IP design standards, they do not ensure that the desired core and actual core functionality may be easily communicated to others [9]. This is because there are no guarantees that an external company will have the same design reuse culture as a company who not only purchases third party IP but also designs their own IP.

The majority of OpenMORE users are systems companies that typically reuse internally developed cores as well as third party IP. Vendors are also using OpenMORE to try and make their cores easier to use for their customers, so as to reduce the number of hours spent on customer support. They also feel that strong reuse practices will help counteract the reputation that third party IP is hard to use [7].

2.3.3 RAPID and the VCX

Reusable Application-Specific Intellectual Property Developers (RAPID) was founded in 1996 by a group of companies that develop and sell IP. The main function of this association is to promote the use and acceptance of external IP products by the electronics industry. The goals of this organization are to establish guidelines and encourage the use of good business and design practices among members when working either within the electronics industry or with industry standards organizations to make IP easier to use and more accessible to designers [6].

The Virtual Component Exchange's (VCX) mission is to facilitate transactions of Virtual Components (VCs) within an efficient, international and open market infrastructure. As an industry-backed initiative, they have organized a "marketplace" for the buying and selling of VCs adopting the best features, services and structure of a mature stock and commodities market [10].

These two organizations have created joint ventures to speed the development by VCX of a global IP business infrastructure. The hope is that the input from RAPID will provide VCX with a broader perspective on important business and legal issues that will aid its product development [11]. There is also a private company called Design & Reuse which has created a Catalog and Yellow Pages to provide information on IP and SoCs. They also hope to leverage the IP business by providing e-Software but unlike the VCX, they are not focusing on providing solutions to the business and legal aspects of core transactions [12].

2.4 Perceptions of IP Cores

In the past few years, both the industry and research sectors have commented on the contribution of IP to the design process. Both the practical and theoretical impact on the design of SoCs has been argued. Even though the introduction of IP cores to the market is not recent, the place of cores in the market has not yet been established, and users and vendors in the core industry are experiencing the difficulties of this maturing period.

2.4.1 Industry User Sentiments

People involved in core-based design have many serious concerns about the lack of infrastructure. When using third-party cores, the most problematic element is documentation, which is closely followed by a desire for testbenches that provide 100% coverage so that the core design can be verified. Designers who purchase IP cores regard it as more than just an RTL file and want a guarantee of performance for their designs [13].

Systems designers find the advantage of using third-party IP questionable due to licensing problems as well as design integration process problems – especially if cores from multiple vendors are used in one design. These problems often introduce considerable delays that seem to negate the suggested advantage that the use of external IP can help designers achieve a shorter time-to-market.

Companies who are thoroughly committed to the use of IP in the design process have put considerable resources into developing their own internal IP policies and requirements. Not only does internally developed IP need to follow the design methodology, but external IP should be thoroughly evaluated before being purchased. The process of evaluating IP is lengthy – taking anywhere from weeks to months [14]. Obviously, the cost of evaluating IP is considerable. This can make the incorporation of IP into a smaller company's design process prohibitive.

Philips Semiconductors has devoted an entire organization to the job of acquiring IP, assessing its reusability, and disseminating it to the groups that deploy its products [14]. They have an internal design reuse methodology, but this does not mean that externally designed IP can be easily incorporated into their internal designs. Since there are no industry standards for IP core designs, the design methodology adopted by each company will differ, making IP integration more difficult.

2.4.2 Industry Vendor Sentiments

The experience an IP user has with the first core purchased from an IP vendor is crucially important. With the IP industry still in a relative stage of infancy, engineer-to-engineer "word of mouth" is key. In fact, some of the IP vendors claim that up to 80% of their sales result from word of mouth contacts [13]. Obviously, good relations with customers are essential, but the IP industry is also looking to the Internet. Not only are the vendors using the Internet as a place to list their products, it is also being used as method of delivery [13].

Programmable logic companies are also devoting major resources to developing IP. Companies like Altera and Xilinx view the development of a successful IP program as a key factor in the success of their new lines of million-gate devices. They have each developed IP design techniques to aid their customers' understanding of this new design approach, as well as programs to offer cores to customers by making deals with third party IP suppliers. These two companies are also developing cores and design tools of their own [13].

Although these companies sell IP cores, their goal is to decrease FPGA design time so as to sell more silicon. Licenses for cores may be sold such that the core is node-locked for a one year period. This allows the user to use the core in as many designs as they want during that year. Other silicon vendors will sell their cores for unlimited use in any design while some allow use in a specific design and charge a nominal fee for reusing the core in other designs. Third party IP vendors who sell IP as their main product have chosen different sales techniques. They may sell a core for a set fee, giving the user the ability to use it for a specific design or for any future design. They may also choose to sell a core on a royalty basis, limiting the risk of the consumer in purchasing the core. A combination of these two models is also used [14].

Altera generally sells only encrypted IP deliverables and does not allow the user to make changes to the source code. The only instance in which a user may be able to see the actual source code is if the product using the core is being migrated to an ASIC due to high volume of production. In this instance, being able to see the core code would be required. This will require the core user to pay an extra fee and is rarely done. Included with the encrypted IP deliverable, Altera provides drivers, and additional testbenches. The documentation for Altera's IP cores is found online and may be downloaded before a purchase is made. Altera's IP Megastore allows the user to download a test version of a core which the user may use in simulation. When compiled and synthesized, the test cores will provide the user with everything but the assembly file required to program a part.

Memec Design Company sells soft cores allowing the purchasers to obtain a netlist or the actual source code at a slightly greater cost. Memec chooses to allow their users to purchase and view the source code at a low price but requires that the design using the core be implemented on Xilinx FPGAs purchased from Memec or Insight. The company does not intend to make profits off cores sales, but intends to increase silicon sales by providing cores that a user needs in designing for their FPGAs. Although the development costs of most of the older cores have been recouped, the core purchasers are required to sign a license agreement that states that any improvements made to the core belong to Memec and not to the core user. This is because if the improvement were patented by the core purchasing company, the improvement could not be used by anyone else and it might impede the usage of the core by other designers.

2.4.3 Research Sentiments

The research published thus far on IP cores and their contribution to SoC design is relatively new. Much thought has been given to the protection of IP including watermarking techniques [15, 16, 17, 18], fingerprinting techniques [19, 20], a forensic engineering technique [21], and public-key cryptography [22]. A study has also been done to analyze the design lifecycle of core-based design [23] and some have suggested the need for a reorganization of the semiconductor industry to incorporate design reuse [24]. A method of characterizing the functional timing analysis for IP has been proposed [25] and the issue of the synthesis of interfaces between IP cores that use different signaling conventions has been addressed [26].

Some of the concerns for soft core design are similar to software design and it has been explained that certain concepts of software design and maintenance also applies to soft cores [27]. In fact, some researchers have chosen to use a C++ based development environment and an object oriented RTL model as opposed to dealing with structural reuse at the VHDL level [28]. IP cores are also being used in hardware/software co-design for embedded systems. A case study has been done using an existing commercially available engine control unit to delineate some of the design issues [29]. A hardware/software partitioning approach for core-based embedded systems suggested by research from C&C Research Laboratories has proven to lower power consumption [30].

There have been suggestions as to possible directions for IP development and its inclusion into the overall design process. For instance, the possibility of creating a database for IP cores, such as the DESPERADO project [31], has been made. The need for a database and other tools for exploiting soft cores was also suggested [32] and an EDA tool, called JavaCAD, has been developed that allows the user to simulate third party IP over the internet whether it has been purchased or it is just being demonstrated [33]. The DSP Solutions Group from Synopsys created a DSP system design environment for the commercial implementation of an Adaptive Differential Pulse Code Modulation codec [34] and a new library layer to support both IP based and traditional in-house design methodologies [35].

Researchers have studied aspects of SoC designs built using specific IP cores such as microprocessors [36, 37, 38, 39, 40, 41], Application-Specific Instruction set Processors (ASIPs) [42, 43], PCI buses [44], Viterbi Decoders [45], and ISDN network routers [46]. There has also been some research focusing on the testing of SoCs. A method of testing embedded cores has been described, comparing the testing of an SoC to that of a System-on-Board (SoB) [47], and an implementation of design for testability (DFT) structures has been described that can reduce testing overhead [48].

2.5 Summary

Obviously, previous research has provided a look at how specific types of cores may be incorporated into SoC design. It has not, however, characterized the cores themselves. Since FPGAs have only recently been available in sizes large enough to support SoC design, all of the previous research has been for SoCs implemented on ASICs.

This research can be differentiated from previous work not only by the fact that it examines the overall effect of using cores in SoCs but also by the fact that all of these previous studies have considered IP cores used to build SoCs implemented on ASICs. Furthermore, an attempt is made to define a core and characterize the properties that should be inherent to core design. Finally, two systems have been designed using off-the-shelf cores to gain insight into the practical issues involved in using cores in system design.

Chapter 3

Circuits

This chapter describes the system built using IP cores. The system was composed of a transmitter and a receiver circuit that used Forward Error Correction (FEC) methods to correct both bit and burst errors. The IP cores were downloaded from Altera's online IP Megastore. Each core is available as an encrypted file that can be included in a design created using MAX+plus II or Quartus tools. The MegaWizard design tool provides a GUI for the newer cores that allows the user to access the parameters of the core and select the HDL. Other older cores have an AHDL header file that the user edits to change the core parameters to the desired settings.

When the design is compiled, an assembler file will not be created unless the user has obtained a license from Altera. Although the downloaded version is available to anyone, there is a fee for obtaining a license that allows the user to place and route the circuit. The documentation for each core is also available online at the IP Megastore and may be downloaded for free. When an individual purchases a core, testbenches may be included to illustrate the operation of the core. While licenses were provided by Altera for the cores used in this system design, no testbenches were made available.

To understand the issues involved with designing a core, a convolutional encoder was designed and used in the transmitter design. Another convolutional encoder was created as a custom design so that it could provide a benchmark for comparisons. While the custom design uses a modular design structure, it differs from the core by not providing the user with a header file that enables the changing of all the necessary parameters at an abstracted

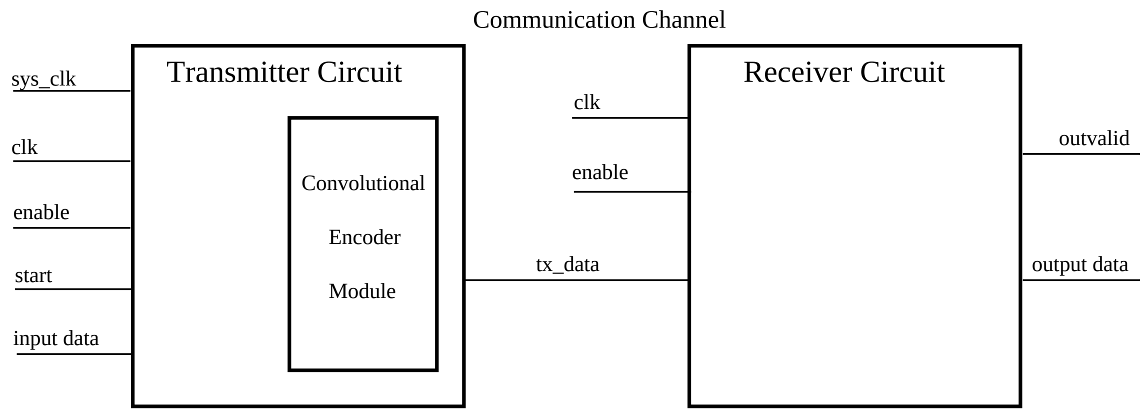

Figure 3.1: Block diagram of the transmitter-receiver system highlighting the convolutional encoder.

level. There is no information hiding so the user must understand the complete functionality of the encoder to change it for use in a different design. Figure 3.1 provides a block diagram of the complete transmitter/receiver system. The location of the Convolutional Encoder core is outlined in relation to the overall system.

3.1 Transmitter Circuit

The transmitter circuit that was chosen is a design representative of transmitters that are being designed and used in industry today. Transmitted data are susceptible to two types of errors, both of which must be detectable and correctable by the receiver for the data to be meaningful. One type of error is a burst error; this results when numerous adjacent bits of transmitted data are destroyed. The other type of error is bit errors, caused by random noise, which results in single bits of data being destroyed at random intervals. The ideal transmitter circuit, illustrated in Figure 3.2, would allow the user to simply connect the different IP cores to provide encoders that help protect the data from both types of errors.

3.1.1 Overview of the Transmitter

To protect the transmitted signal from the two types of errors, two different encoders will be used. First the input data will be encoded using a Reed Solomon Encoder, which will

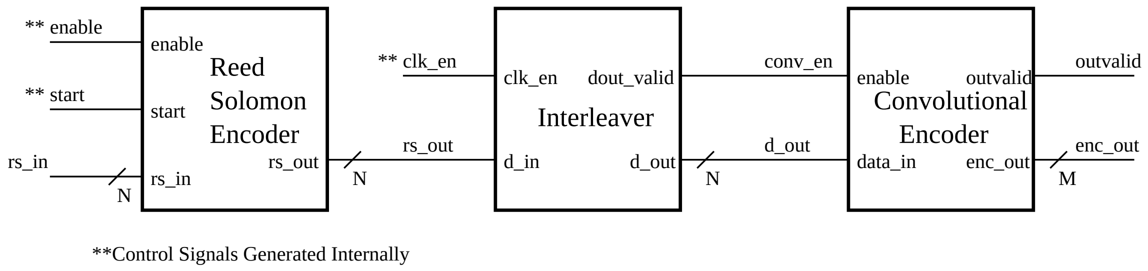

Figure 3.2: The ideal transmitter block diagram implemented with cores.

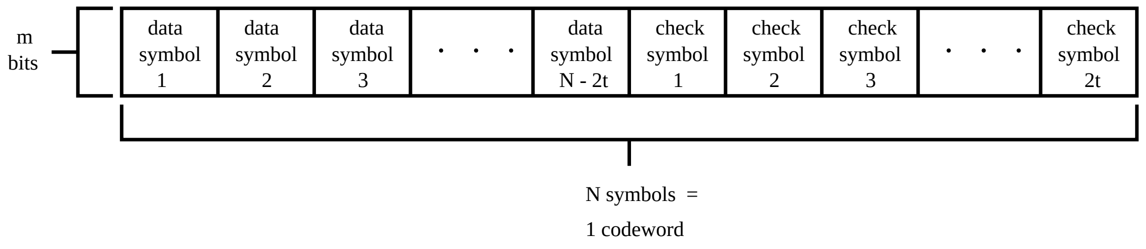

Figure 3.3: A picture of a Reed Solomon codeword.

generate a user defined number of check symbols for a specified number of data symbols. For a symbol width of m bits, the codeword size, N, can be a maximum of $(2^m - 1)$ symbols. If the user chooses to use $2t$ check symbols in each codeword, the Reed Solomon decoder algorithm is then able to correct a maximum of t symbol errors in each codeword. Figure 3.3 provides a pictorial description of a codeword to better illustrate the format of a Reed Solomon codeword.

The codeword is then read from the encoder into an interleaver that permutes the data. In this instance, it writes the data into the rows of memory after which the data is read from the interleaver and written out onto the data bus by columns. The data symbols are then latched into the convolutional encoder and each bit is encoded and transmitted serially. The signals in Figure 3.2 labeled with asterisks are control signals that enable the modules or change their mode of operation. This labeling convention will be used throughout this thesis to indicate all signals that are generated dependent on the present state of the circuit, using logic equations.

The number of errors in a Reed Solomon codeword are equal to the number of incorrect symbols in a transmission and is independent of the number of bit errors per symbol. It should be noted that the check symbols are able to correct errors in both the data symbols and the check symbols themselves. The Reed Solomon algorithm may also be used to correct erasures. Erasures occur when a signal is received indicating that the corresponding symbol value is uncertain. Each check symbol is able to correct one erasure.

The Reed Solomon core provided by Altera allows the user to choose to implement an encoder that supports not only erasures, but variable encoding, which means that the length of the codeword can be changed on the fly. While neither of these features was useful for the purpose of this transmitter design, they provide a flexibility that enables the core to be used in a greater number of designs.

By combining this error correction scheme, which encodes data as words, with an interleaver to interleave the encoded output, the transmitted signal is more robust against burst errors. If multiple codewords are interleaved together, the data symbols are permuted in blocks or convolutionally. This protects the codeword from a burst error as fewer symbols will be destroyed in each codeword. This makes the errors more likely to be correctable, thus making the transmission more robust.

The second encoding format for the data helps protect the transmitted information from random noise errors. It is a convolutional encoder that encodes each bit, producing two or more output bits for each bit to be encoded. If one of these bits is altered, the other encoded bits can be used to help determine the error. Obviously, this encoding method increases the amount of data to be transmitted across the channel by at least a factor of two. To reduce the transmission size, puncturing may be used to remove a specified fraction of the transmitted data in a predefined format.

An overview of the convolutional encoder core is provided in Section 3.2.1 and a description of the decoding mechanism used in the receiver is found in Section 3.3.1. While Figure 3.2 gives a general overview of the circuit functionality, Figure 3.4 illustrates a simplified version of the actual circuit that was required to implement the transmitter due to interfacing difficulties.

The Reed Solomon encoder is designed to receive a specific number of data symbols before generating check symbols. It may be run in a continuous mode, meaning that a codeword symbol, either a data or a check, will be on the *rs_out* bus on each clock cycle. Since this interleaver core is implemented in a fashion that allows it to either read in data

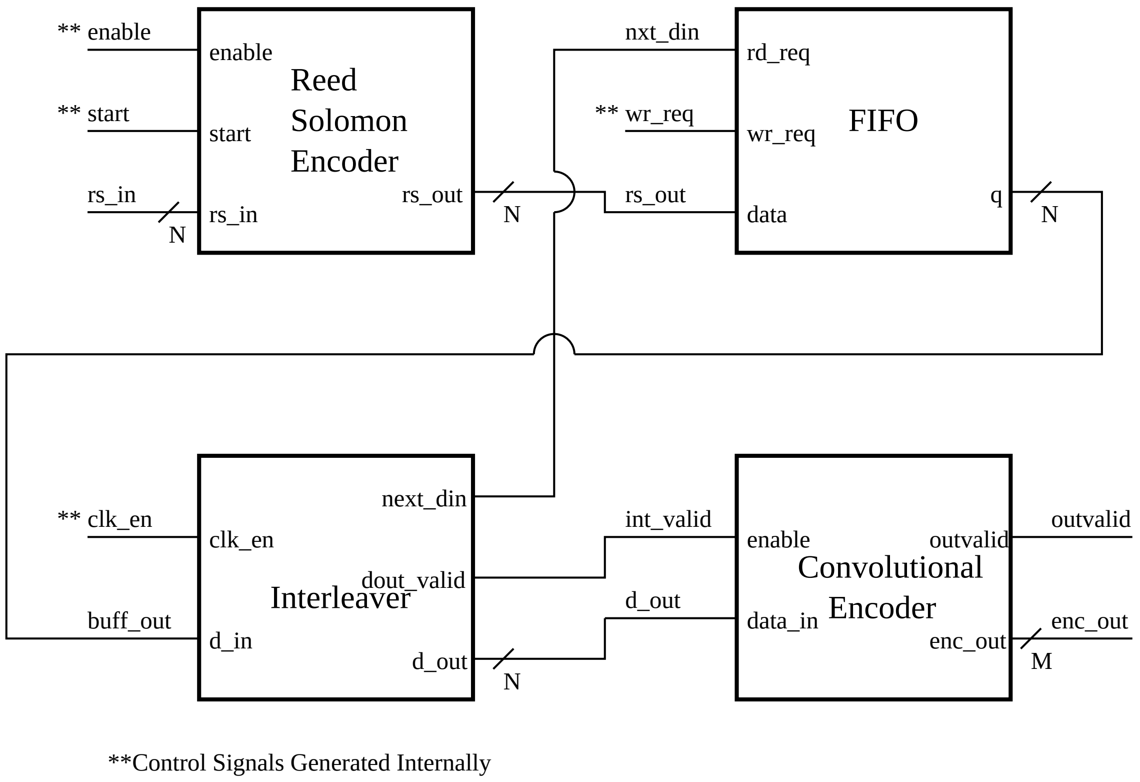

Figure 3.4: The actual transmitter block diagram implemented with cores.

to be permuted or, upon filling the interleaver, write out the data in its new order, it cannot receive new data while it is writing data to its output port. It is, therefore, necessary to buffer the output of the Reed Solomon Encoder. This also means that the encoder cannot run continuously and must be disabled when the interleaver cannot accept new data. Once the interleaver is full, it writes the data out to the convolutional encoder, which encodes the data serially from Most Significant Bit to Least Significant Bit. The settings of the convolutional encoder determine the settings for the Viterbi decoder in the receiver.

3.1.2 Design Decisions

It should be noted that the FIFO used in this design had one clock and performed synchronous reads and writes. To simplify the design, a look-ahead read was used to access the data stored in the FIFO. Another interesting aspect of this design is that there is an inherent conversion of data from a parallel to serial format. The data is latched into the Reed Solomon encoder as a data symbol and the convolutional encoder encodes the transmitted data bit by bit. This requires two clocks – one for the parallel data and another for the serial data.

Initially, the circuit was implemented with a serial clock and the parallel clock was derived on-chip. This was done to simplify the test vectors used to verify the chip's functionality. It was not the final implementation however, as clock division using on-chip logic would create timing problems. This is because the second clock, the parallel clock, would be using normal signal routing paths through the switches as opposed to the dedicated clock signal routing architecture. Therefore, when the final place and route of the circuit was performed, the parallel clock signal was implemented as a circuit input.

As discussed at the beginning of the chapter, a convolutional encoder core was created along with a custom design implementation. These two modules were interfaced to the transmitter circuit by separate component declarations that were selectively instantiated. Also, the interleaver allows users to select one of two methods for permuting data- either in blocks or convolutionally. It was decided that for the purpose of this study, the simpler circuit, the block encoder, would be used.

Finally, the encoded data produced by the convolutional encoder was left unpunctured. For the purpose of this study, transmission size is unimportant, as is increasing or decreasing the overall Bit Error Rate (BER) achieved by the receiver circuit. This is because the focus is on the design experience, as opposed to designing the best possible final system.

3.2 Convolutional Encoder Core

The focus of this thesis is to determine how using cores affects the design methodology used for SoCs. One underlying concept that must be addressed by the core design industry is what the actual definition of a core is. While the difference between HDL modules and hard cores is obvious, the difference between a soft core and an HDL module is a more obscure area. It is the responsibility of the core industry to provide guidelines and standards to differentiate between a core and normal HDL code. The VSIA is dedicated to establishing standards to be used in the design industry [5].

A soft core should be designed with testbenches that the user may utilize to verify the functionality to guarantee that it meets their specifications. Soft core designs are modules of code that are reusable in different applications. They have no fixed placement, which means that there can be no performance guarantees, but they provide a greater degree of flexibility. The designs may often be parameterizable, which enables the core to be used in a larger number of applications by providing a greater degree of freedom. Documentation describing the operation of the module is also essential to enable third party users to incorporate the core into their designs.

3.2.1 Overview of the Core

To obtain some insight into the present state of core design technology, a soft Convolutional Encoder Core was designed using VHDL. This core encodes a transmitted bitstream with specific convolutional codes for later error checking by a Viterbi receiver so that it may resolve possible bit errors. By defining the available parameters in the core, the user selects the minimal number of two generating polynomials as well as the convolutional codes. The ports to the core are described in Table 3.1 and the user defined variables are outlined in Table 3.2.

Figure 3.5 illustrates the functionality of the convolutional encoder module. As can be seen from the diagram, the right side of the figure contains the actual convolutional encoder that encodes the bits to transmit. The left half of the convolutional encoder module contains a register, a counter, and a multiplexer. The register stores the input symbol from the *data_in bus*. The counter is used to select a bit of the symbol through the multiplexer. This bit will be used by the convolutional encoder to generate the encoded output bits, *enc_out*.

The constraint length is equal to the number of registers less one in the encoding path. The user provides the values of the generating polynomials for the circuit illustrated in Figure 3.6. The bits set in the generating polynomial signify the connected taps to the delay path. The most significant bit is connected to the input A and the least significant bit is connected to the output B. A generating polynomial is required for each coded bit. Figure 3.6 displays the encoding circuit for an encoder with two encoding bits, a constraint length equal to five, and generating polynomials GA equal to 19 and GB equal to 29.

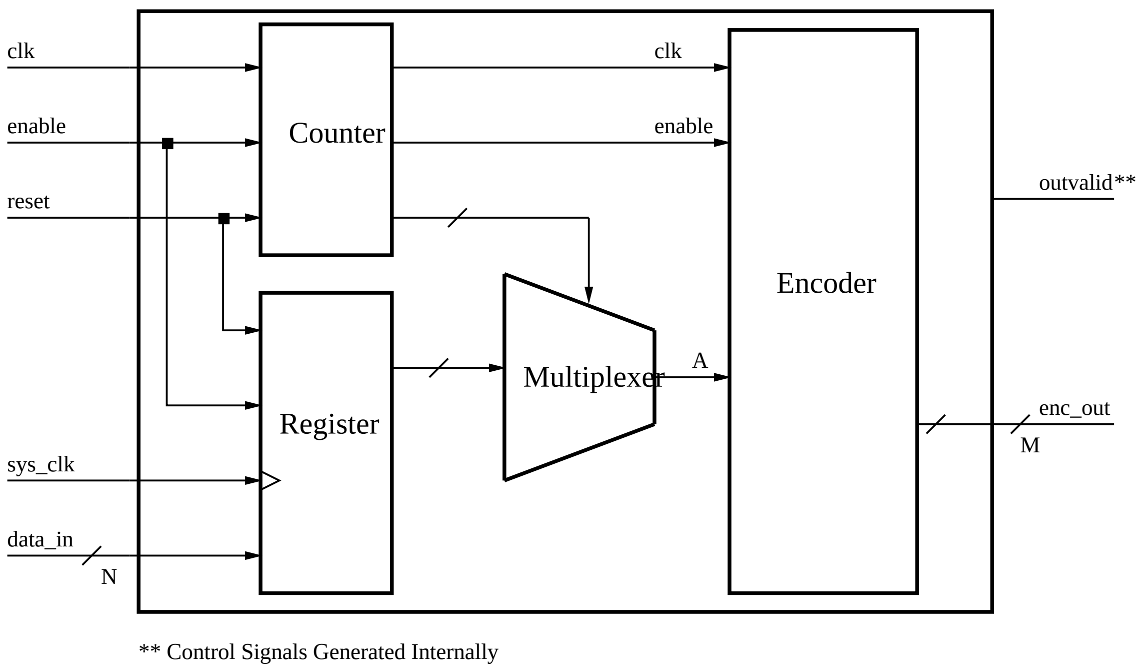

Figure 3.5: Block diagram of the convolutional encoder module.

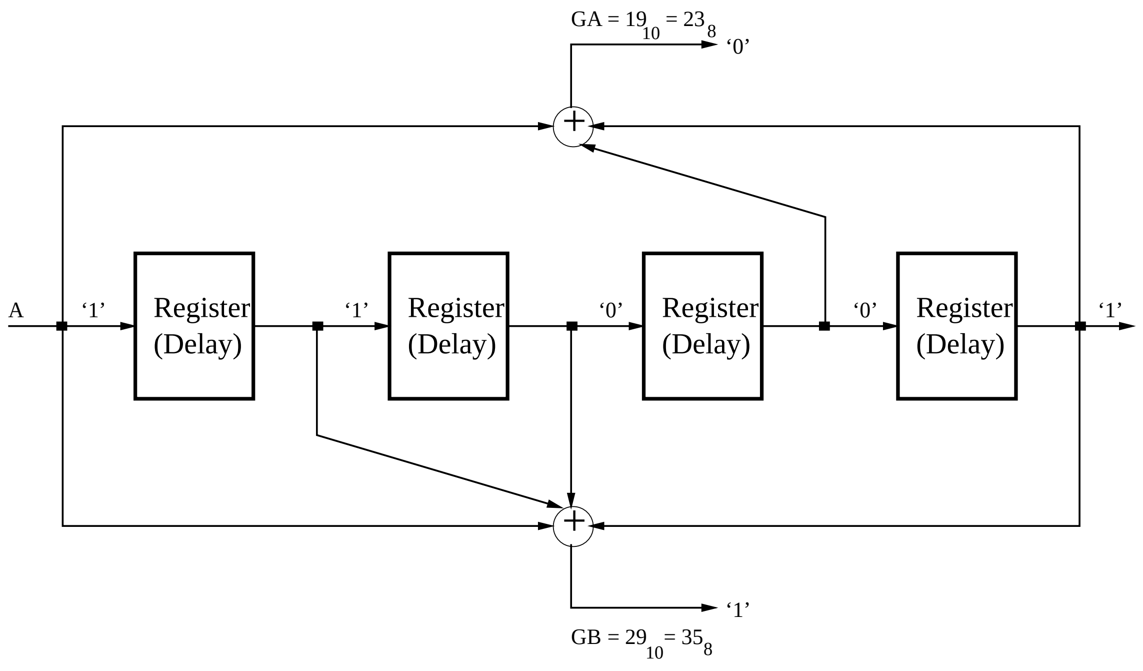

Figure 3.6: Block diagram of the encoding circuit illustrating functionality.

Port Name	**Direction**	**Function**
sys_clk	in	Rate at which the input words are read
clk	in	Rate at which the output bits are generated (4x, 6x or 8x the *sys_clk*)
reset	in	Asynchronous reset which occurs when *reset* = 1
enable	in	The convolutional encoder operates when this is logic 1
data_in	in	Input words are read in to be encoded
enc_out	out	The encoded output bits generated by the encoder
outvalid	out	Data on the *enc_out* bus is only valid when this signal is logic 1

Table 3.1: Inputs and outputs to the convolutional encoder core.

Parameter	**Valid Values**	**Description**
bus_width	4 or 8	Width of the parallel input data bus
N	2, 3, or 4	The number of encoded output bits
const_len (L)	4 to 9 (inclusive)	The constraint length $(NumRegsinpath) + 1$
num_sel_lines	2 or 3	The base 2 logarithm of the bus width
GA, GB	N/A	These two generator polynomials are always used
GC	N/A	This generating polynomial is used when $N > 2$
GD	N/A	This generating polynomial is used when $N = 4$

Table 3.2: User defined core parameters for the convolutional encoder core.

Noting the binary values for each of the registered inputs, and the specified connections given in Figure 3.6, the encoded output bits would equal 0 for GA and 1 for GB. As can be seen from the diagram, the transmitted bits are only sent in encoded form and no systematic bits, copies of the actual data bits, are sent. This differs from many other error checking

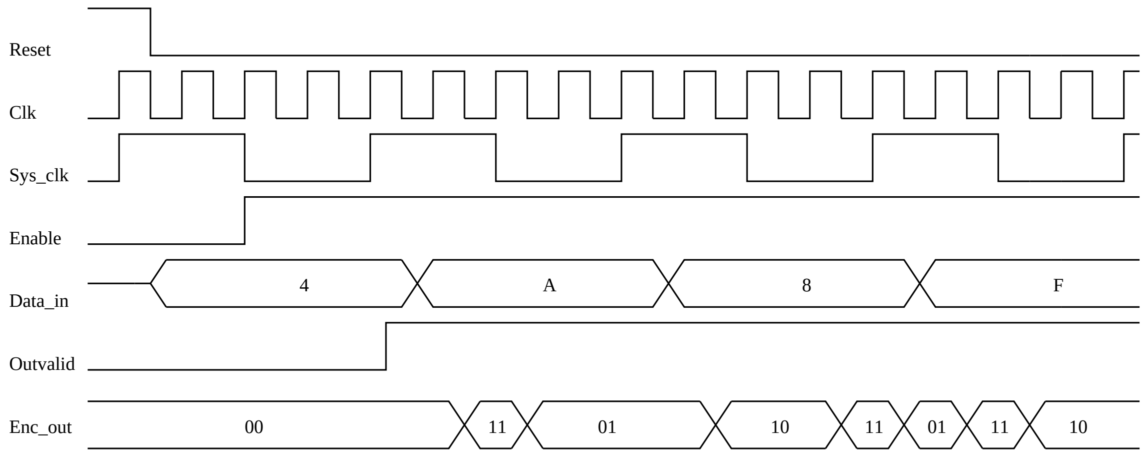

Figure 3.7: Timing diagram of the convolutional encoder.

schemes such as the Reed Solomon routine described in later sections.

Finally, Figure 3.7 contains a timing diagram to clarify any remaining questions as to the operation of the convolutional encoder. The system requires two clock inputs and has an active high reset for all of the registers and the counter in the encoder module. The *enable* signal is also active high and is used to turn the encoder on and off. When the encoder is enabled, the data symbols are latched from the *data_in* bus on the rising edge of the *sys_clk*. Every rising edge from the *clk* signal clocks a bit from the registered data symbol into the delay path and generates a new encoded output.

3.2.2 Design Decisions

The circuit is designed using two clock signals: one to load the data symbols, and one to output the encoded bits serially. Two clocks are used so that the overall circuit can run at faster speeds. If there were only one clock, it would have to be artificially divided by the number of input bits in the circuit. This is because parallel symbols are latched into the convolutional encoder and then encoded serially. This could significantly decrease the throughput of the transmitter circuit because the limiting factor for the speed of both the serial and parallel clock becomes the encoding circuit. If two clocks are used, then the serial encoding of the data operates independently of the parallel portion of the circuit, removing the timing dependencies.

The core user is also responsible for setting the number of select lines for the multiplexer. This is due to the difficulty of providing an equation that could calculate the number of select lines given the width of the *data_in* bus. The problem arises from the fact that VHDL does not support the logarithmic calculation necessary to determine how many select lines are required for any given width of the data bus.

All the generating polynomials were set to a maximum width. The MAX+plus II compiler and synthesis tools had to be used to create these designs because the proprietary cores are encrypted and only recognized by Altera tools. Unfortunately, the Altera compilation tools do not support all of the VHDL design constructs. For instance, a design cannot specify constant declarations in the entity declaration. This makes passing values from top level of the design to the architecture level more cumbersome.

Therefore, Aldec's Active-HDL, version 4.1, was used to design and test the convolutional encoder core as it supports all of the VHDL language constructs. When the core was compiled using Altera tools in the transmitter design, the core header file was edited to remove the constant declarations from the entity declaration and the user was required to specify necessary bus widths in both the entity declaration as well as in the constants in the architecture definition.

Finally, there were also problems arising from the actual semantics of VHDL. It does not allow a user to remove ports on a conditional basis. In other words, a bus cannot be removed from a design by declaring it to be of width zero. With parameterizable cores, it would often be useful to be able to remove a port from its core. For instance, the encoded output bits had to be transmitted on a single bus as opposed to separate buses for each generating polynomial. Overall, the present semantics makes core design more challenging and an update to the language that would include some of these features should be considered.

3.3 Receiver Circuit

The second circuit created using multiple cores was the receiver circuit, which served two purposes. Primarily, it is a more complex circuit logically and would not have fit on the

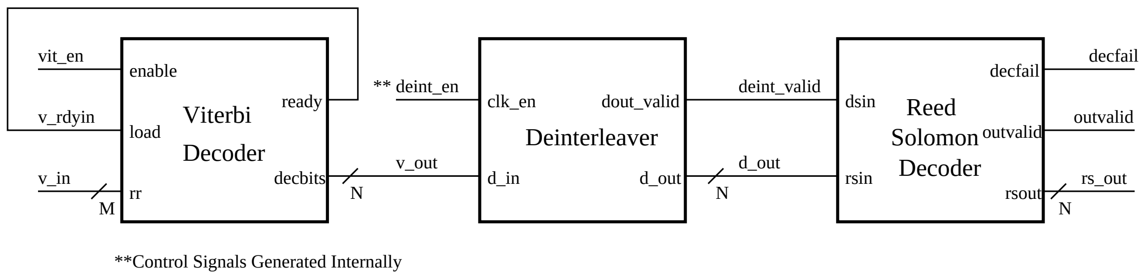

Figure 3.8: The ideal receiver block diagram implemented with cores.

older FPGA devices if designed using industry standard sized components. The reason this circuit is logically more complex than the transmitter is that large state machines are required to predict the expected value of a transmission and then to correct an erroneous bit. The second purpose was that it enabled the verification of the transmitter circuit by encoding data using the transmitter and verifying that the receiver decoded the same transmitted data.

3.3.1 Overview of the Receiver

Figure 3.8 is a block diagram of the receiver circuit, assuming that the cores interface in an ideal fashion. The serially transmitted bits, *v_in*, are the input bit stream to the Viterbi Decoder. Hard inputs were used for the decoder, which means that the encoder used one bit to represent the “hard” logic values of ‘0’ and ‘1’. A second input bit was used by the Viterbi decoder to indicate an erasure. If the data were received as soft inputs, the decoder would quantize the input into multiple levels.

Multibit inputs represent the degree of probability that the input was either a ‘0’ or a ‘1’. It would be used by the Viterbi Decoder to determine the output values based on probabilities as opposed to assumptions. While the use of hard inputs simplifies the circuit, the performance is worse than a circuit that uses soft inputs because by using a degree of probability as the input value as opposed to a binary value, the decoder is better able to resolve the bit errors. The design of the Viterbi algorithm makes it unlikely to correct burst errors because as the number of consecutive incorrect bits increases, the algorithm assumes that it has traveled farther and farther in the wrong direction.

The output from the Viterbi Decoder is deinterleaved so that the original ordering of

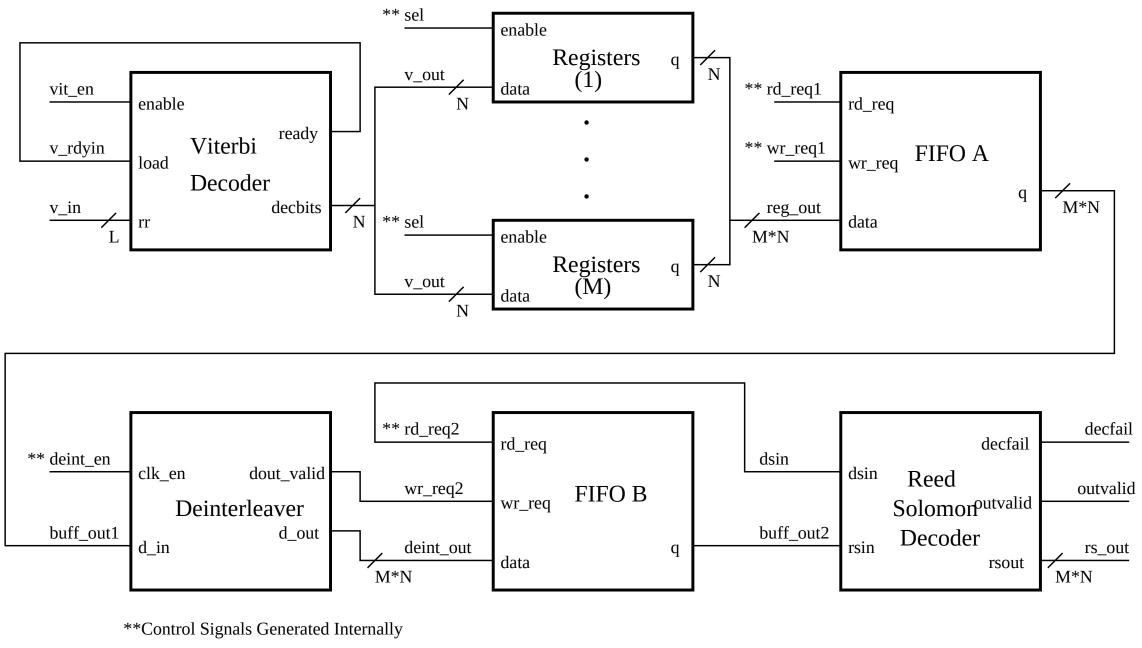

Figure 3.9: The actual receiver block diagram implemented with cores.

the symbols in the codeword could be regained. Then the Reed Solomon Decoder is used to correct any burst errors that had occurred during transmission. Recalling that the Reed Solomon Decoder does not differentiate between a single incorrect bit in a symbol and multiple incorrect bits per symbol, the algorithm is then able to correct a symbol regardless of the number of incorrect bits, subject to the number of symbols that can be corrected per codeword. The decoder indicates the number of errors corrected in a codeword via the *numerr* signal or if it fails to decode the codeword because there are too many errors, the *decfail* signal goes high. The output from the Reed Solomon Decoder is the initial codeword created by the encoder circuit. Both the data and check symbols remain and the designer must implement extra logic to strip off the check symbols from the codeword. This circuit required extra logic to be implemented for interfacing purposes; Figure 3.9 is a simplified version of the actual block diagram of the circuit.

The Viterbi core is coded to have a specific constraint length. To ensure that the decoder is able to determine what the convolutionally encoded bits were, the traceback depth must be set to at least five times the constraint length [49]. Since the Viterbi decoder outputs one or two bits at a time, the traceback length should be an even number. If not, when the designer trys to restore the serial transmission to its initial parallel format, it is more difficult to realign the bits. The number of output bits from the Viterbi decoder is equal to $ceil(v/(2^{L-1}))$, where v is the traceback depth and L is the constraint length.

The output bit(s) from the Viterbi decoder are then stored in a register to reconstruct the symbols from the serial transmission. Once all the bits of a symbol have been decoded, the symbol is stored in FIFO A as shown in Figure 3.9. These symbols are read into the deinterleaver when it is able to receive data. When the deinterleaver is full, the valid output is read into FIFO B. Values are read from FIFO B into the Reed Solomon Decoder when it is ready to decode the next codeword. The Receiver output is the decoded codewords from the Reed Solomon Decoder. These codewords still include both the check and data symbols and should, therefore be the same as the output from the Reed Solomon encoder.

3.3.2 Design Decisions

The transmitted bits were assumed to be hard inputs. This was because even though there is signal degradation over a noisy channel, it was not addressed, as dealing with transmission losses are beyond the scope of this study. Furthermore, no erasures were introduced because the size of the transmission was irrelevant. Instead, this research was more concerned with the actual size of the receiving circuit. Again, the FIFOs used in this design had one clock and performed synchronous reads and writes. To simplify the design, a look-ahead read was used to access the data stored in the FIFOs. This circuit required a significant amount of buffering between cores, which visibly increased the amount of glue logic. This was done to try and pipeline the circuit to increase the data throughput of the circuit. It was significantly slower due to the decoding speed of the Viterbi decoder, which required a large amount of time to decode the bits. Comparatively, the time required by the deinterleaver was negligible and even the Reed Solomon Decoder time was relatively insignificant.

The transmitter-receiver system was implemented assuming Reed Solomon data symbol sizes of 4, 6, and 8 bits. Although it would have been preferable to implement systems that used symbols with widths all equal to powers of two, the Reed Solomon circuit would be too large for 16-bit data symbol widths and two-bit data symbols are not useful to transmit. This is why a six-bit data symbol size was chosen as the third option. The Reed

Solomon Circuit was also available in three different formats– discrete, streaming and continuous. The streaming format was chosen because it allowed for some pipelining of the circuit, where as the discrete circuit would have caused a bottleneck in the circuit. The continuous decoder was not used due to its size, which would have required that only the largest devices of the newest parts could be used.

3.4 Summary of Design Experience

The third party IP cores used in these circuits reduced the the total number of gates that were left to be designed. Unfortunately, much of the benefit of the cores was lost due to incomplete documentation. The time saved by not designing the cores in-house was partially lost in trying to determine their actual functionality. In fact, learning the operation of the interleaver and deinterleaver may have been more time consuming then if they had been created as part of this research.

Over 75% of the design time was spent trying to understand how the cores functioned. Although it is reasonable to expect that becoming familiar with the cores should be a significant portion of the design, much of this time was wasted on determining the importance of undescribed input pins, comprehending unexplained output formats, and guessing the throughput time of the core. This meant that some of the abstraction that should have been provided by the core was lost. The main benefit of using the cores was that a scaled-down version of each of the circuits could be create by changing the core parameters. This simpler circuit could be verified first and once it proved operational, it was relatively simple to scale up the circuits to create the larger system circuits.

The design of the in-house core provided the opportunity to view IP cores from the designer's point of view. There were two main challenges – language restriction and using the available tools for core design and placement. The problem with VHDL as a language used for core design is that it does not offer the flexibility necessary to fully abstract the core. The other problem was that while the Active-HDL tool from Aldec provided a good environment for designing the core, the fact that MAX+plus II does not fully support the language which means that the initial design had to be changed.

Chapter 4

Tests and Observations

This chapter describes the metrics that were designed to address the different issues involved in using cores in system design. The procedures used to design, verify, and test the circuits for these attributes are outlined in Section 4.1. The metrics have been divided into two sections, the quantitative tests and the qualitative characteristics. The quantitative tests are outlined in Section 4.2 and the results are described in Chapter 5. The qualitative issues of using cores in systems design are discussed in Section 4.3, addressing concerns and problems that are faced by both core designers and core users.

4.1 Experimental Procedure

The design procedure can be broken down into three basic portions . The first concern was obtaining cores for the design of different systems. The next was the testing of each core and the overall system. Finally, the issue of choosing tool settings is addressed as they will affect the place and route algorithm, and therefore the numerical results obtained and described in Chapter 5.

The first problem was obtaining third party cores. Initially, it had been hoped that cores could be obtained from both Altera and Xilinx to build multiple systems on both platforms. This would allow a comparison of systems implementation using IP cores on both of the major industrial architectures. Unfortunately, the limited availability of cores that could be used to create systems has resulted in only one system. The second design restriction created by using cores in the circuit designs was that all of the research had to be performed on a personal computer running a Microsoft Windows operating system, in this case Windows NT version 4.1. This platform was necessary because Altera did provide most of their cores in a format that was compatible with the Unix versions of MAX+plus II.

Furthermore, it was not possible to obtain the necessary cores from Xilinx and its partners to create another transmitter-receiver system, which resulted in only one system being built and implemented using Altera IP cores. While this is disappointing, it is an indication of the availability of IP cores for FPGAs. The challenge involved in trying to obtain cores for designs from third party vendors made it impossible to obtain all the necessary IP blocks for the second system on a Xilinx platform.

The system was created and verified using version 9.25 of MAX+plus II. Since the Altera software does not support the VHDL constructs for File IO, all of the verification was done using the Waveform editor. This was not a problem when the verification was being performed at the core level, but it did create many challenges for system verification. This is mainly because verifying the operation of the Viterbi decoder requires the decoding of thousands of bits, which would be very tedious to draw using the waveform editor.

One solution is to use vector files to generate inputs to the Waveform Editor, but the inputs must change at predictable time intervals. Since inputs to both circuits are dependent on handshaking signals and do not have predictable timing characteristics, vector files cannot be used to generate the input signals. As the Altera simulation tool was not providing a simple method of verification, Aldec's simulation tool Active-HDL version 4.1, which does support file IO, was purchased to simplify the verification process. Unfortunately, even though the company claims to support Altera designs, the tool does not support all of Altera's Megacores, which meant that the tool could not be used to verify the receiver circuit. In the end, MAX+plus II had to be used to verify the basic operation of the systems, due to the lack of tools available for testing systems with cores.

Once the system had been verified to operate correctly, the tests described in Section 4.2 were performed. The tools were set to optimize the place and route algorithm for speed. For the smaller circuits MAX+plus II, version 9.25, was used to place and route the circuits on the FLEX 10KA devices. The larger circuits had to be placed and routed on APEX devices, which required that Quartus software be used. The Quartus software used for this research was version 2000.5 which does not fully support core based design. The circuit can be placed on an APEX 20K device but the timing analysis fails.

The FLEX 10KA devices were used because they offered high-speed performance and would fit the majority of the circuits. For the larger receiver circuits, the APEX 20K device was chosen because it had a larger number of programmable resources. When comparing resource usage on these two types of devices, it is important to qualify the terminology as MAX+plus II reports the number of Logic Cells (LCs) and Memory bits used to place a circuit while Quartus states the number of Logic Elements (LEs) and Embedded System Block bits (ESB bits) used in the circuit placement.

The data sheets for the FLEX 10K devices reveal that a Logic Cell is simply another term for an Logic Element and they are equal units of measurement. Similarly, ESB bits are comparable to the Memory bits. The final unit of the Altera FPGA architecture to be discussed is the Logic Array Block (LAB). It is a combination of LEs and provides a slightly higher level of architectural abstraction. It is important to note that Flex 10K LABs are composed of 8 LEs while APEX 20K devices have 10 LEs in their LABs. Since there is a faster interconnect running internal to the LAB, changing the number of LEs in the LAB could effect the timing of the circuit after it has been fitted to a device.

4.2 Tests

The following section describes the metrics used to measure the quantitative values of a core and a design that uses cores. In chip design, area usage and the maximum clock rate of a design are important markers of the quality of the design. Obviously, this is also true for designs that use cores but modifiability is also important to increase the reusability of the core. These tests have been chosen to reflect the characteristics that are important to all chip designs as well as characteristics that are uniquely identifiable with cores.

4.2.1 Core Parameterization

Many cores have modifiable values that allow the user to scale the core or select different functionalities of interest. This can encourage greater reusability for cores such as the Reed Solomon and Viterbi Decoders as they can be used in different applications. To test this characteristic of a core, multiple versions of each core were created to see the effect each core parameter has on the number of logic blocks and memory bits required for the placement of the core. If a core is well designed, it should scale its resource usage proportionately to the increase of the calculation size or functionality.

The number of logic blocks and memory bits required for the convolutional encoder core circuit were compared with those required to implement custom-designed versions of the convolutional encoder. If the core is described in an efficient manner, the core and the custom design should use approximately the same amount of resources. Qualitatively, the facility of adjusting core parameters was also noted.

4.2.2 Core Packing

As previously described, cores should ideally be able to interface with little or no glue code. Similarly, code designed to include cores should not require much extra code to facilitate the interface between an externally designed core and the rest of the design. The systems were designed to minimize the code needed to interface the cores. The total number of Logic Cells and Memory bits required to implement the cores in the circuit were determined and compared to the total number of Logic Cells and Memory bits required to implement the whole circuit. The goal was to determine the percentage of resources utilized to implement the cores versus the percentage wasted on connecting the different cores. Obviously, the goal is to create designs that limit the resources used to connect the cores so that more resources are available to the essential functional modules.

4.2.3 Predictable Timing

It is important that these circuits achieve consistent timing characteristics so that performance is predictable. Multiple place and routes were performed on each circuit. Furthermore, an attempt to optimize the speed of each circuit was done by using the timing-driven routing option. Although this might not achieve the maximum speed that could be obtained by manually adjusting the floorplan after the place and route, the objective of this study is not to study the tools but to observe how the tools treat the cores. Ideally, the maximum clock speed of the circuit should be approximately inversely proportional to the size of the circuit.

4.2.4 Area Usage

A core with a specific set of parameters should require the same amount of memory and logic resources independent of the overall system design and FPGA size, therefore, its performance should not degrade when the circuit using the core is implemented on a larger device. The tools should be smart enough to place the circuit so that the maximum speed of the circuit remains relatively constant or even improves due to the increased availability of resources. By examining the maximum clock frequency for a circuit and the floorplan obtained by the place and route algorithm, it should be evident if the cores are treated as modules of circuitry implemented separately in their own space.

4.3 Observations

The following describes characteristics of core design that are not quantifiable by the metrics of the previous section. They are equally important when choosing to implement a core in an SoC design or to create a core as they can drastically affect the design time and, therefore, the time-to-market of a product.

4.3.1 Implementation Language

There are numerous languages used to create Hardware Designs such as VHDL [50], Verilog [51], AHDL [52], and C++. Of these four, VHDL and Verilog are the most popular in industry today, with Verilog more common to North America and VHDL more so in Europe. Although some believe that AHDL is a language better suited to IP core development, the fact that AHDL is not an industry standard means that it does not have as large an appeal. This has resulted in even Altera's IP design group making their cores available for us in VHDL and Verilog designs as well. Since their cores are provided as encrypted netlists, the actual design language is unimportant, but when the MegaWizard asks the user to choose a design language, a header file is generated for the core in that language. Other third party IP vendors who do provide source code will often design in one language but will willingly translate the code to the other language for the customer.

It seems that neither VHDL nor Verilog has become the commercial standard for core design. Companies such as the VCX do not make any requirements as to which language should be used by core designers who make cores available on their web page. They have also failed to notice any particular pattern emerging as to core design being either mainly in Verilog or VHDL. This is probably due to the fact that IP is a relatively new market. To expand their market, the vendors may create a core in one language, offer it in both languages, and then design it in the other language when required. Another possibility is that the core is designed in one language, and immediately translated to the other upon design completion. The usage of C++ as an HDL is relatively new and has not gained acceptance as an industry standard design language. It is believed that there are currently no commercially available C++ IP cores at this time. Should C++ gain a wider acceptance, it is likely that IP vendors will also create cores for C++.

Still, it may be worthwhile to consider the development of a new HDL which would be better suited to core design. This research has discovered that VHDL is not well suited to core design in its present form. It does not support the removal of ports from a design by defining a bus width of zero. It also does not provide the ability to calculate difficult equations. This suggests that either VHDL should be updated with new constructs, as Verilog is to create Verilog-2000 [53], or that another HDL should be created.

4.3.2 Architecture Independence

Numerous FPGA architectures are presently available. The compatibility of cores among different device families is specified by the core provider. Often the only limiting factor for backwards compatibility of a core on a device is the size. Therefore, if a core is usable on a FLEX device, it should be easily placed on an APEX device, which is also from Altera. Although it should be possible to use the cores from these circuits on APEX devices, the difficulty of achieving a functional place and route with version 2000.5 of the Quartus tools made this difficult to verify.

The possibility of using Altera cores on Xilinx parts or Xilinx cores on Altera parts was examined. This was determined to be impossible due to the format of the core provided to the user. Altera's cores are licensed so that they will only compile using software provided by Altera and, therefore, cannot be used on Xilinx parts. Xilinx cores are supplied as netlists that are structured for the Xilinx FPGA architecture. Since these are significantly different, Xilinx cores are unusable on Altera parts.

Finally, some third party IP vendors such as Integrated Silicon Systems (ISS), sell IP cores to both ASIC and FPGA designers. Since ISS supplies its cores in a firm format as a targeted netlist, the cores are not directly transferrable between the two technologies. A more interesting observation is that ISS cores are available for both Xilinx and Altera FPGA designers. Although, their deliverable is a firm core, they have chosen to make their IP available for multiple types of silicon technologies.

4.3.3 Security

IP vendors have the option of providing the source code or an encrypted netlist for a core. Obviously, these two methods have different security issues. If a vendor provides the source code to the user, the main security issue is to ensure that the core is only used by that company under the specified terms of the agreement between both parties; misuse of the core would likely lead to a lawsuit. Both parties must be trusted to respect the agreement and, if there are questions of enforceability, then possible risks should evaluated.

If a vendor only provides an encrypted netlist of the core, the issue of maintaining core security is also a factor. The encryption methodology should be robust enough to prevent an individual from reconstructing the source code. Ideally, no one should be able to break the encryption, but realistically, as long as the cost of breaking the encryption is greater than that of redeveloping the actual core, the encryption is reasonably secure. Companies, such as Altera, encrypt their source code and their users must then obtain a license to fully utilize the core.

From the users viewpoint, encryption of a core can make the design process very difficult. The first problem is that the encryption normally interferes with the usage of tools from other vendors. By limiting the tools that can be used in the design process, the core may prevent the complete verification of the design. Furthermore, the user is unable to edit the code which means that there may be certain changes that would significantly improve the overall design but they cannot be implemented. Finally, the inability to access the code provides further frustration when the core is accompanied by poor documentation. Without the source code and good documentation, the core is a black box with unknown behavior.

4.3.4 Legal Issues

It is important to realize that the concept of Intellectual Property as some form of reusable hardware design is relatively unestablished in a legal sense. This often results in protracted contractual negotiations as some of the laws are not intuitive. For instance, a vendor may license a core such that the ownership of the intellectual property of the core remains with the vendor (licensor). If a core were sold, instead of licensed, then it could not be subsequently licensed for use by any other company because it would belong to the company who had purchased it. A licensing contractual arrangement leads to the situation where any modifications to a core are the property of the licenser and not the designer.

The license will often also limit the use of the core, and any derivation thereof, to a specific purpose on specified silicon. Therefore, while the system design is the intellectual property of the designer, the core and any modifications made to it are the intellectual property of the core vendor. This problem did not exist when IP was implemented in the form of chips because a chip is fixed and unchangeable. An individual could not change the implementation, algorithm or the performance of the IP as it had been burned into the silicon. Although soft cores offer the benefits of flexibility, they also present new challenges in IP protection.

4.3.5 Modular Core Design

When the convolutional encoder core was designed, the design was made as modular as possible. By doing this, it was hoped that unused sections of code could be removed so that unnecessary resources would not be wasted. This was accomplished by using IF GENERATE statements so that this extra code could be added to the basic design when necessary. Unfortunately, VHDL forces the addition of unnecessary code sometimes due to its syntax. The language will not allow the removal of ports, which can be cumbersome when a core is scalable as is the convolutional encoder core.

The other problem arises from inheritance and the passing of parameters from one level of the design to another. It is not possible to assign a variable bus size at the upmost level of the design. This obviously causes problems when the input or output bus of a core is dependent on a size parameter. The user cannot simply enter values for the constants in the entity declaration but must also change the values of the bus widths. While having the user change the values of the data buses manually solves the problem at a basic level, it creates another problem. The abstraction that the designer wishes to achieve by using cores is no longer intact.

4.3.6 Core Amalgamation Effects

Often a large amount of glue code is required between cores even when they are from the same vendor. This should not be necessary if the core designer plans well enough ahead. Realistically, some of the cores in these designs do not fit together well – even though they are often used together – so that they require numerous registers and buffers. Also, an efficient placement of core designs normally requires that the compiler intelligently remove redundant code such as signals tied high or low permanently, registers permanently enabled, etc.. While the tools available today normally perform this function, most designers would prefer that the removal of these devices occurred at a source level as opposed to curing the optimization of the circuit so that there is no possibility of failure.

4.3.7 Methods of Obtaining Cores

When cores are not parameterizable, as in the case of a PCI core, the simplest method for a user would be the ability to go to an online library/catalogue and fill out a request for the core. Upon doing so, the user could be emailed the core or allowed to proceed to a secure part of the vendor's network to download the core. There are very few concerns about being able to obtain another version of a core in this case because there are no variables which means there is only one core. The core is designed to operate as is and in no other fashion.

In the case of parameterized cores, it is preferable that the request be for the use of a core generator, such as a Finite Impulse Response (FIR) filter generator. The user wants to be able to change the parameter settings without having to request yet another core. This freedom enables the user to easily fine tune a core for the design in which it will be used. It may also allow the user to learn the operation of the core on a scaled down version of the final core, simplifying the learning process.

4.3.8 Tools for IP Cores

The tools that were available for testing these circuits using IP cores were insufficient. To provide for thorough verification of a circuit's functionality, it is crucial that input files can be used to generate the input signals and output files can be used to store the results. As previously mentioned, this was not possible for this system. The design of the core using Aldec's Active-HDL was successful and allowed the user to create a thorough testbench. Unfortunately, when this core was incorporated into the rest of the design, it had to be changed because MAX+plus II does not support constant declarations in the entity declaration which is an essential component of user-designed IP cores. This means that they cannot simply be instantiated and implemented in a system design.

Finally, Altera's new Quartus software, version 2000.5, did not provide adequate support for their own cores included in a user's design. As can be seen from this summary, the tools available do not provide adequate support for the IP core design process. To protect their IP, Altera has made it so that their IP can only be used with their tools. Since their design and synthesis tools are not the best ones available, being restricted to their usage could affect the quality of the final design.

4.3.9 Information Abstraction

When a designer chooses to use a core, as opposed to personally creating the module, the time-to-market should be shortened by the abstraction of the detail of that core. The designer would prefer to treat the core as a block, considering only the inputs and outputs. The following sections describe how the tools and documentation provided by the core vendor help to abstract this information for the designer. If the tools, and more importantly the documentation, are well specified, this greatly facilitates the job of the designer.

4.3.9.1 Core Generation Tools

The objective of using cores in SoC designs is to provide a level of information abstraction so as to simplify the use of the core. By looking at different core generation tools, it was seen that the Graphical User Interfaces (GUIs) available for Altera's Reed Solomon Compiler and Deinterleaver provided a useful level of abstraction. GUIs were also available for other cores such as FIFOs and register LPMs but these were not necessary as they did not abstract very much information.

The Viterbi core had a higher level AHDL file into which the user entered the input parameters. Although there is less abstraction to this design, a knowledge of AHDL was not required to determine how to enter the new parameters. The major benefit of the GUIs available, such as a MegaWizard, is that they abstract the language of the core itself so that it may be generated in AHDL, VHDL, or Verilog. The wizard also "remembers" the state and configuration of the core, simplifying the changing of a core in a design.

4.3.9.2 Documentation

The documentation provided by the core vendor for the designers should be similar to that of a manual for an IC. It should give a succinct description of the core as a black box, describing all the externally-seen states of the core. It should also abstract the inner workings of the core so that unnecessary details remain hidden while it is easy to amalgamate the core into a final design.

The documentation for the cores read for this thesis were poor in general. Overall, the documentation did not enable a designer to understand how to use the core quickly. In fact, it is estimated that the majority of design problems arose due to unclear, incorrect, or absent documentation. The industry attitude toward documentation appears to be that it is of secondary importance and that as long as the core is ready and technically correct, a core release can occur.

Unfortunately, without clear and succinct documentation, the actual core may be well implemented, but the designer is sorely tested to determine how to use it. This should be considered as a serious complaint – using IP cores in SoC designs is only beneficial when the time and cost incurred to use them is less than what is required to generate the core design in-house. Poor documentation is not only detrimental to the designer but to the vendor as well. If the designer is unable to resolve their questions from the documentation, they will require customer support. The larger the amount of customer support required to use a core, the less net profit will result from its sale. This thesis has illustrated that the documentation for some cores is poor enough that there is questionable savings of time and money.

4.4 Design Guidelines

Having completed the design process, a set of design guidelines were developed to reflect the methodology used to design each circuit.

1) Determine what commercially available cores suit your design. There is almost certainly more than one core available for your application. Try to ensure that consideration is given to IP cores from multiple vendors.

2) Check that any cores purchased will be usable with the tools in the design process. If this is not the case, ensure that there are tools available on the market that will support the core.

3) If a specific core is available from more than one company, investigate the pros and cons of using a core from that company. Things to look for:

- Meets the specifications for your design
- Good documentation
- Good technical support
- Good testbenches illustrating how the component functions. It should be noted that the more complex the core's behavior, the larger the number of testbenches that may be necessary to thoroughly describe its behavior.
- Good interface structure
- Any other "bonus" deliverables supplied by the company, such as a higher level model of the cores functionality.

If a core from this company has been used in a previous design, what was your experience? If it was not good and this is the only company that sells this core, consider that it might be more cost effective to have it designed in house. Remember, for cores that are not well documented, it can take almost as long to determine a core's functionality as it can to design it. Also consider that if the circuit must run at extremely high speeds, cores may not give you the performance you need, and must therefore, be thoroughly investigated.

4) Having chosen the cores to be used in your design, test them as separate entities so that there is no "mysterious" behavior. The designer must fully comprehend the behavior of the component for all valid inputs to use the core in a design.

5) Determine what glue logic is needed to connect the different modules in your system. Consider the best way to interface the core such that the glue logic is minimized without destroying the overall system throughput.

Chapter 5

Discussion of Results

The transmitter and receiver systems may be divided into three main components- the Reed Solomon module, the Interleaver module, and the Convolutional module. Each component was designed with adjustable parameters that were permuted in seven different ways. The cores were combined in unique ways to create nine different implementations of the two systems. There are three data symbol bus widths used in this study and each width has three different core configurations. The bus widths were chosen to be four, six, and eight bits. The results were studied to determine the effect of parameterization and core packing on resource usage and the maximum clock rate of these two systems.

5.1 Core Circuit Configurations

Table 5.1 provides a description of the different Reed Solomon components used to create the two systems. Although it was possible to vary the root spacing in the polynomial generator and the first root of the polynomial generator, these values were left constant at one and zero respectively. This is justified by the fact that these are suggested in the documentation as possible values for these components. Also, these values were adjusted for the rs_1 circuit and it was determined that changing the values had little to no effect on the number of LCs used to implement the circuit.

Table 5.2 describes the different Interleaver/Deinterleaver components used in the receiver and transmitter systems. The symbol width is equal to the number of bits per symbol,

Reed Solomon Circuit Labels	**Bits per Symbol**	**Symbols per Codeword**	**Check Symbols per Codeword**	**Field Polynomial**
rs_1	4	5	4	19
rs_2	4	15	4	19
rs_3	6	15	4	67
rs_4	6	19	8	67
rs_5	6	52	8	67
rs_6	8	52	8	285
rs_7	8	204	16	285

Table 5.1: Description of Reed Solomon circuits.

as indicated in column two of both Table 5.1 and 5.2. For these circuits, the symbol width the interleaver must be equal to that of the Reed Solomon component in each circuit. To ensure that the design is robust against burst errors, multiple codewords are interleaved.

For smaller codewords, four codewords are interleaved whereas for larger codewords only two codewords are interleaved. This was accomplished by always setting the number of rows to four and setting the number of columns to the number of symbols per codeword, N, for the first four circuits, and $N/2$ for the remaining three. The interleaver core also requests that the user specify the number of symbols in a codeword, the symbol width, and the estimated bit- error rate. These values are just used to provide estimates for the user as to the effectiveness of the interleaving values chosen.

The Convolutional Encoder/Viterbi Decoder components of the receiver and transmitter systems are outlined in Table 5.3. The systems were designed such that the number of coded bits ranged between two and four. The constraint length was set to be either five or seven and the traceback depth was set to be five times the constraint length plus one. The convolutional encoder core allowed the user to specify any value for the generating polynomial in the header file. The custom-design required the user to edit the actual logic equations used to generate the encoded outputs to change the generating polynomials. Obviously, both methods effect the change, but the abstraction provided by the convolutional

Interleaver/ Deinterleaver Circuit Labels	**Bits per Symbol**	**Number of Rows**	**Number of Columns**
int_1	4	4	5
int_2	4	4	15
int_3	6	4	15
int_4	6	4	19
int_5	6	4	26
int_6	8	4	26
int_7	8	4	51

Table 5.2: Description of interleaver and deinterleaver circuits.

encoder core is more user-friendly.

Viterbi decoder is designed with suggested values for the generating polynomials given the number of coded bits, N, and the constraint length, L. The symbol width must be equal to that of the interleaver. The Viterbi decoder also allows the user to select the number of bits representing each encoded bit read off the channel. Since varying voltage levels are used to represent the various bit levels, a finer resolution makes the channel more robust to noise.

5.2 System Circuit Configurations

The previous tables have described the different components used to create the two systems. Table 5.4 provides a description of the basic system created by delineating which components were used for each instantiation. These combinations were chosen to provide insight as to the effect of changing the different parameters. Implementation 8c was chosen to represent components that are typically in use today. The rs_7 and vit_7 modules are commonly used in Digital Video Broadcasting (DVB). An explanation of the compiled data results is found in the following sections.

The naming of the system followed a simple convention. If it is a transmitter, then the

Convolutional Encoder/ Viterbi Decoder Circuit Labels	**Number of Coded (Bits (N))**	**Constraint Length (L)**	**Traceback Depth**	**Symbol Width**
vit_1 GA=19, GB=29	2	5	26	4
vit_2 GA=91, GB=121	2	7	36	4
vit_3 GA=21, GB=27, GC=31	3	5	26	6
vit_4 GA=91, GB=101, GC=126	3	7	36	6
vit_5 GA=93, GB=93, GC=103, GD=115	4	7	36	8
vit_6 GA=21, GB=23, GC=27, GD=31	4	5	26	8
vit_7 GA=91, GB=121	2	7	36	8

Table 5.3: Description of convolutional encoder and Viterbi decoder circuits.

first two letters of the circuit name are “tx”, otherwise the letters “rx” were used for the receiver circuit. The digit in the name represents the width of the input data bus to the Reed Solomon Encoder. Finally, the letters ‘a’, ‘b’, and ‘c’ were used to differentiate between the three circuits implemented for that particular bus width.

Transmitter/Receiver Circuit Labels	**Reed Solomon Circuits**	**Interleaver Circuits**	**Convolutional Circuits**
tx_4a, rx_4a	rs_1	int_1	vit_1
tx_4b, rx_4b	rs_2	int_2	vit_1
tx_4c, rx_4c	rs_2	int_2	vit_2
tx_6a, rx_6a	rs_3	int_3	vit_3
tx_6b, rx_6b	rs_4	int_4	vit_4
tx_6c, rx_6c	rs_5	int_5	vit_3
tx_8a, rx_8a	rs_6	int_6	vit_5
tx_8b, rx_8b	rs_7	int_7	vit_6
tx_8c, rx_8c	rs_7	int_7	vit_7

Table 5.4: Description of transmitter and receiver circuits.

5.3 Parameterization Effects

Table 5.5 provides the results of the place and route of each of the Reed Solomon circuits described in Table 5.1. The encoder requires no memory bits for its implementation whereas the decoder does. The number of bits required is dependent on the codeword length and the number of bits per symbol. As can be seen from Table 5.1 by comparing the parameters used for the core circuits to the number of LCs required to implement the encoder circuit, the number of LCs is minimally affected by the number of data symbols in the codeword. This is sensible as the encoder is letting the data symbols pass through the encoder unchanged.

The number of LCs used to implement the encoder is affected, however by the number of bits per symbol and the number of check symbols. This is also reasonable since the encoder generates the check symbols by using from passing the data symbols through the generating polynomial. The decoder's usage of the chip's logic resources depends on the total codeword length and the number of bits per symbol. As the decoder must correct errors in both the data and check symbols, independent of the symbol width, it is therefore understandable that the entire codeword length affects the number of LCs used to implement the circuit.

Reed Solomon	Decoder		Encoder	
Circuit Name	Number of Memory Bits	Number of LCs	Number of Memory Bits	Number of LCs
rs_1	256	516	0	40
rs_2	384	533	0	45
rs_3	1152	710	0	77
rs_4	1536	1134	0	106
rs_5	2304	1164	0	106
rs_6	6144	1564	0	200
rs_7	12288	2932	0	292

Table 5.5: Memory and LC usage of Reed Solomon circuits.

Table 5.6 outlines the resource usage of the Interleaver circuits described in Table 5.2. As can be seen from the results in Table 5.6, the deinterleaver and interleaver circuits require the same number of LCs and memory bits. This is a reasonable result as the circuits are almost identical. The only difference is the order in which the data is loaded and read out of the memory. It is also interesting to note that these circuits require very little logic resources but mostly memory resources. Since memory must be allocated in blocks, even if only 1 memory bit is actually used from the block, the rest of the block becomes unusable. This means that that the numbers provided in Table 5.6 represent the total number of bits in the memory blocks used by the circuit as opposed to the actual number of bits used by the circuit.

Table 5.7 provides the resource usage of the convolutional circuits. Neither the encoder nor the Viterbi decoder require memory bits in their implementations. The encoder requires minimal LCs as it is simply a selective exclusive-or operation, but the Viterbi decoder uses a significant amount of logic to implement a trellis to provide error correction to the bit stream. As the constraint length increases, the size of the trellis increases exponentially. The Viterbi circuitry is independent of the symbol width but not of the number of coded

Interleaver Circuit Name	**Deinterleaver**		**Interleaver**	
	Number of Memory Bits	**Number of LCs**	**Number of Memory Bits**	**Number of LCs**
int_1	256	39	256	39
int_2	512	38	512	38
int_3	768	42	768	42
int_4	1536	41	1536	41
int_5	1536	41	1536	41
int_6	2048	45	2048	45
int_7	4096	47	4096	47

Table 5.6: Memory and LC usage by interleaver and deinterleaver circuits.

bits, N. By comparing the number of LCs used by Viterbi decoders of the same constraint, it can be seen that as N is increased by a bit, the decoder LC usage increases by 20 to 30 LCs. Obviously, this increase is negligible when compared to the effect of the constraint of length on the number of LCs used.

Circuit vit_5 provides an interesting result as the number of LCs required to implement the convolutional encoder core is less than the number used to create the custom-design circuit. All the rest of the convolutional encoder circuits require the same number of LCs to place both circuits. This illustrates that using an IP core, as opposed to a custom-designed circuit, does not necessarily mean that it will require more logic to implement the parameterization. The reason that there is a difference between the number of resources the two convolutional encoder circuits use is likely due to the heuristic algorithm used to synthesize the circuits. It should be noted that this was the only instance in which the core and the custom-design provided different results for the number of LCs used in the design. Furthermore, the custom-design only requires one more LC than the core which is a negligible difference.

Convolutional Circuit Name	Convolutional Encoder		Viterbi	
	Number of Memory Bits	Number of LCs	Number of Memory Bits	Number of LCs
vit_1	0	17	0	2248
vit_2	0	21	0	10991
vit_3	0	30	0	2270
vit_4	0	33	0	11012
vit_5 (1)	0	33	0	11039
vit_5 (2)	0	32	0	11039
vit_6	0	28	0	2299
vit_7	0	29	0	10991

Table 5.7: Memory and LC usage of convolutional encoder and Viterbi decoder circuits. The convolutional circuits labeled with a (1) use a custom design for the convolutional encoder circuit whereas the circuits labeled with a (2) use the convolutional encoder core circuit.

5.4 Core Packing Effects

Table 5.8 provides the memory statistics for both the transmitter and receiver circuits. By studying column four of the table, it can be seen that the transmitter is more efficient in its memory usage than the receiver for systems with four and six-bit symbol widths. However, the receiver circuits use less memory as glue logic than the transmitters when the symbol width is eight bits. This appears to result from the fact that the memory usage in the transmitter increases by a fraction due to the change of bit width, but the receiver's memory usage increases at by slightly greater than 100% when the bit width is increased.All of the systems, except for two of the receivers, tx_8b and tx_8c, have used at least 70% of the memory bits for implementing the IP cores which is within reasonable design limits. Unfortunately, it appears that as the two circuits are scaled up in size, the transmitter circuit requires a greater percentage of glue logic to interface its cores. This does not seem to hold true for the receiver circuit, however, which remains relatively constant in the amount of memory required to interface the IP cores to each other as the receiver circuits get larger.

Circuit Name	Memory Used by Cores	Total Memory Used by Circuit	Percentage of Memory Used by Cores
tx_4a	256	256	100
rx_4a	512	640	80
tx_4b	512	576	89
rx_4b	896	1216	74
tx_4c	512	576	89
rx_4c	896	1216	74
tx_6a	768	864	89
rx_6a	1920	2400	80
tx_6b	1536	1632	94
rx_6b	3072	3936	78
tx_6c	1536	1920	80
rx_6c	3840	4992	77
tx_8a	2048	2560	80
rx_8a	8192	9728	84
tx_8b	4096	6144	67
rx_8b	16384	20480	80
tx_8c	4096	6144	67
rx_8c	16384	20480	80

Table 5.8: Memory statistics for transmitter and receiver circuits.

Table 5.9 gives the LC statistics for both the transmitter and receiver circuits. Column four of the table illustrates that the receiver circuits achieve very efficient core packing and need minimal extra LCs for glue logic. Unfortunately, the core packing for the transmitter cores is not nearly as good. This is a problem partially due to the actual transmitter system design. There is overhead logic required to ensure that the system interfaces properly with the outside world.

For instance, the Reed Solomon Encoder is not equipped with signals to indicate when new data words should be supplied to the encoder. This means that a counter has to be used to keep track of the number of data symbols read in by the Encoder so that data symbols will not overwrite the check signals generated by the core. If the core had an output signal indicating when the data symbols had been read in, then there would be a method of indicating when the transmitter needed to encode a new data symbol.

5.5 Timing Results

Table 5.10 provides the results for the receiver circuits, including the maximum clock frequency and the device automatically chosen for the fitting of the circuit. As can be seen from column five of the table, circuits rx_6b, rx_8a, and rx_8c cannot be implemented on any of the Flex parts as they require too many LCs. This is due to the size of the Viterbi decoder in these circuits which has a constraint length of seven and monopolizes most of the logic resources on a device, even on the largest of parts.

It appears that the place and route tool obtains a timing analysis for circuits that do not even fit on a device. This is probably done by assuming that there is an FPGA with a FLEX 10KA architecture but is infinite in size. Obviously, this timing information is only an estimate and cannot be assumed as the maximum clock frequency if the circuit is placed on an APEX device which has a slightly different architecture. The timing data also indicates that as the circuit becomes larger and more complex, the maximum clock frequency decreases. Although circuit rx_6c appears to be an exception to this rule, the maximum clock frequencies are so close that the decrease in frequency is relatively negligible.

Since the placement and routing of the circuit is performed using a heuristic algorithm, this inconsistency in the maximum clock rate for the rx_6c circuit is reasonable. The algorithm basically searches a solution space for a minimum which represents the best possible fit for the circuit. In doing so the the search may obtain a local minimum and fail to find the actual minimum on the search space. This could result in the algorithm obtaining a relatively poor fit for the circuit, as opposed to the best fit which explains why the clock

Circuit Name	**LCs Used by Cores**	**Total LCs Used by Circuit**	**Percentage of LCs Used by Cores**
tx_4a	96	145	66
rx_4a	2803	3027	93
tx_4b	100	272	37
rx_4b	2819	3179	89
tx_4c	104	276	38
rx_4c	11562	12052	96
tx_6a	149	349	43
rx_6a	3022	3437	88
tx_6b	180	391	46
rx_6b	12187	12748	96
tx_6c	177	406	44
rx_6c	3475	3916	89
tx_8a (1)	278	528	53
tx_8a (2)	277	527	53
rx_8a	12648	13283	95
tx_8b	367	645	57
rx_8b	5278	5796	91
tx_8c	368	642	58
rx_8c	13970	14622	96

Table 5.9: LC statistics for the transmitter and receiver circuits.

frequency for the rx_6c circuit is better than expected.

Although a placement could be found on a Flex part for circuits rx_4a, rx_6b, rx_8a, and rx_8c, the timing analysis failed due to a problem with version 5 of Quartus supporting cores. The MAX+plus II software, however, was able to provide estimates for the maximum clock frequency for rx_6b, rx_8a, and rx_8c even though they did not fit on a

FLEX part. What is even more interesting is that the maximum clock frequencies for these three circuits were greater than that for the rx_4c circuit although the tools estimated the maximum clock frequency for these circuits even though they did not fit on the Flex parts. This was accomplished by assuming the basic Flex architecture, given there were sufficient logic resources to place the circuit. As can be seen from the table, the maximum clock frequency for the rx_4c circuit is significantly slower than for the other large circuits. In all probability this occurred because the circuit fit on the EPF10k250 part but used 99 percent of the logic resources, hence the faster routing resources were already used.

The desire was to place and route this circuit, as well as the other three circuits which did not fit on the FLEX parts, on APEX devices using Quartus software and to obtain a timing analysis. Presently only version 5 is available for use in this research which seems able to fit the circuits successfully on a part but unable to perform a timing analysis due to the cores. It is, however, still interesting to note that the Quartus synthesizer chose to implement the receiver circuits in a significantly different fashion than the MAX+plus II software. Much of the circuit has been implemented in the ESB bits as opposed to in the LCs. It would be interesting to know what kind of effect this has on the overall circuit timing.

Table 5.11 describes the results for the transmitter circuits using both the custom-designed convolutional encoder and the convolutional encoder core. The maximum clock frequency given in column four is for the serial clock because it determines the overall operating speed of the circuit. The timing results for these circuits are relatively consistent for circuits of a given bit size. Neither the core nor the custom-designed version consistently achieve a better maximum clock frequency. The six-bit systems run at speeds comparable to that of the eight-bit systems.

Obviously, FLEX FPGAs are designed to better support systems that operate on data that has widths of powers of two since there are eight LEs in each LAB. This is because thee six-bit systems are slowed by the problem of bit boundaries not always falling within a Logic Array Block (LAB) which has a fast routing architecture connecting it. If the six-bit systems were fit on to the architecture such that only part of the symbol were in a LAB and the rest were in another LAB, the signals would have to travel along the slower routing tracks, thus increasing the path between registered data.

Circuit Name	**Number of Memory Bits**	**Number of LCs**	**Max Clock Frequency (MHz)**	**Device**
rx_4a	640	3027	41	EPF10K100ARC240-1
rx_4b	1216	3179	37	EPF10K100ARC240-1
rx_4c	1216	12052	28	EPF10K250AGC599-1
	15488	1284	Uncalculated	EP20K100TC144-1
rx_6a	2400	3437	36	EPF10K100ARC240-1
rx_6b	3936	12748	37	EPF10K- no fit
	18688	1925	Uncalculated	EP20K100TC144-1
rx_6c	4992	3916	39	EPF10K100ARC240-1
rx_8a	9728	13283	32	EPF10K- no fit
	24064	2397	Uncalculated	EP20K100TC144-1
rx_8b	20480	5916	23	EPF10K130VGC599-2
	20480	5796	29	EPF10K250AGC599-1
rx_8c	20480	14622	22	EPF10K- no fit
	32256	3679	Uncalculated	EP20K100TC144-1

Table 5.10: Overview of place and route results for the receiver circuits.

5.6 Area Usage Results

Table 5.12 provides a summary of placing the same circuit, tx_8a, on different size FLEX10KA devices. As can be seen from the maximum serial clock frequency results, the frequency decreases as the parts increase in size. The reason for this can be seen from the floor plans in Figures 5.1, 5.2, and 5.3. As the devices get larger, instead of making a circuit placement that keeps the circuit logic relatively close, MAX+plus II spreads the logic all over chip. By increasing the length of the critical path in these larger devices, the

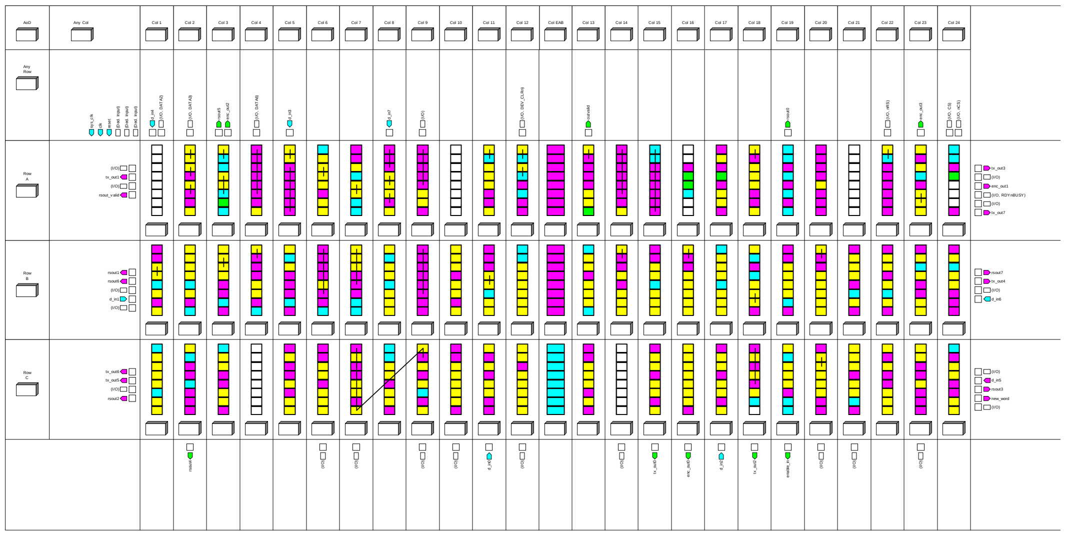

Figure 5.1: Layout of the tx_8a circuit on an F10K10 device.

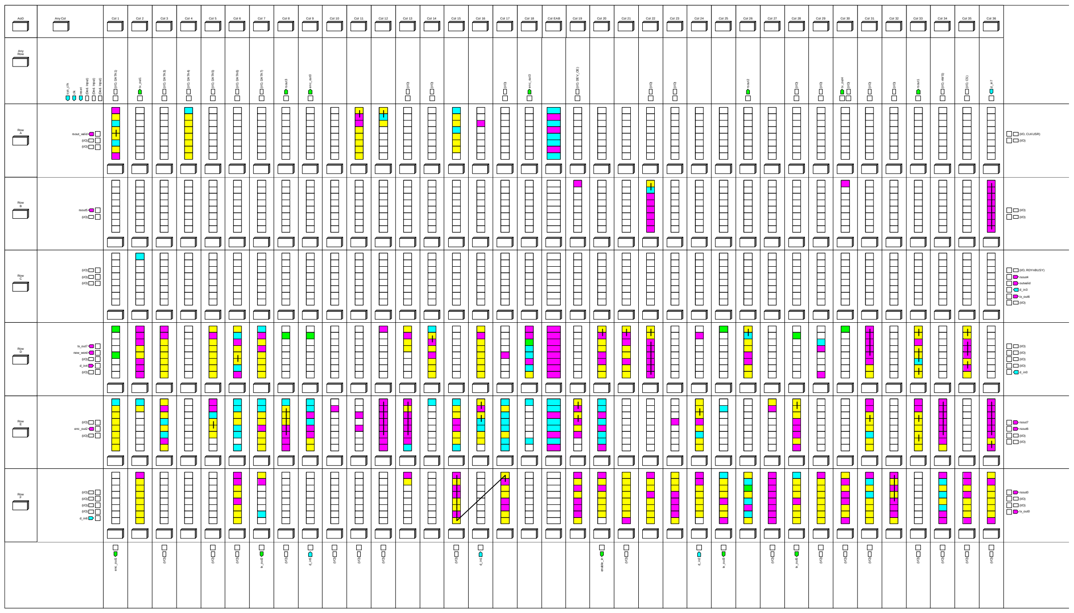

Figure 5.2: Layout of the tx_8a circuit on an F10K30 device.

maximum clock frequency decreases. This illustrates the need for good floorplanning tools to improve the overall circuit performance.

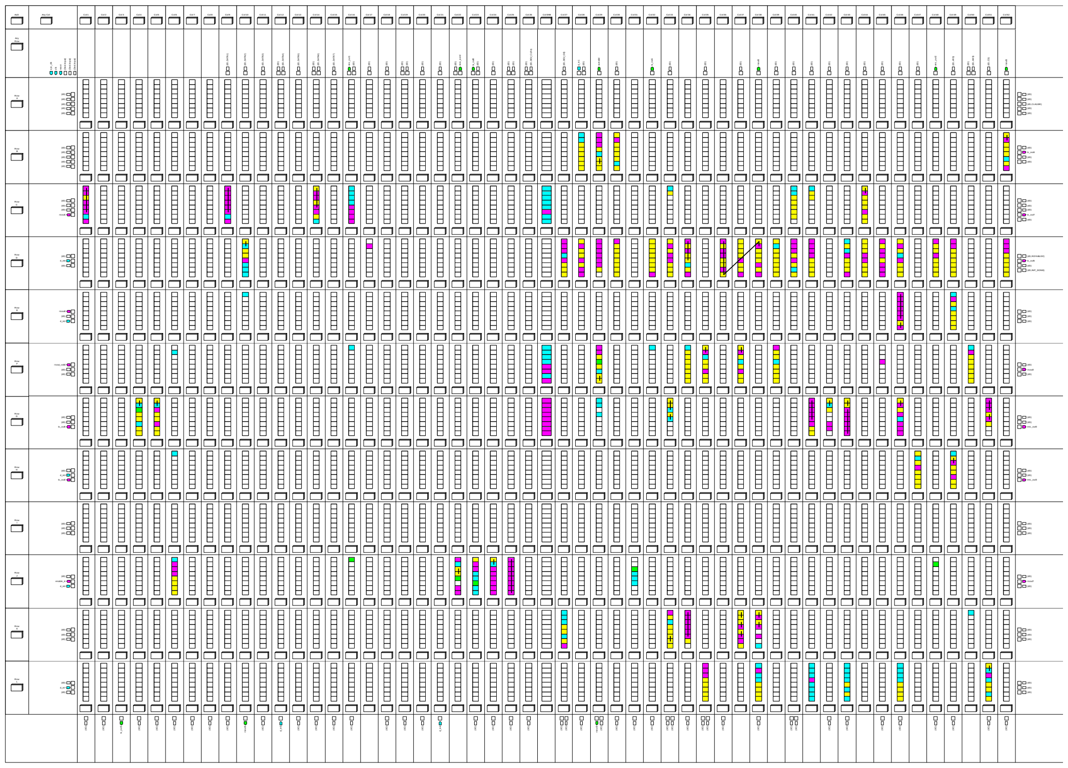

Figure 5.3: Layout of the tx_8a circuit on an F10K100 device.

Circuit Name	**Number of Memory Bits**	**Number of LCs**	**Max Serial Clock Frequency (MHz)**	**Device**
tx_4a (1)	256	145	154	EPF10K10ATC100-1
tx_4a (2)	256	145	149	EPF10K10ATC100-1
tx_4b (1)	576	272	156	EPF10K10ATC100-1
tx_4b (2)	576	272	147	EPF10K10ATC100-1
tx_4c (1)	576	276	164	EPF10K10ATC100-1
tx_4c (2)	576	276	147	EPF10K10ATC100-1
tx_6a (1)	864	349	91	EPF10K10ATC100-1
tx_6a (2)	864	349	89	EPF10K10ATC100-1
tx_6b (1)	1632	391	92	EPF10K10ATC100-1
tx_6b (2)	1632	391	101	EPF10K10ATC100-1
tx_6c (1)	1920	406	88	EPF10K10ATC100-1
tx_6c (2)	1920	406	93	EPF10K10ATC100-1
tx_8a (1)	2560	528	119	EPF10K10ATC100-1
	2560	528	99	EPF10K30ATC144-1
tx_8a (2)	2560	527	116	EPF10K10ATC100-1
	2560	527	111	EPF10K30ATC144-1
tx_8b (1)	6144	645	94	EPF10K30ATC144-1
tx_8b (2)	6144	645	87	EPF10K30ATC144-1
tx_8c (1)	6144	642	84	EPF10K30ATC144-1
tx_8c (2)	6144	642	88	EPF10K30ATC144-1

Table 5.11: Overview of place and route results for the transmitter circuits. The transmitter circuits labeled (1) use a custom design of the convolutional encoder and the circuits labeled (2) use the convolutional encoder core designed for this study.

Circuit Name	**Percent of Memory Bits Used**	**Percent of LCs Used**	**Max Serial Clock Frequency (MHz)**	**Device**
tx_8a (1)	41	91	119	EPF10K10ATC100-1
tx_8a (2)	41	91	116	EPF10K10ATC100-1
tx_8a (1)	20	30	99	EPF10K30ATC144-1
tx_8a (2)	20	30	111	EPF10K30ATC144-1
tx_8a (1)	10	10	97	EPF10K100ARC240-1
tx_8a (2)	10	10	98	EPF10K100ARC240-1
tx_8a (1)	6	4	70	EPF10K250AGC599-1
tx_8a (2)	6	4	74	EPF10K250AGC599-1

Table 5.12: Overview of place and route results for the tx_8a circuit. The transmitter circuits labeled (1) use a custom design of the convolutional encoder and the circuits labeled (2) use the convolutional encoder core designed for this study.

Chapter 6

Conclusions and Future Work

Although the desire to use IP cores for designs has been expressed for many years, it is only over the past couple of years that any inroads have been made to make this dream a reality. Presently, companies are trying to develop an infrastructure that will allow them to efficiently reuse designs that they have created as well as evaluate the quality of third party IP. The ideal future would be a market where IP cores could be regarded as the “soft” equivalent to the ICs commonly purchased from multiple vendors and used today in printed circuit board designs.

6.1 Conclusions

The issues involved with using cores in system designs have been presented throughout this thesis in two categories. The conclusions obtained through the design experience are also presented in this fashion. Overall, this research has proved that the actual IP cores available are well designed and do not seriously degrade a systems performance. However, the success of this technology in industry is dependent on the development of a good infrastructure to address the more qualitative issues involved in a system design using cores.

6.1.1 Measured Performance

The cores are well designed from a technical standpoint. They scale in size, relative to an increase in functionality, at an acceptable rate. In fact, there may be no extra cost for designing a module of functionality as a core as illustrated by the convolutional encoder. The amount of glue logic required to interface cores in these systems was low relative to the total amount of resources used by the circuits. It is not clear, however, that the Reed Solomon and interleaver cores could not have been better designed to allow a direct interface. The need to add buffering between the two seems unjustified as they are often used in combination.

The timing results achieved by these circuits were fairly predictable. They illustrate that using IP cores in these designs has not significantly altered the overall timing performance of the circuit. The convolutional encoder core actually provided better speed performance than the custom-designed circuits in many instances. The area usage of the circuits by the tools was rather disappointing, however. The core is obviously not viewed as a cohesive module that should be placed in a defined area. Instead, it is viewed the same as the rest of the circuit logic and is spread all over the chip which may be detrimental to the maximum speed at which the circuit can be clocked.

6.1.2 Experiential Concerns

From a qualitative standpoint, the problems with achieving the simple integration of IP cores into an SoC design are unfortunately numerous. For instance, vendors must worry about the security of their IP, while designers must worry about the time expended resolving legal issues, which significantly increase the time required to obtain a core. Fortunately, as the industry becomes more established these issues are being addressed.

The development of standards for IP core design and reuse is a serious concern. Many companies have already been developing standard requirements for the IP cores they design and use, but they are only in-house standards, so that industry-wide standards do not exist. The Virtual Socket Interface Alliance (VSIA) is attempting to specify interface standards to be adopted by industry to simplify the mixing of different components from different vendors by facilitating communication between the parts. It is unlikely that companies who have invested capital into developing their own in-house standards will drop them in favor of the VSIA standards, due to monetary loss. A compromise is being reached as these companies are adapting their requirements to at least include some of the standards of the VSIA.

The worst problem with using IP cores in a design, especially third party cores, is documentation. The majority of cores available do not include sufficient documentation to make the core easy to use by another designer. The core is supposed to act as a black box that hides unnecessary detail from the user. Unfortunately, the documentation is often insufficient to provide the user with an adequate description of the core requiring that the user experiment with the core's operation, which can often be very time consuming.

There are also problems with the available HDLs, which were not designed to support the parameter flexibility required for core design, and the design tools. The tools available may not allow a user to easily use an in-house core in a system design. More importantly, however, are the difficulties with verifying systems with cores if the source code is not provided. It is very challenging to verify Altera's encrypted cores because they do not have the tools available for in-depth testing. Furthermore, their cores cannot be verified functionally using any other industry simulator due to the core license. It should also be noted that by designing a core with increased flexibility so that it becomes usable in more designs, it also can become more difficult to verify. The designer will have to trust that the core has been fairly extensively verified by the vendor.

While it may not be possible to completely resolve all these issues, there are some definite improvements that may be implemented to facilitate the use of cores by designers. The first suggestion is that while an industry wide standard may not be achievable, in-house standards are mandatory. Furthermore, when a core is being designed, thought should be given as to the kind of designs in which it will be used. Could it be combined with other cores? What kind of interface would result in the smallest amount of glue logic?

The most important improvements must be made in the documentation. The documentation should not only include a listing of all the input and output signals but also a detailed description of the modes of operation. This description should include detailed timing diagrams, state machines and any other means that might clarify how the core functions. What must be essentially realized is that since cores have an added depth of flexibility, their documentation must provide a better description of the core and the interfacing requirements than if it were an IC.

6.2 Future Work

The challenges encountered in trying to design systems that used IP cores were numerous, largely due to the deficiencies in the available tools. Work is needed to develop design and verification tools that enable the user to maintain the desired level of abstraction of the functionality of the cores. It is also important that some method of standardizing cores be developed so that encrypted cores can be used in other vendor's tools.

There is also a need to improve the HDLs available by adding new constructs to support the desired abilities to parameterize cores. Another possibility worth investigating is the development of a new HDL which focuses on supporting modular design reuse.

Bibliography

[1] David August, Kurt Keutzer, Sharad Malik, Richard Newton. Programmable ASICs to reduce costs. *EE Times*, November 2000.

[2] Virtual Socket Interface Alliance- VSIA. Web Page: Fact Sheet.

[3] Crain Matsumoto and Rick Merritt. Analysis: FPGAs muscle in on ASICs' embedded turf. *EE Times*, July 2000.

[4] John Cooley. The Way of the Furby. *EE Times*, July 2000.

[5] Virtual Socket Interface Alliance- VSIA. Web Page: Architecture Document, 1997.

[6] Reusable Application-Specific Intellectual Property Developers- RAPID. Web Page: About RAPID.

[7] Laura Horsey. OpenMore opens more SoC IP doors. *EE Times*, May 2000.

[8] Richard Goering and Peter Clarke. IP99: Industry rallies to trade IP online. *EE Times*, November 1999.

[9] Synopsys and Mentor Graphics. Web Page: About.

[10] Virtual Component Exchange- VCX. Web Page: About VCX.

[11] Mark Miller, Chairman of RAPID. Web Page: VCX and RAPID Join Forces to Address Semiconductor IP Business and Legal Issues.

[12] Design & Reuse- D&R. Web Page: About D&R.

[13] Jonah McLeod. Building the IP Industry Infrastructure. *Silicon Integration Initiative*, March 1999.

[14] Harriet Harvey-Horn. IP Assessment: Issues and Strategies. *Silicon Integration Initiative*, August 1999.

[15] A.B. Kahng, J. Lach, W.H. Mangione-Smith, S. Mantik, I.L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe. Watermarking Techniques for Intellectual Property Protection. In *Proceedings of the 35th Design Automation Conference*, June 1998.

[16] Andrew B. Kanbg, Stefanus Mantik, Igor L. Markov, Miodrag Potkonjak, Paul Tucker, Huijuan Wang, and Gregory Wolfe. Robust IP Watermarking Methodologies for Physical Design. In *Proceedings of the 35th Design Automation Conference*, June 1998.

[17] John Lach, William H. Mangione-Smith, and Miodrag Potkonjak. Robust FPGA Intellectual Property Protection Through Multiple Small Watermarks. In *Proceedings of the 36th Design Automation Conference*, June 1999.

[18] Arlindo L. Oliveira. Robust Techniques For Watermarking Sequential Circuit Designs. In *Proceedings of the 36th Design Automation Conference*, June 1999.

[19] Andrew E. Caldwell, Hyun-Jin Choi, Andrew B. Kahng, Stefanu Mantik, Miodrag Potkonjak, Gang Qu, and Jennifer L. Wong. Effective Iterative Techniques for Fingerprinting Design IP. In *Proceedings of the 36th Design Automation Conference*, June 1999.

[20] Gang Qu and Miodrag Potkonjak. Fingerprinting Intellectual Property Using Constraint-Addition. In *Proceedings of the 37th Design Automation Conference*, June 2000.

[21] Darko Kirovski, David Liu, Jennifer Wong, and Miodrag Potkonjak. Forensic Engineering Techniques for VLSI CAD Tools. In *Proceedings of the 37th Design Automation Conference*, June 2000.

[22] Marcello Dalpasso, Alessandro Bogliolo, and Luca Benini. Hardware/Software IP protection. In *Proceedings of the 37th Design Automation Conference*, June 2000.

[23] Kayhan Kucukcakar. Analysis of Emerging Core-based Design Lifecycle. In *Proceedings of the International Conference on Computer-Aided Design, 1998.*, November 1998.

[24] Serafin Olcoz, Federico Ruiz, and Alfredo Gutierrez. Design Reuse Means Improving Key Business Aspects. In *Proceedings of the 1999 Fall VHDL International Users Forum (VUIF) Workshop*, October 1999.

[25] Hakan Yalcin, Mohammad Mortazavi, Robert Palermo, Cyrus Bamji, and Karem Sakallah. Functional Timing Analysis for IP Characterization. In *Proceedings of the 36th Design Automation Conference*, June 1999.

[26] Roberto Passerone, James A. Rowson, and Alberto Sangiovanni-Vincentelli. Automatic Synthesis of Interfaces between Incompatible Protocols. In *Proceedings of the 35th Design Automation Conference*, June 1998.

[27] Serafin Olcoz, Ana Castellvi, and Maria Garcia. Improving VHDL Soft-Cores Reuse with Software-like Reviews and Audits Procedures. In *Proceedings of International Verilog HDL Conference and VHDL International Users Forum (IVC/VUIF), 1998.*, March 1998.

[28] Patrick Schaumont, Radim Cmar, Serge Vernalde, Marc Engels, and Ivo Bolsens. Hardware Reuse at the Behavioural Level. In *Proceedings of the 36th Design Automation Conference*, June 1999.

[29] Tullio Cuatto, Claudio Passerone, Luciano Lavagno, Attila Jurecska, Antonino Damiano, Claudio Sansoe, and Alberto Sangiovanni-Vincentelli. A Case Study in Embedded System Design: an Engine Control Unit. In *Proceedings of the 35th Design Automation Conference*, June 1998.

[30] Jorg Henkel. A Low Power Hardware/Software Partitioning Approach for Core-based Embedded Systems. In *Proceedings of the 36th Design Automation Conference*, June 1999.

[31] T.C. Ormerod, J. Mariani, G. Spiers, L. Ball, and L. Maskill. Supporting the process of re-use in innovative design environments. In *IEE Colloquium on Intelligent Design Systems*, February 1997.

[32] Serafin Olcoz. Semiconductor Knowledge Management and Soft-Cores Reuse. In *Proceedings from the XII Symposium on Integrated Circuits and Systems Design*, September 1999.

[33] Marcello Dalpasso, Alessandro Bogliolo, and Luca Benini. Virtual Simulation of distributed IP-based designs. In *Proceedings of the 36th Design Automation Conference*, June 1999.

[34] Herbert Dawid, Klaus-Jurgen Koch, and Johannes Stahl. ADPCM Codec: From System Level Description to versatile HDL Model. In *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors, 1997.*, July 1997.

[35] H.P. Peixoto, M. F. Jacome, A. Royo, and J.C. Lopez. The Design Space Layer: Supporting Early Design Space Exploration for Core-Based Designs. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition 1999*, March 1999.

[36] C.A. Papachristou, F. Martin, and M. Nourani. Microprocessor Based Testing for Core-Based System on Chip. In *Proceedings of the 36th Design Automation Conference*, June 1999.

[37] Jacobo Riesco, Juan C. Diaz, and Pierre I. Plaza. The uPP ASIC: Design, Methodologies and Tools for a Pay Phone System-On-a-Chip Based on an ARM Core and Design Reuse. In *Proceedings of the IEEE Custom Integrated Circuits, 1999.*, May 1999.

[38] Inki Hong, Darko Kirovski, Gang Qu, Miodrag Potkonjak, and Mani B. Srivastava. Power Optimization of Variable Voltage Core-Based Systems. In *Proceedings of the 35th Design Automation Conference*, June 1998.

[39] Ing-Jer Huang and Tai-An Lu. ICEBERG: An Embedded In-circuit Emulator Synthesizer for Microcontrollers. In *Proceedings of the 36th Design Automation Conference*, June 1999.

[40] James Smith and Giovanni De Micheli. Automated Composition of Hardware Components. In *Proceedings of the 35th Design Automation Conference*, June 1998.

[41] Neil Bray. Designing for the IP supermarket. In *Proceedings from the Fall VUIF Workshop, 1999.*, October 1999.

[42] Jin-Hyuk Yang, Byoung-Woon Kim, Sang-Jun Nam, Jang-Ho Cho, Sung-Won Seo, Chaang-Ho Ryu, Young-Su Kwon, Dae-Hyun Lee, Jong-Yeol Lee, Jong-Sun Kim, Hyun-Dhong Yoon, Jae-Yeol Kim, Kun-Moo Lee, Chan-Soo Hwang, In-Hyung Kim, Jun-Sung Kim, Kwang-Il Park, Kyu-Ho Park, Yon-Hoon Lee, Seung-Ho Hwang, In-Cheol Park, and Chong-Min Kyung. MetaCore: An Application Specific DSP Development System. In *Proceedings of the 35th Design Automation Conference*, June 1998.

[43] Hoon Choi, Ju Hwan Yi, Jon-Yeol Lee, In-Cheol Park, and Chong-Min Kyung. Exploiting Intellectual Properties in ASIP Designs for Embedded DSP Software. In *Proceedings of the 36th Design Automation Conference*, June 1999.

[44] Pankaj Chauhan, Edmund M. Clarke, Yuan Lu, and Dong Wang. Verifying IP-Core based System-On-Chip Designs. In *Proceedings of the Twelfth Annual IEEE International ASIC/SOC Conference, 1999.*, September 1999.

[45] R. Burger, G. Cesana, M. Paolini, M. Turolla, S. Vercelli. A Fully Synthesizable Parameterized Viterbi Decoder. In *Proceedings of the IEEE Custom Integrated Circuits Conference, 1999.*, May 1999.

[46] Daniel Geist, Giora Biran, Tamara Arons, Michael Slavkin, Yvgeny Nustov, Monica Farkas, Karen Holtz, Andy Long, Dave King, and Steve Barret. A Methodology For the Verification of a ”System on Chip”. In *Proceedings of the 36th Design Automation Conference*, June 1999.

[47] Yervant Zorian. System-Chip Test Strategies. In *Proceedings of the 35th Design Automation Conference*, June 1998.

[48] Indradeep Ghosh, Sujit Dey, and Niraj K. Jha. A Fast and Low Cost Testing Techniques for Core-based System-on-Chip. In *Proceedings of the 35th Design Automation Conference*, June 1998.

[49] Jr G. David Forney. The Viterbi Algorithm. *Proceedings of the IEEE*, 61(3):268 – 278, March 1973.

[50] *1076-93 IEEE Standard VHDL Language Reference Manual*. New York, N.Y., U.S.A., September 1993.

[51] *1364-95 IEEE Standard Verilog Language Reference Manual*. New York, N.Y., U.S.A., September 1995.

[52] *MAX+PLUSII: AHDL*. San Jose, CA, U.S.A, November 1995.

[53] Sutherland HDL, Inc. Web Page:Verilog-2000 Information Page.