THE UNIVERSITY OF CALGARY

Implementing Digital Signal Processing Algorithms Using Serial Arithmetic

by

P.J.W. Graumann

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER
ENGINEERING

CALGARY, ALBERTA
SEPTEMBER, 1996

Abstract

This thesis describes a set of computer aided design tools, implemented by the author, which can be used to shorten the design time of hard-wired digital signal processing systems. These tools convert a bit-serial or digit-serial register transfer level circuit, described in DFIRST, into a gate level technology specific implementation. The DSIM simulator used to perform design rule checks, serial timing alignment checks and circuit simulations on DFIRST netlists is introduced. The TRANS hardware compiler converts the DFIRST primitives to generic gate level implementations and then applies optimizations to the obtain a smaller/faster implementation in the target technology. Several digital filters are implemented using DFIRST and TRANS and the different TRANS optimizations are evaluated using these designs. Finally the effectiveness of TRANS optimizations, and the quality of solutions generated for different digit-width filters are presented and discussed.

Acknowledgements

I am grateful to Micronet, Dr. Turner, and the Electrical Engineering Department for their financial support during this research. I would also like to thank Dr. Turner for his patience during the slow times and his good advice in the tough spots. I would escpecially like to thank the numerous graduate and summer students who implemented circuits using these tools. There were a lot of questions to answer, software bugs to exterminate and late nights, but most of these projects were successful in the end and the results of this research are the better for their efforts.

Dedication

For my Beloved wife Koreen.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The use of Digital Signal Processing (DSP) Integrated Circuits (ICs) has increased dramatically. They are used in everything from cellular phones to microwave ovens to CD players. The operations performed by these devices increases in complexity as IC geometries shrink and the need for faster/smaller processing devices increases. This increased complexity results in larger ICs and longer design times to create working products from specifications. In order to shorten the design cycle and hence reduce the time to market for a new device, designers must rely on more and more sophisticated and capable Computer Aided Design (CAD) tools. This thesis describes a CAD tool, implemented by the author, which can be used to reduce the time required to convert a DSP IC specification to a digital circuit implementation.

1.1 Digital Signal Processing

A signal is defined as a physical quantity which conveys information [OS89]. An analog signal is one in which the independent variable such as time, and the dependent variable such as amplitude can take on a continuum of values. In these systems time is going forward and the amplitude contains some information to be processed. In a discrete time system, time can only take a discrete set of values and the amplitude information is continuous in nature. In a digital signal both time and amplitude are discrete in nature. A digital signal can be represented by a sequence of finite

precision, or quantized, numbers representing the information in a signal at discrete intervals of time.

The transformation of one signal to another signal is defined as signal processing. During this transformation unwanted portions of the first signal may be removed or information may be added to create a desired output signal. Digital Signal Processing (DSP) is performed using simple computational blocks such as addition, multiplication, conditionals and storage elements. The types and interconnections of these processing elements define the processing algorithm and control the behaviour of the processing system.

1.2 Digital Signal Processor Implementations

1.2.1 Programmable Devices

General purpose Digital Signal Processing devices such as the TMS320 [Ins88], the ADSP2100 [Dev89] and the Motorola 56001 [Mot89] have been used extensively to perform DSP operations. Their programmable nature makes them easy to use and re-use in the face of ever changing specifications. Programmable devices generally contain one multiplier, one adder and a bit shifting unit as well as a number of registers and internal memory. These resources make programmable DSPs able to perform most signal processing applications. The wide use of these devices also means that silicon implementation technology will be constantly upgraded, resulting in ever faster and larger versions in the same processor family. The large numbers of devices which are fabricated also reduces the per unit cost, resulting in lower monetary costs for the user.

While the programmable nature of these devices makes them easy to use it also results in reduced algorithm security. The processing application must be stored in a memory device of some sort which can be easily copied. Also, because these devices are so flexible they may not be particularly well suited to a specific application. The programmable nature adds some overhead to circuit area and the fixed nature of the resources may not be optimal for many algorithms. This can mean slow performance or larger system cost.

1.2.2 Custom Implementations

Digital Signal Processing applications may also be implemented using custom integrated circuits. There are three different general forms of these devices: full custom, semi-custom gate arrays and Field Programmable Gate Arrays (FPGAs). In full custom the designer has complete control over the placement, sizing and interconnect of every transistor constituting a given design. Full custom devices require the longest design time but yield the most efficient solutions in terms of area, throughput or cost (given sufficient quantities). The complexity of the design process for full custom devices makes re-design and re-fabrication very expensive, forcing the designer(s) to get the product right the first time or face long delays and higher costs. The implementation time from specification to working ICs for these devices can range from several months to a year.

In semi-custom integrated circuits, or Mask Programmable Gate Arrays (MPGAs), the transistor patterns are fixed. The designer only has control over the interconnection of groups of transistors as logic and the interconnection of these logic elements. This results in a smaller design time but also results in slower, larger

circuits as compared to a full custom implementation. The fabrication time for these devices is shorter than that of full custom devices and can be several weeks to months. The re-design time for these components is much less than that for full custom devices but significant effort is still needed to change the circuit and re-fabricate the final device.

Field programmable gate arrays are similar to MPGAs except that the interconnection wires are programmable in the field instead of requiring a fabrication facility to place the interconnects. These devices can be classified into two basic types. one time programmable such as ACTEL [ACT89] FPGAs and n time programmable such as XILINX [XIL94] FPGAs. These devices lead to implementations which are relatively large and slow and not particularly well suited to high throughput or high volume applications. They are well suited for proto-typing and in situations where the application requirements change over time. The availability of these devices has led to novel applications in which a single FPGA is re-programmed on the fly to perform different portions of a single large application.

1.2.3 Integrated Circuit Efficiency

For any application there may be several possible IC implementations. The efficiency of different implementations is judged in terms of the amount of circuit area required. the time required to compute the algorithm and the power consumption of the circuit. One measure used to evaluate the efficiency of an IC implementation is the Area-Time (AT) product [HC90] which equally weights both area and time.

The area of an implementation is the total area required for processing blocks. routing resources and I/O pads. This measure may be a simple cell count of tech-

nology specific elements required for the implementation. The time of an implementation is made up of two parts, the critical path and the number of clock cycles required to complete one computation of the implemented algorithm.

The critical path of a synchronous circuit dictates the maximum clock rate at which the circuit will correctly function. The critical path is the maximum D-type flip-flop (DFF) to D-type flip-flop logic delay time. This logic delay is made up of logic element propagation delays, routing delays and set-up and hold times on the DFFs. The logic element delay for each component is dependent on the output loading for that element. A higher number of loads means that a higher total current is required from the driving point, also each load and wiring element adds capacitance to the output of any element. The combination of this load current requirement and the load capacitance dictates the time required to drive a node to the required state.

The AT product of IC is the area (cell count, mm^2) multiplied by the critical path length (time) multiplied by the number of clock cycles to process one input sample.

1.3 Architectures

The architecture of any system defines a set of rules or principles which guide the design and functionality of anything created using that architecture. For instance Roman architecture brings to mind columns of stone and archways while. Egyptian architecture brings to mind sand-stone blocks and hieroglyphs. In a similar way the architecture of a digital signal processor can be defined. In the DSP domain the parameters of the architecture include the number format used to store a number.

the word format used in data transfers, the number of bits used to represent each word and the number, type and interconnection of computing resources.

1.3.1 Number Formats

The number format refers to the particular digital represenation of numbers used by a given DSP architecture. The number format dictates the operations necessary to perform arithmetic operations and causes some arithmetic operations to be simpler at the expense of causing other operations to be more difficult to perform. The number format, together with the number of bits used to represent a value, also controls the range of values which can be represented.

In most number formats the value of a number is broken down into a series of digits, each digit taking on a range of values, having a weighting factor associated with it according to the digit significance. In the decimal representation each digit takes on one of 10 values and the weight for each digit is 10^n, where n is the digit number (starting from zero). In the computer world a digit weighting of 10 is not practical so a binary number system is used. Here each digit takes on one of two values which are commonly referred to as 1 and 0 or high and low. A common number format for the representation of binary number is Two's Complement (TC).

In TC the weighting for each binary digit is 2^n where n is the digit number, with the Most Significant Bit (MSB) having a negative weighting. The decimal value of an N bit TC number is given in equation 1.1. An N bit TC number can represent numbers from -2^{n-1} to $2^{n-1} - 1$.

$$x = -2^{n-1} * j_{n-1} + \sum_{i=0}^{n-2} j_i * 2^i, j_i = 1.0 \quad (1.1)$$

Another number format is the Canonic Signed Digit Format (CSD) [LEL91]. Each digit of a CSD number can take on one of three different values 1. -1 and 0. so this number format is a ternary system. Since computers are by their nature binary. two binary bits are required to represent each single bit of a CSD number. This results in poor storage properties for CSD numbers but for representing fixed coefficients this number format has some advantages over other systems. Since several redundant forms exist for each number. the designer may choose the most suitable one for a given application. resulting in reduced area or increased speed. The value of a CSD number can be calculated using equation 1.2.

$$x = \sum_{i=0}^{n-1} j_i * 2^i, j_i = 1, -1.0 \quad (1.2)$$

Another number format commonly used in computer systems is floating point numbers. Here each number is broken into an exponent (exp) and a mantissa (man). The value of the floating point number is $man * 2^{exp}$. Both the mantissa and the exponent may be positive or negative resulting in a dramatic increase in dynamic range over a fixed point system. The number of bits used to represent the mantissa and the exponent are chosen to obtain the desired accuracy and dynamic range.

Other number systems such as the Residue Number System (RNS) [SJJT86]. Logarithmic Number System (LNS) [Lew93], and the Symmetric Level Index (SLI) [CT88] are used in computing machines. Each of these number systems has unique properties which can be used to combat shortcomings in other number systems such

as long carry paths. expensive multiplication and division, or small dynamic ranges. While these number systems are useful. the number systems used in this thesis are TC and CSD. TC and CSD are chosen for their simple integer number representation and their relatively small operator size.

1.3.2 Word Formats

Another architecture parameter of a DSP system is the format in which each word of data is stored or transmitted from one component to another. The most common format. used in most computers and DSP chips. is full parallel. In this format the N bits of a data word are transmitted in parallel on N different wires. during one cycle of the system. The primary advantage of the parallel format is the speed of transmission. one data word per system cycle on each data bus. However the parallel transmission of data leads to several disadvantages.

The N bit result of each operator is also computed in parallel resulting in separate processing elements for each bit of the output data word. For some arithmetic operations information from the generation of one bit of output must be used in the computation of the next bit of output. This carrying or rippling of information from one computation to the next leads to long computation times for the completion of one parallel output word. This computation time or propagation delay can be reduced by using methods such as carry look ahead [Ann86] which shortens time to calculate the full parallel result but also results in larger circuit area for the computational unit. Another method to shorten the propagation delay through an operator is pipelining the operator so that a result is only partially completed on each clock cycle. This results in shorter propagation delays but requires more clock

cycles to generate the full result.

The N bits of a parallel data word also lead to large data buses which are difficult to route and consume a large amount of integrated circuit area. This large routing requirement often results in large pin counts for an integrated circuit and low utilizations for semi-custom devices in which the routing resources cannot be tuned to meet the requirements of a particular design.

Bit-serial architectures have been proposed to create high performance, low cost VLSI implementations of DSP applications [DR85, Joh92, NT91, Erc84]. In bit-serial an N bit data word is transmitted on a single wire, one data bit per clock cycle. The word can be transmitted in the Most Significant Bit (MSB) first format [Erc84] or in the Least Significant Bit (LSB) first format [DR85, Par91, PM89]. The MSB first format is most suitable for operations such as division, square-root and sorting since these operations are naturally performed from MSB to LSB. The LSB first format is best suited for addition and multiplication since information is propagated from LSB to MSB in these operations. In most bit-serial systems only one of these formats is used at the periphery of each computational block but may be converted to the appropriate form within each operator.

Another advantage of the bit-serial architecture is the reduced routing requirements. Communication between operators is done using only a single wire, while in a parallel architecture N wires are needed. This results is less area overhead for routing, higher design routability and utilization for semi-custom technologies and reduced pincounts for integrated circuits designed using the bit-serial architecture.

Bit-serial operators are N times smaller in size than the parallel version of the same operator. The penalty for this size advantage is the processing time, bit serial

operators are N times slower than a parallel operator (as defined without carry logic). If the two systems were operating at the same clock rate they would be equally efficient in terms of area-time (AT) product. However the propagation delay through a bit-serial component is 1/Nth of that for the parallel component, so the maximum clock rate permitted in a bit-serial system is higher than in a parallel system. Thus when the maximum clock rate is considered bit-serial systems have a better AT product than parallel systems [HC90].

Digit-serial [HC90] is a word format in which each fixed precision data word is transmitted on W (Digit-Width) different wires. the total word being divided into N/W=P separate digits. For many architectures there must be an integer number of digits in each data word. this restriction is not necessary but results in simpler control structures for the overall system. As W approaches full parallel the routing resources. operator size and latency increase. while the number of clock cycles required to complete one full computation decreases.

1.4 Synthesis

Over the last few decades rapid developments in integrated circuit technology have made possible the integration of larger and larger electronic systems. In order to support this growth in complexity new design methodologies and more sophisticated Computer Aided Design (CAD) software tools have been required. Initially this CAD software focused on design verification through simulation. but starting in the late 1970's CAD software began to take over some of the design tasks traditionally done by hand. The various types of automation and the resulting effects on design

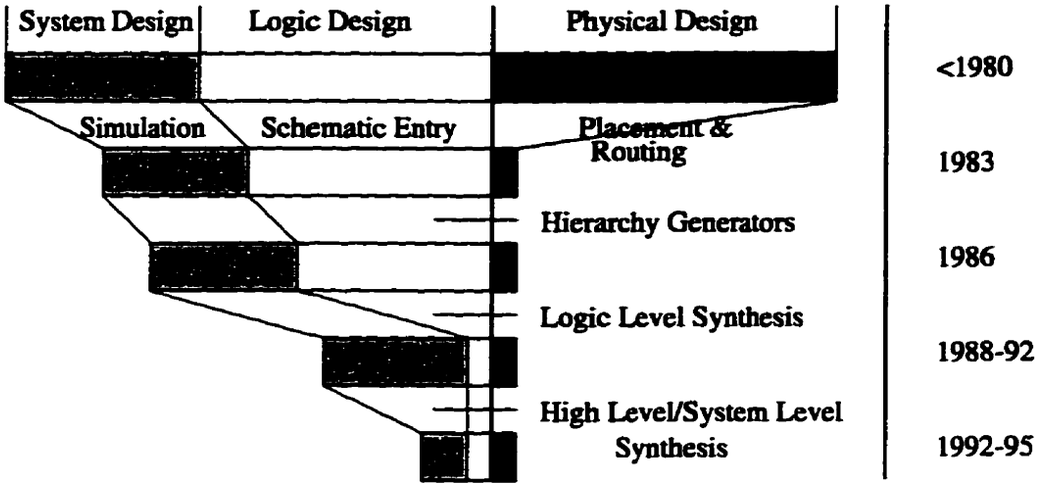


Figure 1.1: Time Line

time are shown in figure 1.1 taken from [MLD92].

Over the years design automation advancements such as automatic placement and routing, hierarchy generators and logic level synthesis have reduced design time considerably. At present the next generation of automation software is concerned with high-level or system level synthesis. These CAD tools will automatically convert a high level description of an algorithm or even a specification of an algorithm to a final IC implementation.

The use of high level or system level synthesis results in shorter design cycles and reduced time to market for integrated circuits. This occurs because of increased designer efficiency and fewer design errors since many of the error-prone steps of the design have been replaced by automated tools.

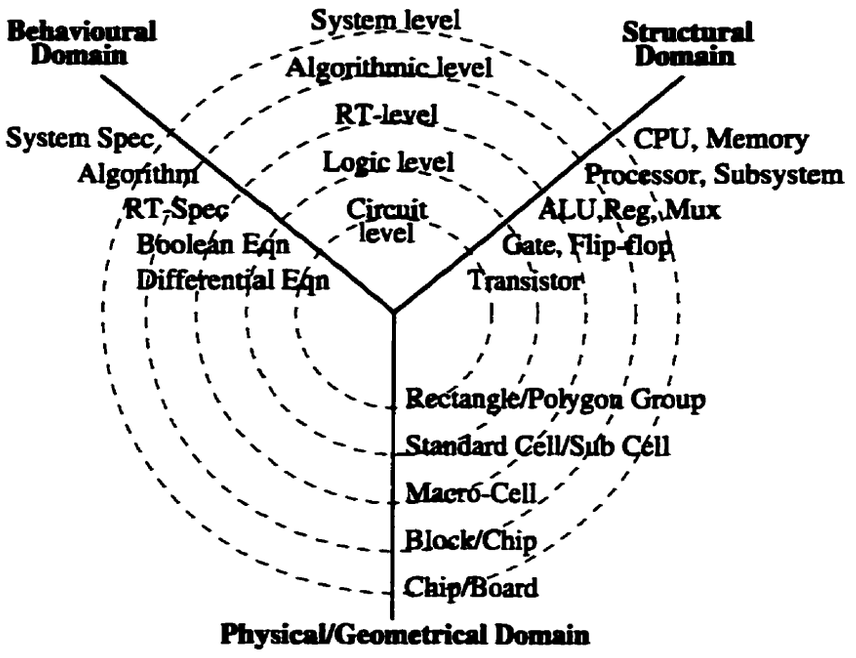


Figure 1.2: Y-chart

1.4.1 Y-Chart

When discussing synthesis issues it is useful to refer to a Y-chart introduced by Gajski and Kuhn in 1983 [GK83] as shown in figure 1.2. In a Y-chart, a design is represented in three different domains, the behavioural, the structural and the physical domains. The behavioural domain is used to describe the behaviour of a system without any notion of how the behaviour is implemented. In the structural domain a circuit is represented by a hierarchy of functional elements and their interconnections. In the physical domain a circuit becomes a layout without any reference to functionality.

The various rings of this chart indicate different levels of abstraction for the circuit with the outermost ring being the most abstract and the innermost ring being the most specific. The innermost level is referred to as the circuit level, at this level of abstraction the structural elements are transistors, resistors and capacitors. The

behaviour of these elements becomes a set of differential equations relating currents and voltages for each component. The next level of abstraction is referred to as the logic level, in this level the entire circuit is composed of collections of transistors known as gates. The behavioural representation of this information is a set of boolean equations. The voltages and currents at the circuit level have been abstracted to the logical values true and false or 1 and 0.

At the next level of abstraction the logical values have been grouped into words of data. The structural elements which make up this level are collections of gates such as adders, registers, Arithmetic Logic Units and multiplexers. This level of abstraction is referred to as the Register Transfer Level (RTL).

The next level of abstraction is referred to as the algorithmic level. At this point the behavioural description of a circuit is an algorithm or sequence of operations required to perform a given task. In the structural domain this corresponds to a collection of RTL components to form a processor or subsystem.

The highest level of abstraction is known as the system level. At this point the behaviour of a system is described only in terms of functionality. No notion of implementation is present. This corresponds to the complete system in the structural domain.

In the physical domain the various levels of abstraction correspond to ever larger polygons or blocks finally resulting in complete integrated circuits or connections of integrated circuits as on a Printed Circuit Board (PCB).

Y-charts can be used to define the various information conversions which may be required during a design, as shown in figure 1.3. The transitions on one axis of this chart are defined as refinement, abstraction and optimization. Any transition from

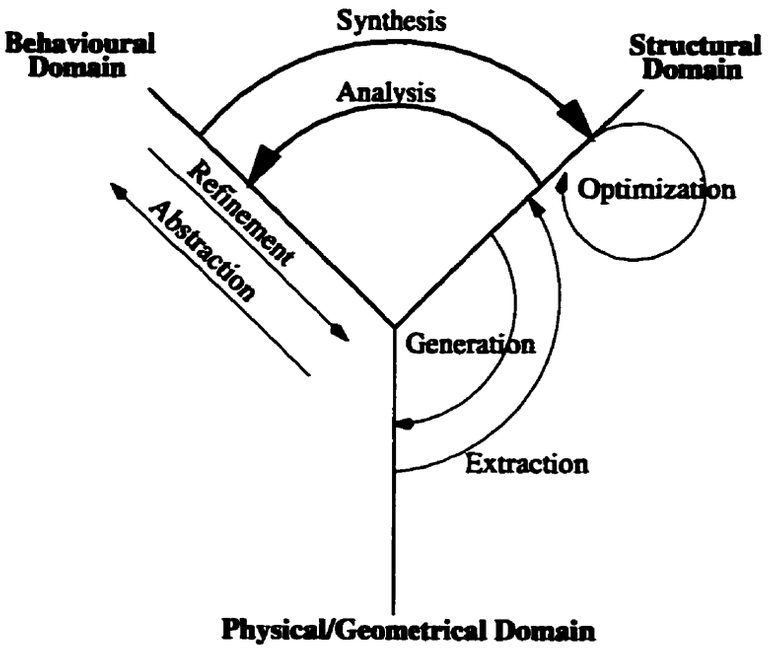


Figure 1.3: Y-chart Transformations

the periphery of this chart to the interior. on any axis. is the process of refinement. and the reverse transition is abstraction. An arc whose head and tail are at the same level of abstraction describes an optimization. During an optimization the basic functionality remains the same but the design has been improved with respect to some cost factor such as speed or area.

The transition between different axes of this chart are defined as synthesis. analysis. generation and extraction. In synthesis a behavioural description is converted to the structural domain. this defines how a behaviour is to be implemented at any level of abstraction. The reverse process. analysis. converts structural information back to a behaviour. Analysis is generally used for design verification. The process of creating a layout from a structural description is defined as generation and the reverse process is an extraction. Extraction is also used during verification to examine

the effects of routing delays and signal loading on the performance of the system.

1.4.2 Silicon Compilation

The ultimate goal of a synthesis system is to convert a system level specification in the behavioural domain to the lowest level representation in the physical domain. Any software tool which can perform this conversion can be called a silicon compiler. In practice this task can not be carried out in a single step but is broken down into several smaller synthesis. optimization and generation steps. There are several different ways to break this design process down into steps, one possible method is described below.

The first step in this process is system level synthesis. This process converts a system level specification to a set of algorithmically defined subprocessing modules. On the Y-chart this operation combines a behavioural refinement and a synthesis operation. Each of these subprocessing modules executes in parallel to perform the desired algorithm. The input specification consists of a function to perform, such as a digital filter transfer function or a computer instruction set and a set of constraints on the solution such as the desired speed, size and power consumption for the final solution.

The next step in the design process is to convert the algorithmic description of the system to an interconnection of realizable processing elements or high level synthesis. On the Y-chart this operation may combine a refinement in either the behavioural domain or the structural domain with some synthesis. The input to this stage is algorithms defined in terms of processing elements such as adders, multipliers, control structures such as branches and loops and storage elements. The output from

this stage of synthesis is a structural Register Transfer Level (RTL) design containing adders, multipliers, control structures and registers. This RTL level design may be a shared resource implementation where each operator within the design may perform two or more different operations during the algorithm computations. This synthesis step is often broken down into three major portions which are resource allocation, scheduling and resource assignment.

In resource allocation a set of functional units which will perform the required processing steps is selected. The resource assignment phase assigns each algorithm operation to one functional block made available by the resource allocation operation. The scheduling step assigns time steps to each operation on each functional resource. This synthesis operation has been addressed by several software programs, such as HAL (Force Directed) [PK87], SPAID [HE89], MAHA [PPM86], SE (Simulated Evolution) [LM90], SAVAGE [ND90], BITSYN [NT91] and SNAFU [Joh92]. Each of these programs searches the design space and attempts to find the most suitable design, given the user constraints and an estimate of area-time characteristics of each function unit available to the synthesis system.

The RTL description must now be converted to a gate level description using RTL synthesis. This process converts the functional units (behavioural/structural) which have been allocated and scheduled to gate level implementations (structural) containing simple gates and storage elements. In doing this, some knowledge of the implementation architecture is required in order to generate the most effective implementation possible.

Finally the logical descriptions are converted to collections of gates and storage elements, representing the cells or primitives available within the final implementa-

tion technology. The objective here is to obtain the most appropriate technology specific representation for each logical operation, minimizing area and/or time delay for each block. This step can be carried out early if the RTL description is converted directly to a technology dependent format. After the circuit has been mapped to technology specific cells the design is placed and routed. This operation attempts to find the optimal placement for each cell of the design which will minimize the routing area and delay for each interconnection in the circuit. Finally all the interconnections in the design are routed using the available area or routing resources in the device. The design is now ready for fabrication as required by the implementation technology.

1.4.3 Validation

The design of complex digital circuits is inherently an error prone process. Even with the use of CAD tools the resulting circuits are not guaranteed to be functionally correct. In order to deal with this problem it is important to use validation software which checks the results of a design step to make sure the circuit functions as intended. There are basically two different methods of validation, formal methods and simulation.

In formal methods the transformations applied at each stage are proven to be correct and maintain functionality. If all circuit transformations are proven to be error free then the final circuit will function as specified. In formal methods it is critical to correctly specify the operation of the system and to be certain that the proofs are complete.

In simulation a set of test inputs or test vectors are applied to the input(s)

of a circuit. the outputs are examined against the expected results and if they are consistent then the circuit is functioning correctly. The difficulty here is in identifying the correct set of test vectors which will exercise all circuit components to reveal any flaws. In simulation it is critical to understand input/output signal requirements from a circuit under test. any deviation between this understanding and the real world circuit requirements will render the simulation invalid.

Just as there are several levels of abstraction for a design there are also several levels of simulation and design verification. As each level of synthesis or refinement is applied. the results are checked with an ever more detailed simulator or formal proof. requiring longer and longer run times as the design is refined.

1.5 Research Goals

The objective of this research is to generate a CAD tool which will automatically convert an RTL behavioural description of a DSP application to a structural digital logic circuit containing logic elements. D-type flip-flops and technology specific elements. The target implementation technologies will include gate arrays. FPGAs and full custom implementations. The CAD tool should be flexible enough to support new architectures with a minimum of changes to the CAD tool itself. The target applications for this compiler are low to medium (up to 2 MHz) throughput rate digital filters and other DSP applications which are dominated by additions and multiplications. For the input/output specifications an architecture which utilizes a bit-serial or digit-serial (reduced routing), TC fixed point word traveling in the LSB first format (additions and multiplications) will be used.

The compiler will convert a RTL behavioural description, defined in the DFIRST language which is an extension of the bit-serial language FIRST [DR85], into a gate level structural description. This process is performed by synthesizing the behavioural RTL description into a structural gate level description. A series of appropriate optimizations are then applied to obtain a technology specific gate level implementation for the DSP application.

The output of this compiler must be easily retargetable to deal with the ever expanding set of new technologies and new data formats. In addition the circuit generated should be reasonably optimal for the chosen implementation technology to justify the use of the compiler. The final generation stages of the overall design procedure, placement and routing, will be performed by vendor supplied software tools for each implementation technology.

In order to reduce design time for the final implementation several analysis steps should be used to verify the correctness of each refinement or optimization step. A separate simulation program, DSIM (Dfirst SIMulator), will be created in order to perform RTL simulations, presently available logic simulators will be used to perform unit delay gate level simulations. Finally, extraction software supplied by the implementation technology vendor can be used to extract the wiring delays for final timing verification.

1.6 Overview

In the following chapter the DFIRST architecture specifics as well as language syntax are presented. In addition the DSIM RTL simulator for the DFIRST language

is discussed and the input/output format is presented. In chapter 3 the RTL hardware elements which make up the DFIRST language are presented. In particular the architecture for DFIRST adders. multiplers. bit shifters and format conversion operators is given.

In chapter 4 the TRANS gate compiler which can convert DFIRST to technology specific gate level implementations is presented. In particular the set of refinements and optimizations which can be applied to the circuit in order to obtain a smaller. more efficient design is discussed. In chapter 5 several examples which have been generated and tested using DFIRST. DSIM and TRANS are given. Finally in chapter 6 some conclusions on the DFIRST language and TRANS are presented. In addition some possible avenues for future research are explored.

Chapter 2

DFIRST Language and Simulator

This chapter presents the DFIRST register transfer level digit-serial hardware description language. The DFIRST data word format and control signals are discussed and the DFIRST language syntax is presented, including operator instantiation, hierarchy, signal declarations, chip to chip communication, and constants. In addition the DSIM simulator is discussed including input/output data formatting and simulation error reporting.

2.1 DFIRST Architecture

In a binary data DSP environment, operations are performed on N bit words of digital data. This data can be transmitted in a number of formats. The most common format is bit parallel, in which all N bits of the word are transmitted simultaneously on N different wires. The disadvantages of Parallel architectures are that large amounts of chip area are required for operator implementation and routing and that parallel operators have long propagation paths [1] leading to a reduced maximum operating clock frequency . Alternatives to word parallel data transmission are bit-serial or digit-serial formats. In bit-serial the N bits of a data word are transmitted on a single wire in N clock cycles. This leads to operators which are typically N times smaller than parallel versions of the same operator but require N times more

[1] Look-ahead-carry techniques lead to reduced propagation delays but require more hardware.

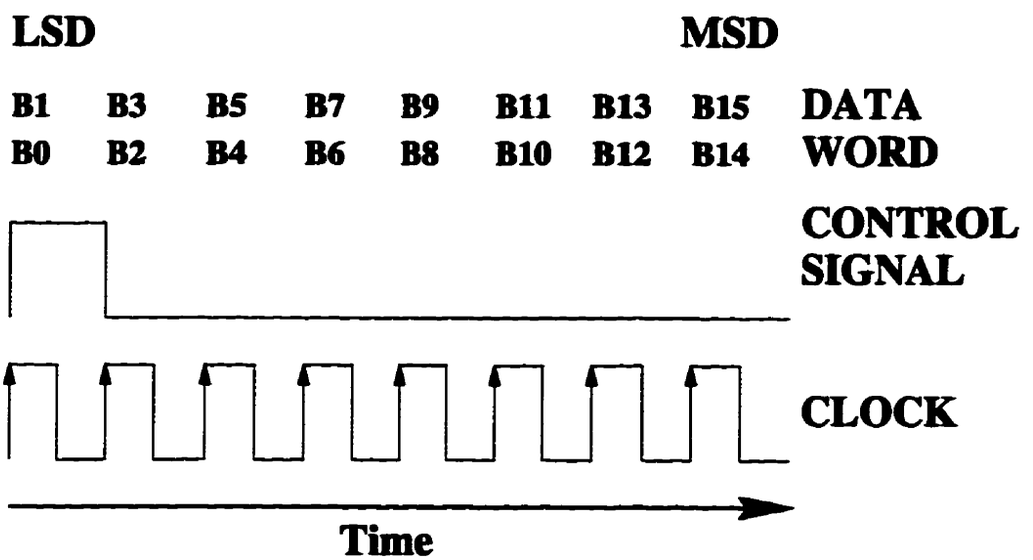


Figure 2.1: Sample DFIRST DataWord

clock cycles to complete the transmission of one data word.

In digit-serial architectures the N bit data word is divided into P separate digits. each W bits wide. For the DFIRST architecture. N must be equal to W*P. That is. there must be an integer number of digits in each data word. This restriction is not required in general. but the control circuitry is much simpler when all data words are divided into an integer number of digits. Since the data for each word takes several clock cycles to propagate along a given data path. a word framing control signal indicating some fixed point in the data word is needed. In the DFIRST architecture a control signal indicating the Least Significant Digit (LSD) time of the data word is used. as shown in figure 2.1. For some operations in DFIRST a Most Significant Digit (MSD) indicating signal may also be required to completely frame a given data word.

Each operator within DFIRST is pipelined at the digit level, resulting in a short

propagation delay through any logic elements and a higher potential operating frequency for the overall system. Each operator has a latency(L) which is the number of bit or digit clock cycles required to generate the LSD of the output signal(s), after the arrival of the input signals. The LSD of each input to an operator must arrive and be valid during the same clock cycle in order to assure correct operation.

The iteration time for a serial algorithm is defined as Q, the number of clock cycles required for one complete iteration of the algorithm. Some pipelining latency may occur between the arrival of the first input and the completion of the first output signal but the system can accept a new input every Q clock cycles. The Q of an algorithm is dictated by the minimum number of clock cycles required to update all internal states in a recursive system. If all of the operations within the DSP algorithm are implemented on dedicated resources then the resulting serial circuit will exhibit the minimum Q possible for that system with the given operator set.

In practice a more area efficient implementation can be obtained by sharing physical components between the various operations within an algorithm. This requires the multiplexing of large components such as multipliers and dividers, and results in a larger iteration time (Q) but can result in significant area savings. For some applications the optimal area-time product measure exists within a shared resource environment [NT91, Joh92, Nag91].

A generic DFIRST operator is shown in figure 2.2. Each operator requires several input and output signals which may be bit-serial, digit-serial or parallel depending on the particular operator. As well, several parameters may be required to specify the exact nature of the operator, these parameters may include for example digit width, data wordlength, coefficient wordlength and latency. Each primitive will

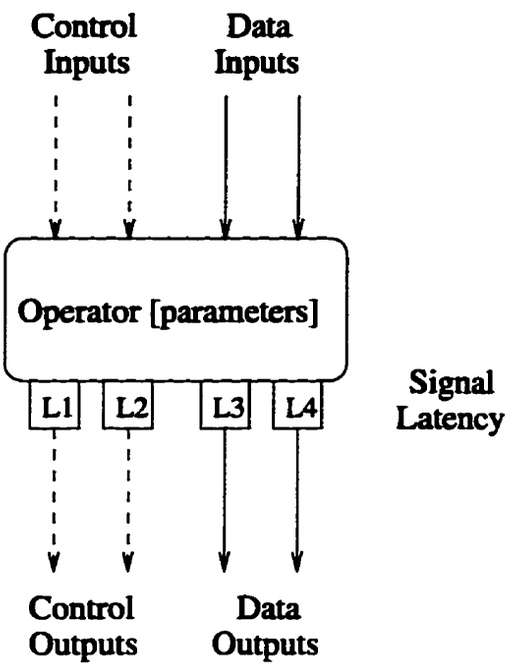


Figure 2.2: Generic DFIRST Operator

exhibit a latency which is dependent on the parameter settings for that element. this latency(L) is recorded in the small box before each of the output signals. Each output from a primitive usually has the same L but this may not be the case for all operators.

2.2 Design Example

The Signal Flow Graph (SFG) for a biquadratic [Jac89] digital filter is shown in figure 2.3. For this example let W=1 (bit-serial) and assume that the latency of each of the five multipliers is 10 and that the latency of each of the four adders is one. In any serial circuit it is useful to label a single point as reference time zero. which means that the LSD of the signal at that point is valid at time zero. For this filter

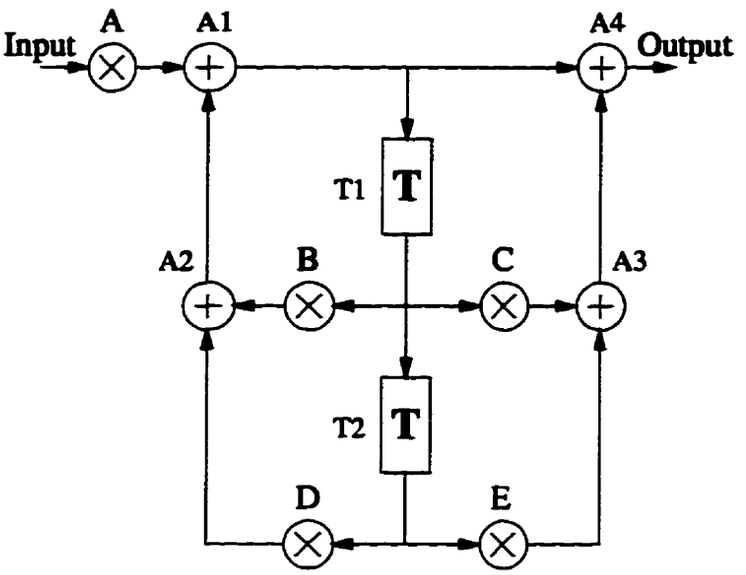


Figure 2.3: Biquad Digital Filter Flow Graph

let the input signal be the reference point. The final bit-serial circuit. complete with timing information. is shown in figure 2.4.

From figure 2.4. where the LSB times for each signal are shown. it can be seen that the minimum time required to compute each of the state variables is twelve clock cycles as found in loop1. this time is the Minimum System Wordlength (MSW) of the system. It is interesting to note that the first delay element T1 is not implemented as a separate element but is distributed through multipler B and adders A1 and A2. The second delay is implemented using a one-wordlength long shift register. So for this circuit. using these primitives, we find that Q=12 and that the LSD of the first output is completed at time twelve.

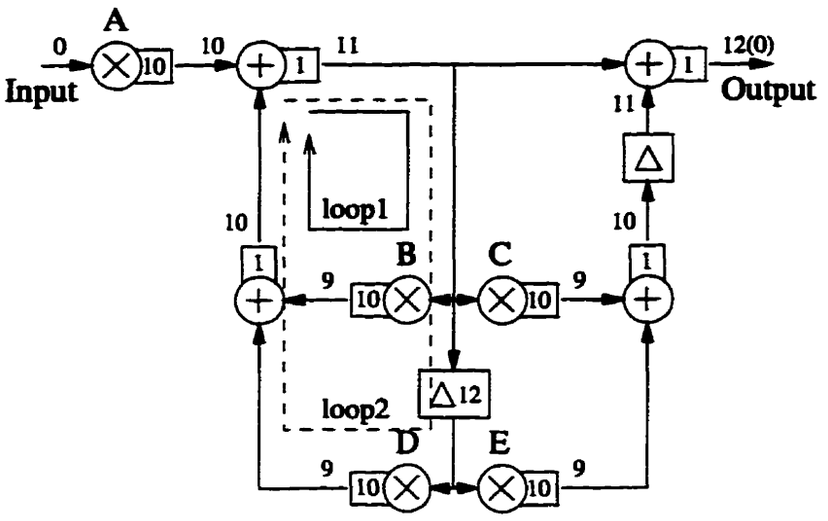


Figure 2.4: Bit-Serial Implementation of Biquad Filter

2.3 DFIRST Language Parameters

The DFIRST language is based on the FIRST (Fast Implementation of Real-time Signal Transforms) hardware description language developed by Denyer and Renshaw [DR85]. This language was used as a RTL description language for bit-serial systems. Denyer and Renshaw also developed a simulator (simfirst) for use with the language in order to facilitate rapid design and testing of FIRST descriptions. The language also included a compiler (fcc) which converted the high level FIRST language into a full custom NMOS IC implementation. In order to develop DFIRST several additions were made to this FIRST language.

The DFIRST language fully supports not only bit-serial architectures ($W = 1$) but also digit-serial architectures ($N > W > 1$) and parallel architectures ($W = N$). It is also possible to mix digit widths within a single design. The language supports elements which have a latency of zero clock cycles, such as simple gates, in order to

allow greater flexibility when designing serial circuits. Several new primitives such as CSD multipliers, parallel to serial and serial to parallel converters and conditional operators are also included to increase the range of possible applications.

2.3.1 Signal Declarations

In any hardware description language, information is transferred from operator to operator with signals or variables. Each signal can have a variety of parameters attached to it, defining the precision or format of the information contained in the variable. In the DFIRST language each signal must have a digit width which defines the W of the signal. Each signal also requires a flag indicating whether the information is serial or parallel in nature. This is required because the timing constraints on the two types of signals are different. Each signal must also be defined as either a control signal containing timing information or a data signal containing processing information.

The SIGNAL command is used to declare each data signal in the system. Several different styles of signal declarations are available, these are shown in table 2.1

From table 2.1 it can be seen that the THROUGH and * signal modifiers are equivalent in DFIRST and can be used to flag a signal as being parallel. The default digit width is adjusted using the DIGIT command which must be at the top of the DFIRST file. For example, to set the default digit width to two for a whole system, the compiler directive 'DIGIT 2' should be on the first line of the DFIRST file. The signal orientation which indicates whether the most significant bit is the smallest or largest bit number is not presently adjustable. The default setting in this case forces the MSB to be the smallest bit number in both serial and parallel signals.

Signal Declaration	Meaning
sig	A digit-serial signal of the default width.
sig0 THROUGH n	Define an n+1 bit wide parallel signal. The sig0 signal is the Most Significant Bit (MSB) of the bus.
sig[0:n]	An n+1 bit wide digit-serial signal. The sig0 signal is the MSB of the digit.
sig[n:0]	An n+1 bit wide digit-serial signal. The sign signal is the MSB of the digit.
*sig[0:n]	The * signal modifier may be used to set a signal to be parallel. This is equivalent to the THROUGH signal type from above.
sig[n]	n bit wide digit serial signal.

Table 2.1: Possible DFIRST signal Declarations

This bit order can only be over-ridden by exactly specifying this information using the sig[MSB:LSB] signal declaration.

For the most part DFIRST can accept any alphanumeric strings as signal names. there are no restrictions on the length of names, and numbers can be used in the names but not as the first character. There are however some reserved signal names which must be avoided. These include the 'VDD' and 'GND' signals which are used to tie inputs high or low respectively, and the 'NC' (no-connect) pin which is used as a place holder in situations where a signal is generated but is not used. Also, signal names cannot start with the character 'X' as this is used to specify a hexadecimal fixed value signal.

The control signals can be declared using the CONTROL signal type instead of the SIGNAL construct introduced earlier. There are in general two different sorts of control signals, LSB/MSB indicating signals and word selection signals. The LSB/MSB indicating signals are used to frame serial data words and are high for only one clock cycle per data word. Word selection signals are used for data steering

and are high or low for one or more entire data words. There is no difference in the declaration of these two different types of control signals.

2.3.2 Hierarchy

One way of reducing the complexity of a system, is to break the overall operation down into a hierarchy of simpler operations. In doing this, the task performed by a system is broken down into convenient sub-modules which do a portion of the overall processing. Each of these sub-modules can be broken down into even smaller portions resulting in further simplification. The use of hierarchy not only reduces design complexity but also increases design re-usability. It is very difficult to re-use a complete system in a new project but a sub-module which performs a generic task can easily be re-used, or modified for re-use, resulting in reduced design times for future projects.

The five hierarchical levels within DFIRST have been retained from the original FIRST language and include the SYSTEM, SUBSYSTEM, CHIP, OPERATOR and PRIMITIVE levels. The SYSTEM level represents the entire system which performs the processing task, a CHIP represents a single integrated circuit and the PRIMITIVEs are pre-designed components available within the DFIRST language. Each of these three hierarchy levels cannot make use of components at the same level, so a CHIP cannot contain another CHIP within itself. The other two levels of hierarchy are provided to facilitate partitioning. A SUBSYSTEM is a collection of CHIPS and other SUBSYSTEMs, and an OPERATOR is a collection of PRIMITIVEs and other OPERATORs.

2.3.3 Instantiation

To instantiate an operator of any hierarchical level the following syntax is used:

```
label: name [parmlist] (cntrllist) datalist
```

The *label* is an optional feature which allows the user to name an individual component for future reference. if no label is present then a unique label is generated by the program reading the DFIRST netlist. The *name* defines the type of operator being instantiated. the *parmlist* contains user defined parameters specific to each operator. The *ctrllist* contains all the control signals for the operator, and the *datalist* contains all the data signals for the operator. For the *cntrlist* and the *datalist* the inputs and outputs are separated by an arrow ($->$). For example a serial adder could be instantiated as follows:

```
ADD [1,0,0,0] (c0) a0,b0,GND -> s0,NC
```

In this example the label for the adder is not included so one will be generated by the program reading the netlist. Information on what the parameters, and I/O signals for this operator are discussed in chapter 3.

2.3.4 Encapsulation

In order to encapsulate a set of operators within a hierarchical element the following syntax is used:

```
hierarchy name (ctrllist) datalist
        CONTROL ctrlsignals
```

```
        SIGNAL datasignals
        instancelist
END
```

The first line of the hierarchical element, or macro, defines the hierarchy level, the reference name and the input/output signals for the part. The *hierarchy* must be one of the four non-primitive hierarchical types SYSTEM, SUBSYSTEM, CHIP or OPERATOR. The *name* can be any alphanumeric string which does not conflict with a PRIMITIVE name, previously defined element or DFIRST keyword. The *ctrlist* contains a listing of control inputs and control outputs with the inputs and outputs being separated by an arrow. The *datalist* contains a list of all data signal inputs and outputs, again with the I/O signals being separated by the arrow.

The internal portion of the macro defines the behaviour of the element. The *ctrlsignals* list contains a listing of all internally defined control signals separated by commas. The *ctrllist* signals must not be included here and no arrow separator is required to define input and output signals. The *datasignals* list defines the internal data signals. If the signal lists are too long the line can be continued by ending one line with a comma and continuing the signal declarations on the next line. In addition any number of CONTROL and SIGNAL commands may be present but they must occur before the *instancelist* begins.

The *instancelist* is a sequence of components which describe the functionality of the macro. The elements of the *instancelist* are instantiated as shown in section 2.3.3. There are some limitations on the components which can be used within a macro depending on the hierarchy level of the operator. Only PRIMITIVEs and

OPERATORs can be used within OPERATORs and CHIPs, and only CHIPs and SUBSYSTEMs can be used within SUBSYSTEMs and SYSTEMs. In all cases a recursive definition is illegal, so no instances of *name* can be contained within the macro description of *name*.

2.3.5 I/O Pads

The input-output pads of an integrated circuit perform the important function of buffering signals from the chip internal environment to the chip external environment. They are used to provide sufficient drive for system level signals and to provide static and overdrive protection for each I/O signal. In a synchronous system simple buffering of I/O signals is not sufficient in a multi-chip environment, due to the phase difference or clock skew between different devices. Clock skew is caused by differing propagation delays for a clock signal arriving at two or more different points. Since clocks are active on edges only, a small skew can cause a circuit to fail. In order to correct for this problem, D type flip-flops are inserted on both the input and output pads, with the input pads being rising edge triggered elements and the outputs being falling edge triggered elements. This configuration allows for a combined clock skew and signal delay of up to one half of the clock period, in either direction [DR85].

The I/O pads within FIRST are only included at the CHIP level of the language and cannot be used at any other level of the DFIRST hierarchy. The I/O pads are instantiated using two DFIRST commands, PADIN for input pads and PADOUT for output pads. The syntax of these commands is as follows:

```
PADIN (extern_cntrl->intern_cntrl) extern_data->intern_data
PADOUT (intern_cntrl->extern_cntrl) intern_data->extern_data
```

The *extern* signals represent chip external signals and the *intern* signals are the chip internal versions of the external signals. A buffer of the appropriate type (input/output, rising/falling edge triggered DFF) will exist between the external and the internal signal. The PAD commands must be placed between the signal declaration section and the instance list of a CHIP level macro.

2.3.6 Constants

Many of the primitives within DFIRST are parameterized in terms of wordlength, precision and latency. Often one change in system specifications such as the system wordlength or coefficient wordlength results in a drastic change in the parameter settings and the bit level timing for all elements in the system. CONSTANTs can be used to shorten the re-design time when high level system parameters are changed.

A CONSTANT is a string which is given a numerical value made up from other constants and numerical operations such as addition and multiplication. The possible integer arithmetic operations are addition, subtraction, multiplication and division. Brackets are also supported. The evaluation order for CONSTANTs is brackets, followed by multiplication/division and finally addition/subtraction. A CONSTANT is defined using the following syntax and must be defined before it is used within the DFIRST code. A CONSTANT can be used in any parameter location. In the following example the CWL constant is set to eight and the LATENCY constant is set to 13.

```
CONSTANT CWL=8
CONSTANT LATENCY=3*CWL/2+1
```

2.4 DSIM

For any high level language it is important to have an effective simulation program to aid in the design process. A register transfer level simulator models the behaviour of the language elements at the RTL which results in fast simulation times as compared to lower level simulations (gate level or transistor level). In order to be effective, a high level simulator should include a comprehensive design rule checker to highlight as many design faults as possible at an early stage so that fewer time consuming runs of a lower level simulator are necessary. A simulator should have an effective data entry system in order to speed up the circuit debugging and finally a simulator should be as fast as possible so that the designer does not spend an undue amount of time waiting for a simulation. The RTL simulator for the DFIRST language is the DFIRST Simulator or DSIM.

DSIM is an event driven simulator which models each data signal as a fixed point N bit integer, with a time value to indicate when that signal became valid (LSD time). The primitive functions are performed on the integer values and the time values are used to verify timing constraints on each signal.

2.4.1 Signal timing

The time steps within the simulator are broken down into two different components, the bit time and the gate time. The bit time represents the clock cycle at which a signal becomes valid. The gate time is a finer scale time which accounts for the delay time through a component which has a zero clock cycle latency. For most DFIRST components the bit time is all that is needed because most primitives have a latency

of at least one clock cycle. However some DFIRST components, such as simple gates and zero delay adders and multiplexors, have a latency of zero clock cycles. For these components the gate level timing is needed during simulation. Each clock cycle is broken down into 20 gate ticks, and the latency of any zero delay component is set to one gate tick. Any combination of zero delay elements must generate a final value within this time or the simulation will generate incorrect results. This set of delays does not reflect routing delays or even real component delays but does allow for correct unit delay simulation, which is necessary at early stages in the design process. For more exact simulations, with more realistic logic delays, a gate level timing simulator must be used.

The gate level simulation portion of DSIM uses the transport delay model [Vie93], which simulates every transition even those which may be only one gate tick wide. An alternative strategy is inertial delay modeling which effectively filters any transition which is shorter than or equal to the delay through a logical element. The transport delay model results in slightly longer settling times for logical blocks but is simpler to implement.

2.4.2 Parallel vs. Serial Signals

Most DFIRST primitives, such as ADD or MULT, have input/output signals which are exclusively serial in nature. Other primitives such as PTOSB and PMULT have inputs which must be in the parallel format. Within the simulator both signal types are treated the same in all respects except with regard to timing violations.

Each serial signal in a system travels in a LSD first format on W wires. The LSDs of all serial signals arriving at any operator must arrive at exactly the same

time. If this is not the case a timing fault will be flagged and the user must correct the timing of these signals.

Parallel signals travel on N wires and once a parallel bus is given a value it will hold that value until a new value takes it's place. This holding feature on parallel signals means that the timing of parallel signals is not as critical to proper operation. The correct data on a parallel bus must be present on a parallel bus when the control signal which samples the bus arrives.

2.4.3 Data File Format

In order to simulate a DFIRST netlist with DSIM the user must provide input stimulus information. which output signals are to be examined. the simulation duration and the data wordlength of the DFIRST system. This information is provided in the DSIM data file.

The WORDLENGTH command must be the first command in the DSIM data file. This WORDLENGTH value is used within the simulator to check overflows on all signals within the simulation. The command to set the simulation time frame is the SIMULATION CYCLES command. which indicates the number of data words to be simulated. The number of clock cycles in the simulation will be SIMULATION CYCLES * WORDLENGTH. The syntax of these two commands is as follows for a data wordlength of 16 bits and a simulation duration of 100 data words (1600 clock cycles).

```
WORDLENGTH 16
SIMULATION CYCLES 100
```

The command for inputting data to the simulator is divided into two separate portions. the signal declaration and the signal simulation data. Since the input signals to a simulation may be serial in nature. a LSD time must also be provided with the data for the signal. This timing information is added using the SIGNAL command which uses a signal within the simulation, which has known timing properties. to define the LSD time for input signal(s). In the following example the signals *ain* and *bin* will have the same LSD time as the simulation internal signal *c0.*

```
SIGNAL ain,bin SYN WITH c0
```

The synchronizing signal may be a control signal or a data signal. chip internal or chip external but generally a system level (chip external) control signal is used for this purpose since the input data signals are usually chip external.

The simulation data is provided using the *inputsignal* command. Where an *inputsignal* is any signal name declared using a SIGNAL command in the data file. Data is provided as a set of ordered pairs of numbers. the first number is the word time and the second number is the data value. The word time is defined in terms of the synchronizing signal for this *inputsignal.* A word time of zero means that at the first occurrance of the sync signal this input signal should take on the value indicated by the data value for word time zero. Each LSD occurrence of the sync signal increments the word time. Any word times which are not present will maintain the previous setting of the data value. The data ordered pair list must be terminated by either a -1 or a -2 value. A negative one value means that this *inputsignal* should maintain the same signal value (the last data value specified) until the end of the simulation. a negative two value means that the pattern of the data should be repeated

from the first value in the order pair list. In addition a separate file which contains the data for the signal can be included by using the FILE construct, the format of the data file is the same as the standard data style. The examples presented here illustrate the data format, the data terminators and the FILE command.

Example 1

```
[siga]
0 1
1 -1
2 4
3 -5
5 8
6 0
-1
```

Example 2

```
[sigb]
0 0
1 1
2 4
3 8
-2
```

Example 3

```
[sigc] FILE sig.in
```

In Example 1 the value of the signal siga follows the sequence 1, -1, 4, -5, -5, 8, 0. During time 4 the value of the signal remains as in the previous time and at any time after time 5 the value of siga is 0. In example 2 the value of sigb has a repeating sequence of '0,1,4,8' for the duration of the simulation. In the final example the file sig.in contains the data for the sigc input signal. The format of the sig.in file is the same as the standard input data.

The WATCH command is used to observe data signals during the simulation. The WATCH command specifies which signals are to be observed during the simulation and the name of the file in which to store the information. Any simulation signal can be WATCHed and the timing granularity for the output information can be set to either GATE, BIT or SAMPLE format.

If the BIT level format (default) is used then the timing information for all signals generated within the simulation will be the clock cycle number. If the GATE level format is used then all BIT times will be scaled by the gate ticks parameter. At this timing level the performance of gate level combinational circuits can be observed. The final timing format is the SAMPLE format, here the timing for the output signal is word level information. In the SAMPLE format the timing information starts from zero and is incremented by one for each successive output value.

The syntax of the WATCH command is demonstrated in the following examples.

```
Example 1
WATCH sig STORE IN sig.out
```

Example 2

```
WATCH sig1,sig2 STORE in sig.out WITH SAMPLE
```

In Example 1 the simulation information from sig is stored in the file sig.out. The timing information in this case is the default BIT level format. In Example 2 the information for both sig1 and sig2 is stored in sig.out and the timing information for both signals is set to the SAMPLE format.

The ordering of the DSIM data file commands must be WORDLENGTH, SIMULATION CYCLES, SIGNAL, WATCH, followed by the [inputsignal] data commands and the file must be terminated by an END command. Any number of SIGNAL and WATCH commands may be present in the data file, there must also be one [inputsignal] command for each signal declared using the SIGNAL command.

2.4.4 Signal Tracing

Any signal within the simulation may be used in the SIGNAL and WATCH commands, including CHIP internal signals and signals which are several layers of hierarchy deep in the design. However signal names used within the DSIM data file must be unique. Non-unique signal names will occur when several instances of a single user defined operator are used or the same signal name is used within two separate user defined operators. For these signals some of the hierarchy tree must be traced to more precisely identify which signal is being referenced.

In order to trace a signal within a particular operator which may have a conflict, the instance of the operator must be labeled as defined in section 2.3.3. Now the signals within that operator can be uniquely identified as illustrated in the following example.

```
WATCH sig1/mac1,sig2/mac2 STORE IN sig.out
```

In this WATCH command the signal sig1 within the operator labeled 'mac1' and the signal 'sig2' within the operator labeled 'mac2' will be watched and the results stored in the file sig.out. Any number of / operators may be used to trace the hierarchy of a given signal.

2.4.5 Number Interpretation

In any bit serial system a number of different effective wordlengths may be present. The primary wordlength is the data wordlength or System WordLength (SWL) of the circuit in question. In addition to this data length each multiplier may have a different Coefficient WordLength (CWL), and each Parallel converter may have a different number of parallel bits to convert.

The rule for interpreting or inputting data to all these parts is that each data value is a two's complement (TC) number of the required length. A value of -1 indicates that all bits of the word in question are set to one. A value of 1 indicates that the Least Significant Bit (LSB) of any word is one and all other bits are zero.

2.4.6 Simulation errors and warnings

One of the most important aspects of a simulation program is the error reporting. The earlier an error is identified the less time is wasted proceeding with a faulty design. DSIM reports two general kinds of circuit errors: design rule errors and simulation timing errors.

The design errors reported by DSIM include sourceless or loadless nets or multiply driven nodes occurring within the design. Loadless nodes can occur within a design

and result in only wasted area generating signal(s) which are not ultimately needed. Sourceless nodes however must be corrected before final implementation, since a floating input to a logic element results in undefined values at the output of that element. These sourceless nodes may be tied high or low using VDD or GND signals to absolutely define the operation of the resulting logic. Multiply driven nodes are also design errors which must be corrected. Each node in a DFIRST file must have only one driving element.

The second kind of errors reported by DSIM include all simulation timing errors and internal signal overflows. Unlike the design rule errors, these problems are progressively harder to identify and correct as circuit refinements are performed. Design rule errors can be flagged at any level of abstraction and are readily traceable to a single cause. Timing errors are more difficult to trace to a single root cause.

All serial signals arriving at a DFIRST primitive have strict requirements for their relative LSD times. For most elements all input signals must have their LSD valid during the same clock cycle for correct operation. If a timing violation occurs DSIM reports the offending symbol and the timing of all signals connected to that primitive. Using this information a timing error can be quickly corrected.

Internal signal overflows occuring during a simulation are also reported by DSIM. All data signals within the DFIRST language are fixed precision values. A word overflow can occur after addition, subtraction, and multiplication by constants greater than one. DSIM checks all values generated within a simulation against the bounds of it's precision and if an overflow occurs, DSIM reports the violating signal and the time at which the violation occurred. The simulation continues after this error but the results generated may not accurately reflect real circuit operation.

2.5 Summary

DFIRST is a Register Transfer Level language which supports the description of bit-serial or digit-serial DSP circuits. All serial signals in DFIRST travel on W wires in the LSD first data format. The serial signals within the DFIRST language require a framing control pulse which indicates where either the LSD or the MSD is in the data word. The DFIRST language uses five levels of hierarchy (PRIMITIVE. OPERATOR. CHIP. SUBSYSTEM. and SYSTEM) to simplify the partitioning of a large application into smaller pieces. The DFIRST language also supports clock skew robust CHIP to CHIP interconnection of serial data signals.

The DFIRST primitives are parameterized in terms of precision, digit-width data wordlength and latency. Each primitive exhibits a latency (L) which defines the number of clock cycles between the arrival of input signal(s) to the operator and the generation of the output signal(s).

The event driven DSIM simulator is used to simulate the performance of circuits described in the DFIRST hardware description language. DSIM loads both the DFIRST netlist and a data file to perform a simulation. In the data file a user describes the wordlength of the serial signals within the circuit. the length of the simulation. the input signals and their desired stimulus and the signals which are to be monitored during the simulation. DSIM also performs design rule checks on DFIRST circuits and provides feedback on serial timing alignment and overflow errors which may occur during a simulation.

Chapter 3

DFIRST Primitives

In this chapter the gate level form of some DFIRST primitives is presented. In particular DFIRST adders/subtracters, right/left shifters, multipliers and format converters are implemented using non-technology specific or generic gate level elements.

In addition to generic gate level implementations, where appropriate, alternate implementations are presented to highlight the strong points and the weak points of each implementation. The latency and critical path of each operator is discussed in terms of the operator parameters and digit width.

3.1 Standard Components

All DFIRST primitives are made up from a small set of logic elements, including standard gates such as AND, OR, XOR, NAND, NOR, XNOR and INVERTERs as well as two to one multiplexor elements, delay elements (DFFs) and BLATCHes. A BLATCH is an active high clock enabled DFF element which only latches the input signal to the output if the control pin is high. The internal logic for multiplexors and BLATCHes as well as their schematic representation are shown in figure 3.1. The control signal for both of these elements is normally connected vertically, and the data pins(s) are connected horizontally to the component. The upper input signal of the MUX is selected if the control signal is low and the other data input is selected

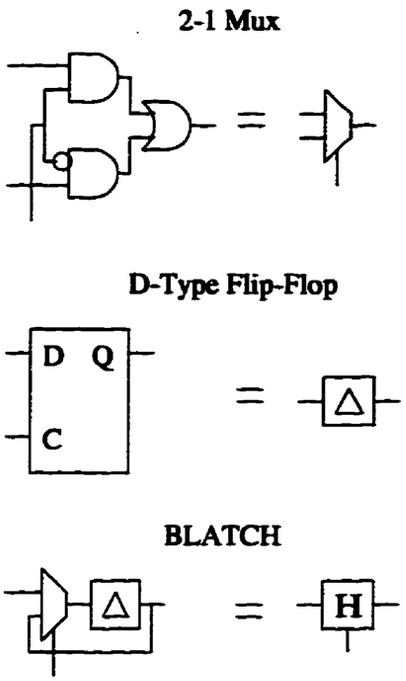


Figure 3.1: Schematic Representation for a MUX. DFF and BLATCH element

if the control signal is high.

3.2 Storage

Bit-serial or digit serial data storage is implemented using the BITDELAY DFIRST primitive. This primitive may also be used to control the timing alignment of serial signals. The BITDELAY primitive is implemented using W shift registers of the appropriate length. A W=2, length=6 BITDELAY primitive is implemented using 2 6-bit shift registers. For control signals the CBITDELAY primitive is used. The implementation of this primitive is the same as the BITDELAY primitive.

The instantiation of BITDELAY and CBITDELAY primitives is as follows. Note that the digit-width of the BITDELAY primitive is determined by the digit-width of

the input/output signals of the BITDELAY part. The digit-width of the input and output signal must be the same.

```
BITDELAY [3] din -> dout
CBITDELAY [3] (cin -> cout)
```

3.3 Serial Adder

Addition is a naturally LSD first operation so it is well suited to the DFIRST architecture. A carry-save bit-serial adder is shown in figure 3.2. During the first clock cycle the LSBs of both input words are present on the data lines A and B. At this LSB time the control signal is high and is used to set the carry input on the full-adder to a user defined value. The sum is generated and delayed by one clock cycle to pipeline the operator (latency=1) and the carry is stored for use in the next clock cycle. On the next clock cycle the second bits of both A and B are available and the carry output from the previous clock cycle is used as the carry input to the full adder. The addition proceeds in this manner until all N bits of the data words are processed.

The extension of the bit-serial adder to a digit-serial adder can be implemented by using W full adders with a ripple carry from the LSB to the MSB of the digit. the most significant bit carry output is saved and used as the LSB carry input on the following clock cycle. During the LSD time the control signal clears the carry feedback and sets the carry input to a user selected value. The resulting hardware for a digit serial adder with W=3 is shown in figure 3.3.

As the data wordlength (N) increases the hardware size of the digit-serial adder

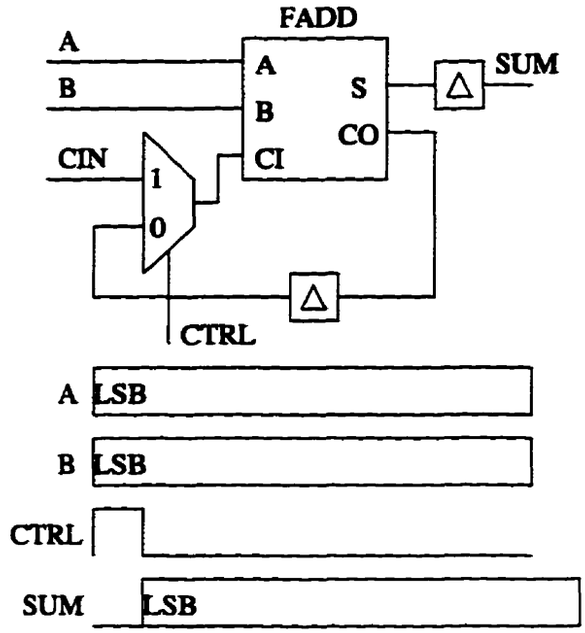


Figure 3.2: Bit-Serial Carry Save Adder

does not change, unlike parallel adders which linearly increase in size with data wordlength[1]. The number of clock cycles required for the complete addition is N/W. This operator can be converted to a subtraction operator by complementing all W bits of the B input and setting the carry input high (complement and add one).

The number of full adders and pipelining delays for the sum is equal to the digit width, so the size of digit-serial adders increases linearly with W. The number of clock cycles required to process a given data word decreases linearly with the digit width because more bits of the data words are being processed during each clock cycle. The maximum logic propagation delay or critical path of a digit serial adder is W full adders plus the carry input generation logic.

The DFIRST call to a digit serial adder is:

[1]Ignoring fast carry methods.

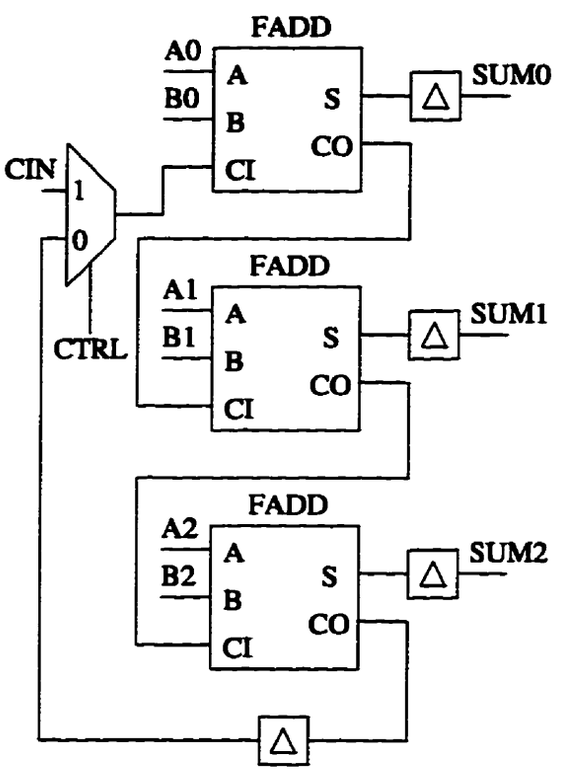


Figure 3.3: Digit-Serial Carry Save Adder (W=3)

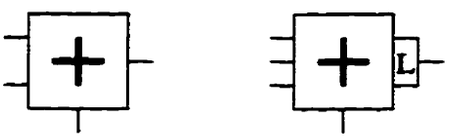
```
ADD [latency,0,0,0] (c0) a,b,carin -> sum,carout
```

The latency parameter of an ADD element selects the latency (L) of the adder and can be any integer value (including zero) in the DFIRST language. W*latency pipelining delays are used at the output of the adder to generate the desired latency. If the latency selection is zero then no pipelining delays are placed at the sum output. This option must be used with some care because a cascade of zero delay parts will result in longer critical paths for the overall circuit. Whenever using zero delay components some consideration must be given to the type of operators being driven by the primitive.

The second. third and fourth parameters are remnants from the old FIRST language and were used to optionally implement pre-delays of one bit on each of the three input signals. This feature is not supported in the DFIRST language and zeros are inserted in place of these values. In order to implement these delays separate BITDELAY operators must be used.

The *c0* signal is the LSD indicating framing control signal while *a* and *b* are the data inputs which must have the same LSD time as the control signal. The *carin* signal is normally set to GND but can be set to VDD to implement a carry input of one. The *sum* is the sum output of the adder and the *carout* signal is the carry output from the Most Significant (MS) full adder. The carry output signal is not generally used and a NC signal is often connected here. The digit width of the adder is selected by the digit width of the input and output data signals. All three data signals *a.b* and *sum* must be of the same digit width.

The DFIRST instantiation of a subtracter is:



(a) (b)

Figure 3.4: Adder Schematics (a) cin grounded and zero delay (b) latency of L

```
SUBTRACT [latency,0,0,0] (c0) a,b,borin -> diff,borout
```

The subtracter has the same options as the DFIRST adder. The *borin* signal is normally set to GND but can be set to VDD in order to implement an increment by one on the difference output.

3.3.1 Adder schematic

The digit-serial adder is one of the primitive elements used to build up more complex operators as discussed in the following sections. The schematic diagram(s) representing a digit-serial adder is shown in figure 3.4. If there are only two inputs on one side of the adder then the cin signal is assumed to be connected to GND (grounded). The signal connected to the bottom of the element is the control signal and the signal on the right is the sum output from the adder. If no latency block is present then the adder has a latency of zero clock cycles. otherwise the latency is indicated by L. If there are three inputs then the lower most is the cin signal. The adder can be converted to a subtracter by placing a negative sign above one of the two data input signals. A negative sign above both data signals is invalid as this operation cannot be implemented in a single serial adder.

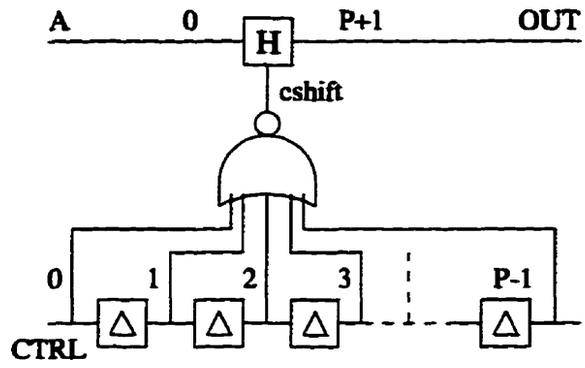


Figure 3.5: Circuit for Bit-Serial right shift by P

3.4 Right Shift

A right shift operation can be used to perform a power of two division. This operation right shifts the N bit data word by p bits resulting in a division by 2^p. In TC right shifting, the lower p bits are removed and the upper p bits become sign extensions, the p+1th bit is the LSB of the output word. The general form of a bit-serial right shift element is shown in figure 3.5.

The data signal (A) arrives at the BLATCH element at time zero. The control path delay chain is p-1 delays long, the p points of this delay chain are used as inputs to a p input nor gate. if p=1 then the nor gate becomes an inverter. The output from this nor gate is low for p clock cycles starting at time zero. While *cshift* is low the BLATCH element recycles the previous output from the shifter which is the MSB of the previous word in a fully word packed system. This performs p bits of sign extension on the previous data word. Also at this time the p least significant bits of the present word are discarded. On the next clock cycle the p+1th bit of the present word becomes the next output and is delayed by one clock cycle by the

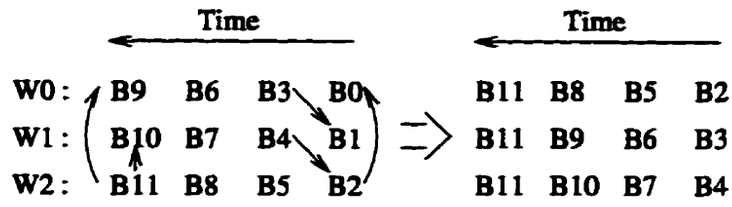


Figure 3.6: Twelve bit W=3 serial data word and data word right shifted by two

flip-flop in the BLATCH. The resulting latency of this primitive is p+1 clock cycles. p for the right shifting action and one to pipeline the operator.

3.4.1 Digit Serial

The digit serial version of the DSHIFT element is more complicated to implement because the bits making up each digit are re-arranged if the shift value p is not an integer multiple of the digit width W. Consider the digit serial data word shown in figure 3.6, with W=3 and a shift value of p=2.

This right shift operation has resulted in the W0 line being moved to the W1 line with a delay of one, the W1 line being moved to the W2 line also with a delay of one and the W2 bit being moved to the W0 position with no delay. Also, the upper two bits have become sign extensions. In order to implement an arbitrary digit width right shift element it is useful to break the operation down into two components. The first portion re-orders and delays the bits forming the digit as well as performing sign extension across the digit. The second portion performs right shifts by integer multiples of the digit width.

Since there are an arbitrary number of possible digit widths and shift values it is not practical to use a library of different elements to implement the two sections of the shifter. Instead all of the hardware for the shifter is generated using delay

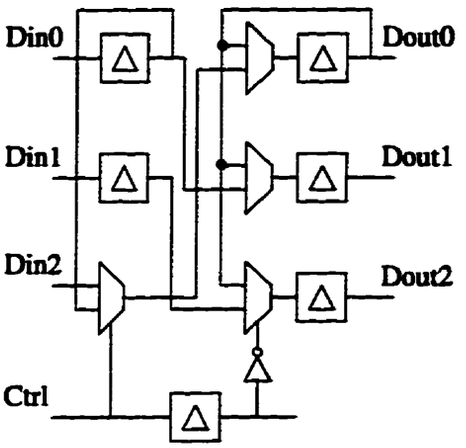


Figure 3.7: Digit-Serial right shift operator for W=3 and p=4

elements and multiplexors.

The second section of the right shifter is the same as the bit-serial version only repeated for each bit of the digit. Each bit is a selection between the incoming bit of data or the sign bit of the previous word. If the sign bit is selected then the present word is right shifted and the previous word is sign extended. The hardware generated for a W=3. right shift by four operator is given in figure 3.7.

The latency of digit serial right shift operators is determined as follows. If the shift value p is an integer multiple of the digit width W then the latency is due to the integer shifting section only and is $p/W + 1$. If $p < W$ then the latency of the operator is due to the second section only and is one. When a combination of both sections is used the Latency is $p/W + 2$.

3.4.2 DFIRST instantiation

The right shift operator in DFIRST is the DSHIFTA primitive and is instantiated as

```
DSHIFTA [p] (c0) a -> out
```

where p is the number of bits by which to right shift. The *c0* control signal and the *a* data signal must have the same LSD time. Another operator which is maintained from the FIRST language is the DSHIFT operator which performs the same operation but has a latency of p+3 for the bit-serial shifter and is not available in digit-serial. The DSHIFT operator accepts the same parameters and input signals as the DSHIFTA operator but a second parameter which enables a pre-delay on the input signal is also required. The pre-delay feature is not implemented in DFIRST and if needed must be implemented using the BITDELAY primitive.

3.5 Left Shift

Left shifting is a common operation which can be used to implement a multiplication by a power of two. The left shift operation left shifts the N bit data word by m bits performing a multiplication by 2^m. In TC left shifting the bottom m bits of the data word become zero and the top most m bits of the input must have been sign extensions prior to the left shift operation or an overflow occurs. The general form for a bit-serial left shifter is shown in figure 3.8.

The data signal *IN* and the input control signal *CTRL* arrive at the shifter at time zero. The control signal is delayed by m-1 clock cycles, much like the right shift operator, and m nodes of this shift register are nored together to generate the *cshift* signal. If m=1 then no shift register is required and the cshift signal is the inversion of C. During the m clock cycles in which cshift is low the output from the shifter is zero, during this time m bits of the input data word must be stored to be output in

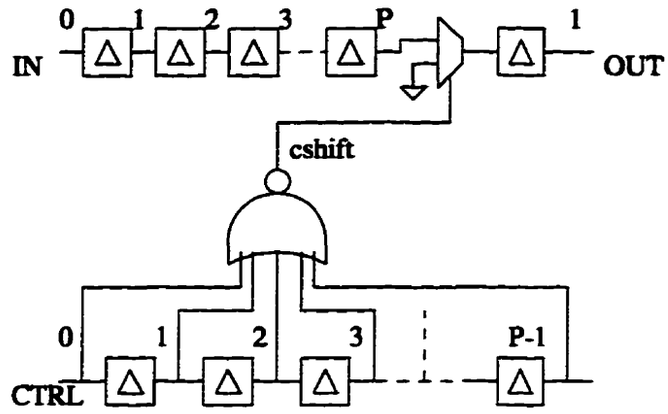


Figure 3.8: Circuit for Bit-Serial left shift by P

their new left shifted positions. this is done with an m bit shift register on the input data line. After m clock cycles the LSB of the input word arrives at the multiplexor input and the cshift control signal goes high allowing this bit and the remaining N-m bits of the input word through to the output pipeline delay. For this implementation the upper most m bits of the input signal are not present in the output signal so these bits of the input word should have been sign extensions or the output of the shifter is not correct.

The latency of the left shift component for any value of m is one due to the pipelining delay at the output of the multiplexor. Unlike most DFIRST operators the left shifter can be implemented with a negative latency since the first relevant bit of the input signal arrives at the operator m clock cycles after the start of the operation. If the operator is started at time -m then the LSB of the input signal can be connected directly to the multiplexor input and no data delay shift register is required. This implementation is faster (negative m latency) and smaller (no input storage chain) but has not been implemented to remain consistent with other

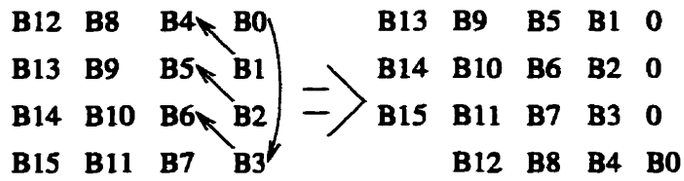


Figure 3.9: Sixteen bit W=4 serial data word and data word left shifted by three DFIRST primitives.

3.5.1 Digit Serial Left Shifter

The digit-serial left shift operator is very similar to the digit-serial right shift operator in that the bits which make up each digit have to re-arranged if the number of bits to shift by is not an integer multiple of the digit width. A W=4 digit-serial data word and the resulting data word after a left shift by 3 bits is shown in figure 3.9. For this shifting operation the W0 digit is moved to the W3 bit position and the W1.W2. W3 bits are moved to the W0. W1 and W2 bit positions respectively in the next digit. So some bits may be moved and others may be moved and delayed.

The left shift element may be broken down into two sections. one section to perform the bit re-arrangement and another section to perform left shifts on entire digits.

The second portion of the left shift element is the same as the bit-serial component. repeated for each bit forming the digit. The number of delays to insert in front of the multiplexor and the number of clock cycles which the multiplexor must select the zero is equal to the integer division of m/W. In order to generate the control signal for the multiplexors in this stage m/W-1 delays must be used for the control path and an m/W input nor gate must be used to generate the select signal. If

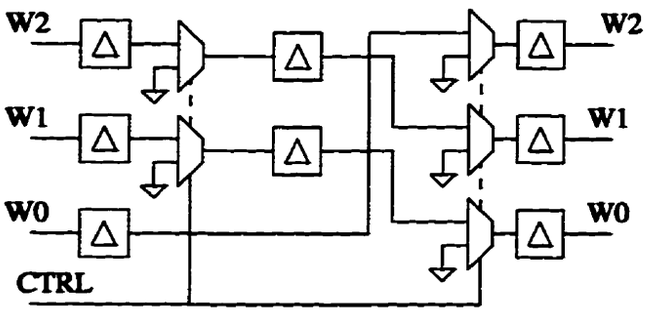


Figure 3.10: Digit-serial Left shift operator with W=3 and P=5

m/W=1 then no control delay chain is needed and the select signal is the inversion of the input control signal. The resulting hardware for two left shift operations and the inputs and resulting outputs are shown in figure 3.10.

The latency for the arbitrary digit width left shift operator is always one regardless of digit width or shift value. Both sections of the element are generated without pipelining delays and the output of the operator is pipelined to force a latency of one. The digit-serial left shifter can also be implemented with a negative latency saving all data storage elements within the shifter. This implementation has not been included in DFIRST to remain consistent with other primitives.

3.5.2 DFIRST

The left shift operator within DFIRST is the MSHIFT operator which is instantiated using the following syntax:

```
MSHIFT [m,0] (c0) in -> out
```

The m parameter indicates the number of bits to left shift the input data signal (in) by. The control signal (c0) and the input signal must have the same LSD time. The digit width of each MSHIFT operator is determined by the digit width

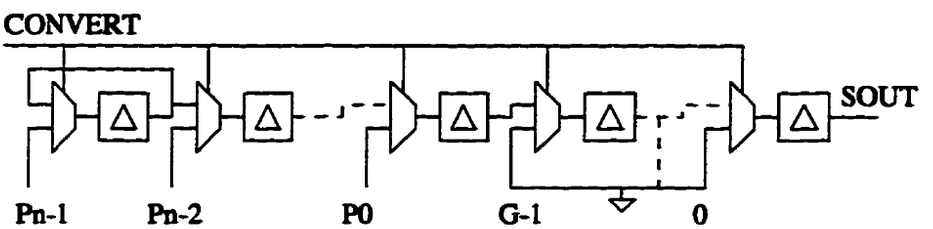


Figure 3.11: Bit-Serial Parallel to Serial Converter

of the input and output data signals which must both have the same digit width. The second parameter represented an optional pre-delay on the input signal in the FIRST language. but is not supported in DFIRST.

3.6 Parallel to Serial

In most digital systems data is communicated in a word parallel format using NP separate wires for an NP bit data word. To interface this parallel information to a serial processing system. format conversion elements such as parallel to serial and serial to parallel converters must be used.

The parallel to serial converter is responsible for converting an external NP bit parallel data word to an N bit serial word for internal use. If $NP < N$ then the parallel to serial converter must provide sign extension for bits to the left of the incoming data and insert zeros for the bits to the right of input word. If $NP > N$ then the N most significant bits of the parallel data should be used as input. The hardware for a simple bit-serial parallel to serial converter is shown in figure 3.11.

Each element of the converter is a multiplexor in front of a D type flip-flop. If the control signal is high then the parallel input lines are loaded into the bit delays. otherwise the other mux input is selected forming a shift register. The converter

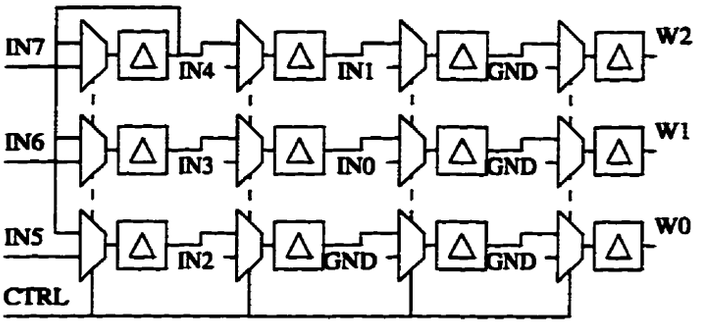


Figure 3.12: Parallel to Serial Converter (W=3, NP=8, G=4)

shown in figure 3.11 accepts NP bits of parallel input, inserts G ground bits to the right of the parallel data and can perform an arbitrary amount of sign extension due to the recirculation of the MSB.

The conversion of a bit-serial parallel to serial converter to digit-serial is relatively straight forward. The latching elements of the converter are arranged in a W bit wide fashion with the parallel inputs of the least significant G bits being tied to ground and the parallel inputs of the remaining NP elements being connected to the appropriate bits of the input signal. The serial inputs of all elements are connected to the output of the previous latch in the same bit position of the W bit wide digit. The serial inputs of the upper most elements are all connected to the serial output from the latch to which the sign bit of the parallel input is connected. The resulting hardware for a W=3, NP=8, G=4 parallel to serial converter is shown in figure 3.12. The latency of all converters in this configuration is one, from latching the parallel inputs to the generation of the LSD of the serial output.

3.6.1 Word Recirculation

In many applications it is useful to be able to save a parallel input signal for future use. This operation can be performed within the parallel to serial converter by changing the shift register operation of the converter to a recirculating register form. This feature requires one additonal control signal to place the converter in recirculating mode and one additional parameter to set the data wordlength to be stored.

The form of this converter is similar to the standard non-recirculating form except for the addition of a feedback path from the serial output to most significant bit of the converter. In this feedback path there are N-NP-G additional flip-flops which makes the total loop delay around the device exactly one SWL. The *lcntrl* signal goes high for one clock cycle to load the parallel inputs and then remains low while the converter is shifting and recirculating. The serial input to the most significant element of the converter is now one of three things, parallel input. sign extension or recirculating data. The scntrl signal is used to control the number of sign extensions and must be high for SWL-NP-G clock cycles. A full recirculating parallel to serial converter with NP=8, G=4. SWL=16 is shown in figure 3.13

The *cin* signal is the standard LSB indicating signal and the *cmux* signal is a higher order control signal which is high for one entire data word and then low for one or more full data words. With this converter the LSB of the input data word is valid once every SWL clock cycles until the converter is loaded again by both *cin* and *cmux* being high. The latency of this converter is now two clock cycles because of the additional DFF in the control path. If the recirculating property is not needed

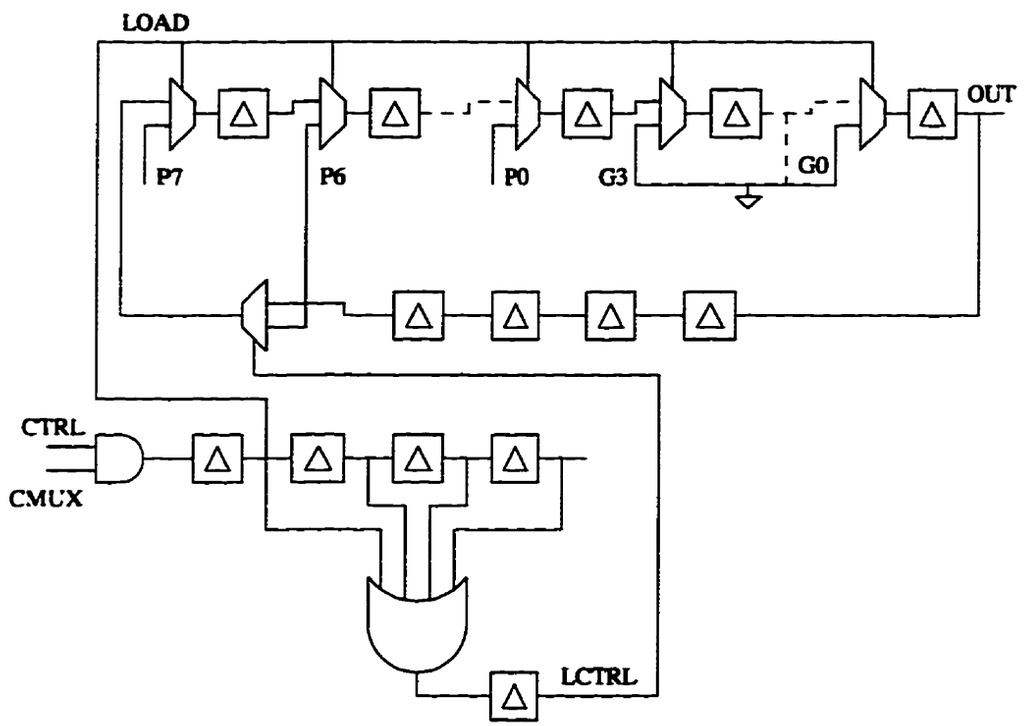


Figure 3.13: Parallel to Serial Converter with data storage

then the *cmux* control signal is tied high and the additional hardware needed to implement word storage is not generated. In this case the latency of the converter is still two to maintain consistency.

3.6.2 Fan-out Control

In the parallel to serial converter the control signal used to load the parallel data and least significant guard bits is connected to at least NP+G loads. This high fanout results in slow transitions of the control signal. this transition time may become the critical path. limiting the maximum bit clock frequency. depending on the magnitude of N+G and the technology being used. In order to control this situation a Fan-out Control (FC) parameter has been added to the parallel to serial primitive. The FC parameter indicates the number of driving points which are available for the parallel loading signal. The N+G loads within the converter are divided equally between the FC driving points. These signals are generated by FC parallel DFFs whose inputs are connected to the output of the *cin-cmux* AND gate.

3.6.3 DFIRST

The DFIRST call for a parallel to serial converter is:

```
PTOSB [NP,G,FC,SWL] (cin,cmux) in0 THROUGH N-1 -> out
```

Each of the four parameters are as described above and the $SWL >= NP + G$. The most significant bit of the NP bit input bus is *in0*. The latency of this part is always two from the arrival of the *cin* signal to the generation of the LSD of the output signal. The recirculating and fanout control options are not available on digit

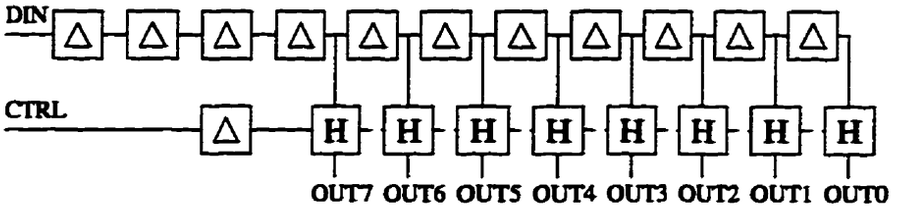


Figure 3.14: Bit-Serial Serial to Parallel Converter (NP=8,SK=2)

serial parallel to serial converters. These two features are used infrequently and were not included for the digit serial version of the part.

3.7 Serial to Parallel Converter

A serial to parallel converter can be used to convert a DFIRST serial signal to a parallel format used by an external parallel data device. To define the operation of the DFIRST serial to parallel converter two parameters are required. the number of parallel bits to generate (NP) and a formatting parameter which selects the NP bits of the serial word to convert to parallel. In the parallel to serial converter this was done by selecting the number of least significant end guard bits to add. but in the serial to parallel converter this parameter is the number of most significant bits to skip (SK). This component can be broken down into two parts, a shift register to store the serial word and a NP bit parallel latch to load and store the parallel data until the next output time. A bit-serial serial to parallel converter is shown in figure 3.14 with NP=8 and SK=2.

At the LSB time of the present data word the previous data word is present in the shift register, this word is latched into the parallel storage elements. The length of the shift register chain is NP+SK and the final output of this chain is

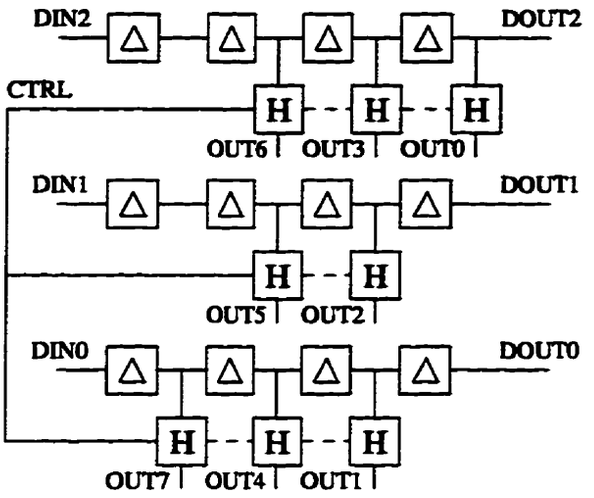


Figure 3.15: Serial to Parallel Converter (W=3. NP=8. SK=2)

available as a second output from the primitive. For proper use all of the bits which have been skipped must be sign extensions or the parallel output will not reflect the true value of the serial data word. The additional delays on the control and data input are present to facilitate a fan-out control signal and have a consistent delay performance for the primitive. Like the parallel to serial converter one control signal drives several (NP) loads which may be excessive. In order to combat this fanout problem FC separate control signals are generated using FC DFFs and the NP loads are shared equally between each DFF output.

A digit-serial serial to parallel converter is very similar to the bit-serial primitive. The only change is that the shift register used to store the data word is W bits wide and the NP bit parallel output signal is derived from selected points within this shift register. Like the parallel to serial converter the fanout control parameter is not available for digit serial components. The hardware for a W=3. NP=8. SK=2 serial to parallel converter is shown in figure 3.15.

3.7.1 DFIRST

The DFIRST call to a serial to parallel converter is:

```
STOPB [NP,SK,FC] (ctrl) in -> delout, out0 THROUGH NP-1
```

The three parameters perform the functions discussed above. The *ctrl* control signal indicates the LSB of the previous data word to the word being converted to parallel. The *delout* data signal has a latency of NP+SK+1 and the parallel output is generated two clock cycles after the primitive is triggered.

3.8 Multiplication

Most signal processing applications require several multiplications which must be performed in as little time as possible, using as little device area as possible. DFIRST multipliers perform the multiplication of an N bit data word by a Coefficient Word Length (CWL) bit coefficient. Depending on the application different types of coefficients may be used such as fully general bit-serial, fully general parallel and constant coefficients. In the following sections the hardware implementations of DFIRST multipliers are discussed.

3.8.1 Two's Complement Multiplication

A general twos complement multiplication can be implemented using equation 3.1, where P is the product, Y is the CWL bit coefficient and X is the N bit data word.

$$P = -2^{CWL-1} * X * Y_{CWL-1} + \sum_{i=0}^{CWL-2} 2^i * X * Y_i \qquad (3.1)$$

```
                    X5 X4 X3 X2 X1 X0
                  × Y5 Y4 Y3 Y2 Y1 Y0
                  ------------------
                    X5 X4 X3 X2 X1 X0  × Y0
                 X5 X4 X3 X2 X1 X0     × Y1
              X5 X4 X3 X2 X1 X0        × Y2
           X5 X4 X3 X2 X1 X0           × Y3
        X5 X4 X3 X2 X1 X0              × Y4
     - X5 X4 X3 X2 X1 X0               × Y5
  ------------------------------------------
  P10P9 P8 P7 P6 P5 P4 P3 P2 P1 P0
```

Figure 3.16: Two's Complement Long Multiplication

For this multiplication there are CWL partial products. Multiplication of each partial product by 2^i is done by left shifting the input data value by i and multiplication by the ith bit of the coefficient is done by anding the entire data word with the ith coefficient bit. To implement the negative sign bit of the coefficient the most significant partial product is subtracted instead of added as for the other partial products.

The resulting product is CWL+N-1 bits long. In finite precision machines only a fixed number of bits resulting from a multiplication can be saved as the result. any remaining bits are discarded resulting in truncation errors in the product. For the DFIRST architecture the outputs from all operators must conform to the N bit data word size. so CWL-1 bits of product must be discarded. The product can be broken into two portions, the upper N bit word and the lower CWL-1 bit word. If the lower data word is discarded the multiplication becomes fractional with the decimal point residing just to the right of the most significant bit of the coefficient. A long multiplication with the CWL=N=6 is shown in figure 3.16.

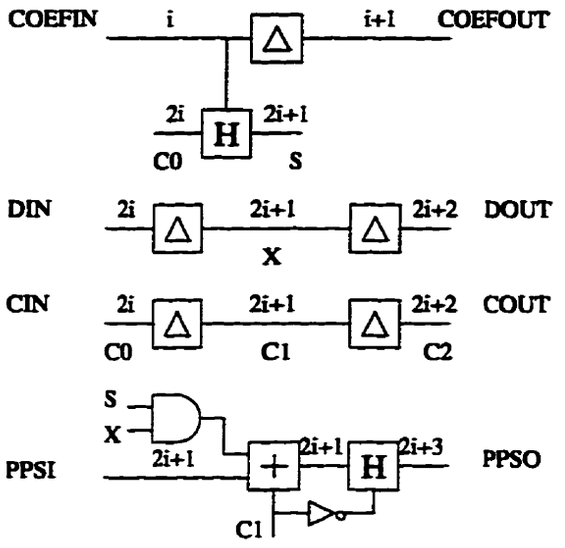


Figure 3.17: Bit-Serial Two's Complement Multiplier Stage

3.8.2 Two's Complement Multiplier Implementation

In designing the simple twos complement multiplier it is assumed that the least significant bit of the CWL bit coefficient and the N bit data signal arrive at the multiplier at time zero. Also at this time a LSB indicating active high control signal is available. The multiplier can be broken down into CWL different sections. each generating a single output which is the accumulated product obtained from the weighted summation of all partial products generated up to and including the present stage. All of the CWL stages are identical except for the final stage which implements the sign bit of the coefficient.

There are CWL different sections in the multiplier, the ith stage generates the ith partial product and sums this product with the accumulated results from the previous stages. The first stage in the chain is the zeroeth. The hardware for the ith section where $CWL - 1 > i > 0$ is shown in figure 3.17.

The ith section must store the ith coefficient bit for use in the ith stage and delay the serial coefficient input for use in the i+1th stage. The time that the coefficient bit arrives at the ith section is i. The control signal valid at time 2i is used to latch the coefficient bit into the S signal which is valid one clock cycle later or 2i+1. The LSB time of the data arriving at the ith section is 2i and this data is delayed by two for input to the next stage. The data signal valid at time 2i+1 is marked as X. The control signal is delayed in the same manner as the data. The data (X) is multiplied by the stored ith coefficient bit using an AND gate where both S and X are valid at time 2i+1. This partial product is added with the summation of previous partial products, also at time 2i+1. The PPSI input of the zeroeth stage is grounded. This partial product summation is right shifted by one bit for addition in the next stage of the multiplier. The latency of the shift operation is two, resulting in a time of 2i+3 for the ith partial product.

To verify the boundary times for this multiplier section it is required that the input time of the i+1th stage be equal to the output time of the ith stage for each signal. The timing verification for each data and control signal is shown in table 3.1.

Signal	ith Output	i+1th Input
Coefficent	i+1	i+1
Data	2i+2	2(i+1)
Control	2i+2	2(i+1)
Partial Product Summation	2i+3	2(i+1)+1

Table 3.1: Timing Alignment Verification for Two's Complement Multiplier Stage

The circuitry for the final ($i = CWL - 1$) stage is shown in figure 3.18. The final stage differs from the previous stages in three regards. The serial coefficient is not passed on to another stage, the partial product generated in this stage is subtracted

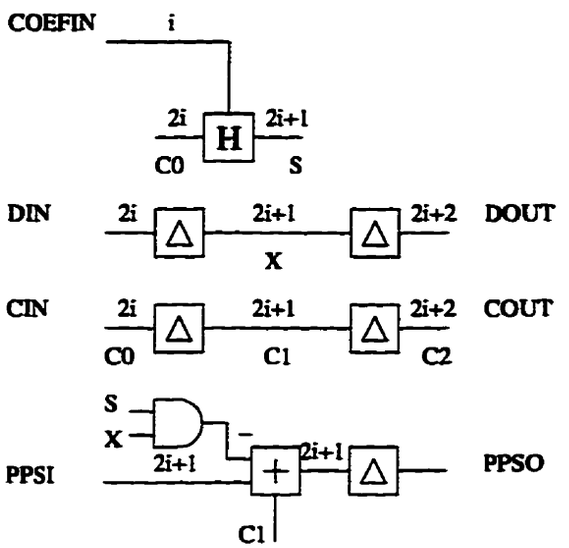


Figure 3.18: Bit-Serial Two's Complement Multiplier Final Stage

from the previous partial products because this stage represents the sign bit of the coefficient and finally the result of the summation is not right shifted by one because the output from this adder is already at the correct precision as shown in figure 3.16. The resulting latency for this multiplier implementation is 2CWL clock cycles.

3.9 Coefficient Recoding

Binary numbers are often represented in higher radix forms to reduce the complexity of representing a given number. Some commonly used number formats are octal (radix 8) and hexadecimal(radix 16) which require N/3 and N/4 digits respectively to represent an N bit binary number. Recoding the multiplier coefficient into higher radices can be used to design smaller/faster multiplier implementations [PM89. Lyo76].

Recoding methods can be broken down into two major types. redundant and non-

redundant. In non-redundant recoding methods each bit of the coefficient is used to generate only one recoded value, while in a redundant recoding some bits of the coefficient may be used in determining several different recoded values. Redundancy can be added to a recoding in order to simplify the possible recoded values [Lyo76] at the expense of increased recoding cost.

The second major characteristic of recoding method is the radix. The radix of a recoding scheme determines how many bits of coefficient are examined to generate a single recoded output. The radix of a non-redundant recoding is 2^R where R is the number of bits used to generate a single recoded value. The radix of a redundant recoding method is $2^(R-1)$. The TC multiplier is a non-redundant radix two recoded multiplier implementation. In the following sections different recoding methods are used in order to obtain a smaller/faster multipler.

3.9.1 Booth Recoding

A bit-serial two's complement multiplier has an irregular implementation since the final stage is different than the previous stages. The multiplier can be made completely regular by using a redundant radix 2 recoding on the coefficient bits of the multiplier [PM89]. In this recoding scheme two bits of the coefficient are combined to create a single signed recoded coefficient bit in the set -1,0,1. The product for a booth recoded multiplication can be calculated using equation 3.2. In this recoding the ith bit is implemented as the i+1th bit minus the ith bit. The recoding table for the pair of bits y_i and y_{i-1} is given in table 3.2. If i=0 then the i-1th bit is set to zero.

y_i	y_{i-1}	Recoded Value
0	0	0
0	1	1
1	0	-1
1	1	0

Table 3.2: Booth Recoding for two coefficient bits

$$P = \sum_{i=0}^{CWL-1} (y_{i-1} - y_i) * X * 2^i \tag{3.2}$$

A multiplier implemented using this recoding is regular but the size of each module is larger than the module size for the TC multiplier and no speed up is obtained so this recoding is not used for DFIRST multiplier implementations.

3.9.2 Modified Booth Recoding

The modified booth recoding uses three coefficient bits to generate a single representation for two bits of the coefficient Each two bits of the coefficient are used to generate a single partial product. so only CWL/2 partial products are needed to generate the final product. In modified booth recoding the i and the i+1th bit form a single digit. If the ith bit is set. it is implemented as one ith bit. and if the i+1th bit is set it is implemented as the i+2th bit minus the i+1th bit. Given this relationship the product for a TC multiplication using this recoding scheme can be calculated using equation 3.3.

$$P = \sum_{i=0}^{CWL/2-1} (y_{2i-1} + y_{2i} - 2y_{2i+1}) * X * 4^i \tag{3.3}$$

Using this recoding method each two bits of the coefficient are represented by one element from the set -2. -1. 0. 1. 2. The recoding table for the bits y_{i+1}. y_i and

y_{i+1}	y_i	y_{i-1}	Recoded Value
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

Table 3.3: Modified Booth Recoding for three coefficient bits

y_{i-1} is given in table 3.3. For this recoding y_{-1} is set to zero.

3.10 Lyon's Multiplier

One bit-serial multiplier implementation which uses modified booth recoding is Lyon's multiplier [Lyo76]. Each partial product is generated using three coefficient bits (y_{i+1}, y_i, y_{i-1}) and two data signals (X and 2X) according to table 3.3. In Lyon's original implementation each two bit coefficient section was identical, leading to a completely regular structure.

The arrangement of this multiplier is like the TC multiplier described in section 3.8.1, except for the partial product formation stage which is more complex. Again it is assumed that the LSBs of all input signals and the control signal arrive at the multiplier at reference time zero. For this multiplier there are CWL/2 stages numbered i=0 through i=CWL/2-1. The hardware implementation for the ith stage is shown in figure 3.19.

In the coefficient control stage the appropriate bits of the coefficient are converted to three gating signals. A is high when the recoded coefficient value is either 1 or -1,

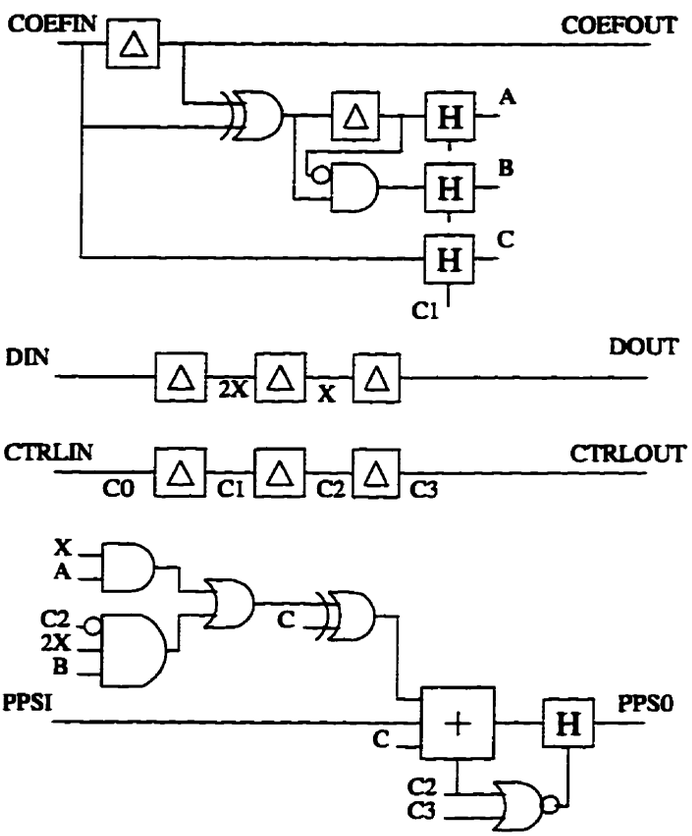


Figure 3.19: Lyon Modified Booth Recoded Bit-Serial Multiplier Stage

B is high when the recoded value is either 2 or -2 and C is high when the recoded value is negative. These control signals are combined with the X (1 times data) and 2X (2 times data) data signals to form the partial product for this stage. The 2X signal is the data signal which is valid at time two. At time one when the 2X signal is first used a zero is inserted by the C2 control signal since the data here is the MSB of the previous word. If the partial product is negative for this stage then the X,2X or 0 product is complemented and the carin input of the partial product summation adder is set high (complement and add one). The partial product generated here is right shifted by 2 bits to obtain the correct data precision for use in the next stage. Some modifications are needed on both the first and last modules of this multiplier to conform to the DFIRST wordlength requirements.

The input section must be modified to make sure that the y_{-1} coefficient bit is zero. In the multiplier module shown in figure 3.19 a zero must precede the LSB of the coefficient into the multiplier or the recoding for first module will be faulty. In DFIRST this data signal is the MSB of the previous word so a small modification to the recoding stage which gates this signal off at the correct time is required.

For a DFIRST multiplier the decimal point on the coefficient is placed just to the right of the coefficient sign bit. If the multiplier module shown in 3.19 is used as the final stage the decimal point is left of the MSB. so in the final stage the product output need only be right shifted by one bit to obtain the desired product. The nor gate which controls the operation of the right shift BLATCH is replaced by an inverter.

The resulting latency for the modified DFIRST version is 3CWL/2 + 1 from the arrival of the LSBs of the inputs to the generation of the LSB of the product.

3.11 FIRST Multiplier

The original FIRST multiplier designed by Denyer and Renshaw [DR85] is a redundant radix four recoded coefficient multiplier. The primary difference between the Lyon and FIRST multipliers lies in the partial product formation stage. For Lyon's multiplier the three coefficient bits (y_{i+1}, y_i, y_{i-1}) are recoded into three gating signals controlling the one time and two times products as well as the sign of the partial product. In the FIRST multiplier the three coefficient bits and the two data signals are passed into a single block which generates one of the following elements (2X, X, 0, not X and not 2X). If the partial product is negative the carry input of the partial product summer is tied high. The other major difference lies in the manner in which the partial product from one stage is passed on to the next stage. For Lyon's multiplier the partial product summation is right shifted by two bits and passed on to the next module. In the FIRST multiplier the un-modified partial product summation, a sign extension signal and a control signal are passed on to the next stage. The control signal selects either the partial product or the sign extension signal for use in the partial product summation of the next stage. The FIRST multiplier is described in [DR85]. The latency of this multiplier is 3*CWL/2+2.

3.12 Radix Four DFIRST multiplier

The primary multiplier type used in the DFIRST language uses a non-redundant radix four recoding on the coefficient. So for all stages, except the final stage, two bits of coefficient y_i and y_{i+1} are recoded into the digit set 0,1,2,3. In the final stage the last two bits of the coefficient are recoded into the set 0,1,-2,-1. This form

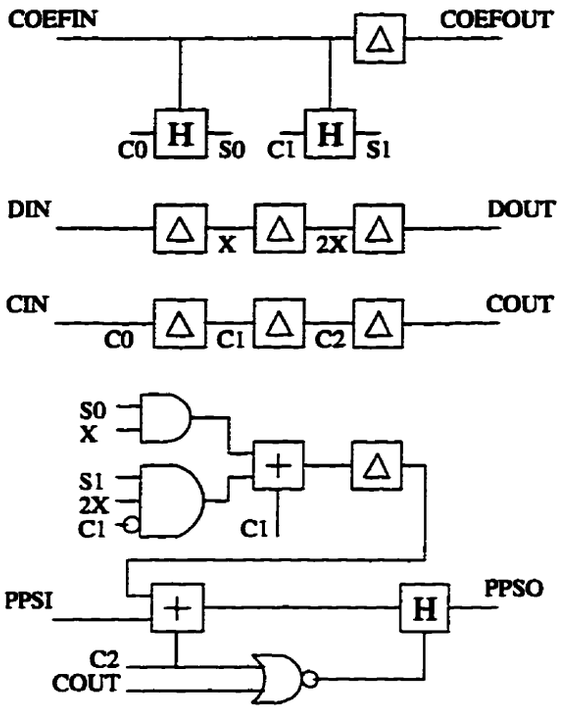


Figure 3.20: Radix Four DFIRST Internal Multiplier Section

of multiplier is not generally used because of the 3X product needed in all stages except the last. In order to form this partial product an additional adder is required. however in bit-serial the size of this adder is relatively small. The hardware for all stages but the last is shown in figure 3.20.

The two coefficient bits for this stage are loaded at two different times. The low bit of the radix four digit is loaded at time 3i and the high bit is loaded at time 3i+1. This is possible because the high bit used in conjunction with the 2X data signal. is not needed until time 3i+2. The two coefficient bits are valid at time 3i+1 and 3i+2 respectively. These two signals are used as gating signals for the X and 2X signals. If the low bit is on then X is enabled and if the high bit is on then the 2X signal is enabled. These two products are summed together at time 3i+1. with

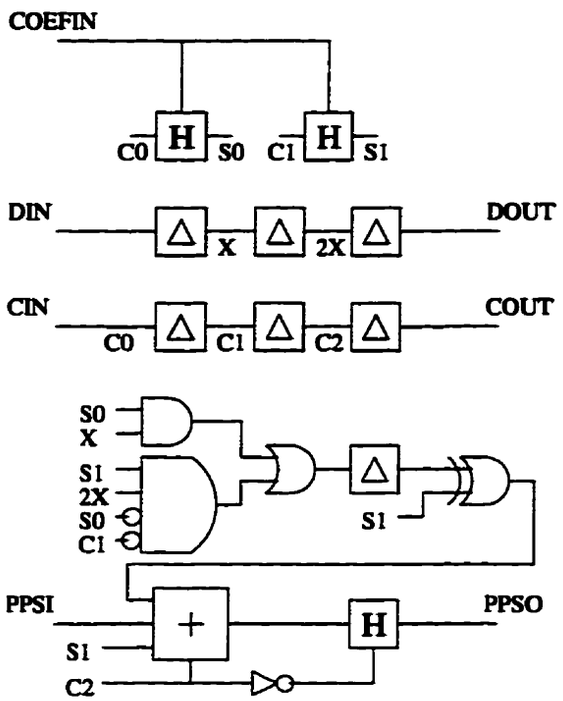


Figure 3.21: Radix Four Bit-Serial Final Multiplier Stage

a zero being inserted in the LSB of the 2X line at time 3i+1. This partial product is delayed and passed on to the partial product summer at time 3i+2. The partial product summation is right shifted by two bits and passed on to the next stage for use in the next partial product summation. In the first stage the partial product summation input is grounded and all input signals (control.data and coefficient) are valid at time reference zero.

The final radix four stage is responsible for the sign bit of the coefficient and only needs to right shift the product generated by one bit instead of two to match DFIRST precision requirements. The final stage of this multiplier is shown in figure 3.21. If the low bit of the digit is on then the partial product for this stage is either 1 or -1 depending on the value of the high bit. If only the high bit is set then the

partial product is 2X. If the high bit is set then the partial product is negated and the carry input of the partial product stage is tied high (complement and add one). If neither bit is set then the partial product for this stage is zero. The final summation is right shifted by one bit. The overall latency for this multiplier is 3CWL/2+1.

3.13 Radix Eight DFIRST Multiplier

For a radix eight non-redundant coefficient recoding it is necessary to recode each three bits of coefficient (y_i, y_{i+1}, y_{i+2}) into a single value. For a non-redundant recoding each three bits of coefficient would be converted into one of the set 0,1,2,3,4,5,6,7 where y_i is weighted by 1, y_{i+1} is weighted by 2 and y_{i+2} is weighted by 4. For the final stage containing the sign bit of the coefficient the weighting for y_{i+2} is -4. this results in a digit set of 0,1,2,3,-1,-2,-3,-4 for the final multiplier section. For radix eight recoding only CWL/3 separate partial products are formed but the hardware complexity to generate each partial product is higher than in lower radix multipliers. A radix eight non-redundant recoded multiplier module is shown in figure 3.22.

This multiplier module is very similar to the non-redundant radix four multiplier module from section 3.20. In this module three coefficient bits are stored as gating signals in the partial product formation stage, and the partial product from this stage must be right shifted by three bits to align this product for summation in the following stage. A 4X signal is required to generate the partial product which means that the data shifted left by two bits is needed. For the first stage the partial product summation input signal is tied low, resulting in one less adder for the overall multiplier. In the final stage the 4X product is subtracted instead of added in the

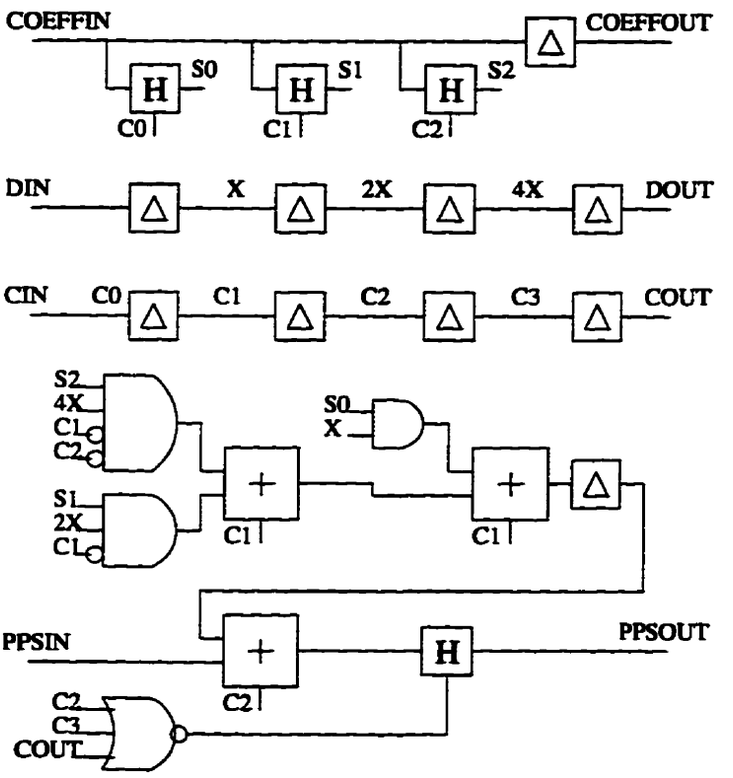


Figure 3.22: Radix Eight DFIRST Multiplier Section

partial product formation stage. This is done by complementing the 4X product line and setting the carry input signal high on the 4X + 2X adder. The final stage also only right shifts the final product by two bits in order to conform to the DFIRST conventions for decimal point placement. The resulting latency for this multiplier is 4*CWL/3+1.

3.14 Evaluation of Bit-Serial DFIRST multipliers

To select which bit-serial multiplier implementation to use the size of the multiplier, the latency and the critical path within the multiplier must be considered. Each of the different multiplier implementations exhibit different characteristics for each of these parameters. In this section the performance of these multipliers is evaluated. The TC implementation and Lyon's implementation of the multiplier are included only for comparison reasons and are not a part of the DFIRST language.

3.14.1 Hardware Complexity

One important characteristic of any operator is the integrated circuit area required to perform the operation. The size of DFIRST multipliers is measured in numbers of elements such as delays, blatches, adders and random logic gates. For a generic implementation technology it is difficult to estimate the relative sizes for each component. In order to get some measure for the size of random logic gates the number of gate inputs excluding inversions is used as an estimate for the additional gate area. The CWL parameterized size estimates for each of the bit-serial multiplier implementations are shown in table 3.4. To better understand the size relationships

Resource	TC	Lyon	FIRST	Radix 4	Radix 8
Delays	5CWL	8CWL/2	13CWL/2	8CWL/2	10CWL/3
Blatches	2CWL	4CWL/2	4CWL/2	3CWL/2	4CWL/3
Adders	CWL	CWL/2	CWL/2	CWL	CWL
Gate Inputs	2CWL	15CWL/2	26CWL/2	7CWL/2	12CWL/3
LSI Gates	57CWL	48CWL	63CWL	49CWL	44CWL
Latency	2CWL	3CWL/2+1	3CWL/2+2	3CWL/2+1	4CWL/3+1

Table 3.4: Hardware Complexity for various Bit-Serial Multipliers

of the different multipliers, the gate count can be converted to LSI logic [Log86] gate equivalents. In this technology a DFF is 5 gates. a BLATCH is 8 gates. bit-serial adders are 15 gates and each two gate inputs is 1 gate equivalent. The information in this table does not include the slight modifications to hardware complexity due to small changes in the first and last multiplier sections as these are insignificant as the CWL becomes large. Also included in the table is the latency for each multiplier implementation.

The TC multiplier has the highest latency and does not exhibit significant hardware savings to warrant the extra latency when compared to the other multipliers. Of the radix four multipliers exhibiting a latency proportional to 3CWL/2 the FIRST multiplier is clearly the largest. It is larger than Lyon's implementation and larger than the DFIRST radix four multiplier. The non-redundant multipliers use twice as many adder elements but fewer BLATCHes and fewer random gate inputs. The radix 8 implementation has a smaller latency than the radix four recoded multiplier implementations and uses fewer hardware resources than the radix four FIRST multiplier. The penalty for these advantages is the critical path of this multipier when compared to the other implementations.

Resource	TC	Lyon	FIRST	Radix 4	Radix 8
Blatches	1	1	0	(1)	0
Adders	1	1	1	1(1)	2
Gates	1	3	2(5)	1	1

Table 3.5: Bit-Serial Multiplier Critical Paths

3.14.2 Critical Path

The critical path of a digital logic circuit defines the longest logical delay path between registers or D-type flip-flops. This is a very important parameter in logic design because it dictates the maximum clock frequency at which a circuit will function correctly. The shorter the critical path the higher the potential maximum clock speed. The critical path for each of the four bit-serial multipliers discussed in the previous sections is presented in table 3.5. The critical paths are measured in terms of the maximum number of adders, muxes and random gates between latching elements. The bracketed numbers indicate alternate paths which may be the critical path depending on the technology of implementation. All random gates are treated equally regardless of type or number of inputs and inverters are not included as they can typically be absorbed into other components. No provision for loading or routing delay is included as these are technology and implementation dependent parameters. These two components of the critical path should be relatively small since each element drives only one or two loads and the bit-serial nature of the data path should yield short wiring delays.

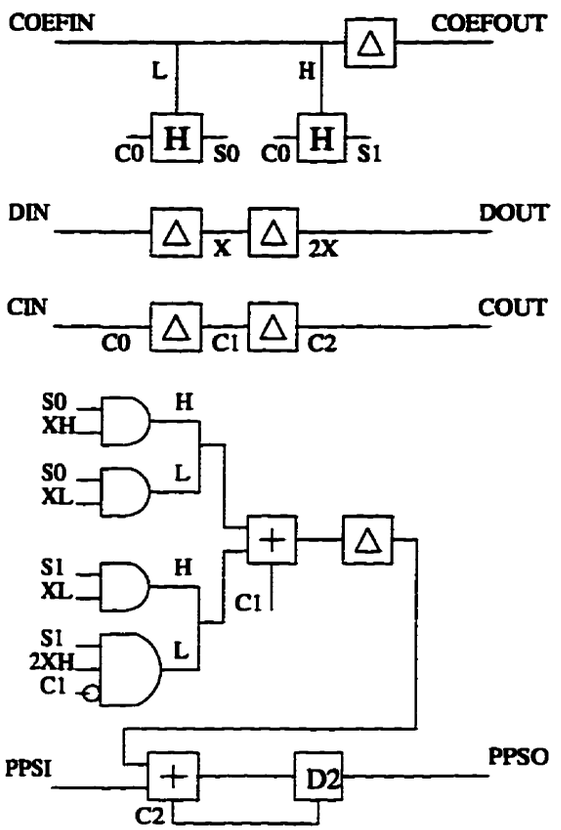


Figure 3.23: Digit-Serial (W=2) DFIRST Multiplier Module

3.15 Digit-Serial multiplier

For the DFIRST digit-serial multiplier a non-redundant radix four recoding scheme was selected for it's reduced number of partial products and it's simple structure. The hardware for a W=2 multiplier module is shown in figure 3.23.

The format of this multiplier is very similar to the non-redundant radix four multiplier described in section 3.12 except that all data operations are on W=2 digits. The two bits of coefficient storage needed for partial product formation in this section are loaded on the same clock cycle instead of the staggered load used

in the bit-serial version. All internal data path and coefficient path shift registers are W bits wide. The partial product is formed by the addition of two radix four digits representing X and 2X. The X signal is derived from the X tap of the data delay register. The low bit of the 2X signal is the high bit of the 2X data delay tap grounded at the LSD time to place a zero in the LSB of the 2X signal. The high bit of the 2X signal is the low bit of the X data delay. The partial product adder is a W bit adder and the output from this adder is delayed for summation with the partial product input. The partial product summation is right shifted by two bits to align the partial product output for use in the next stage. The right shift element only has a latency of two instead of the latency three shifter needed in bit-serial. This reduced shifter latency leads to a multiplier module which only requires a 2 bit shift register for the data and control delay chains. The overall latency of this multiplier is CWL+1 instead of the 3CWL/2+1 latency required for the bit-serial version.

For the first stage of a CWL bit multiplier the partial product input is grounded. In the final stage the two most significant coefficient bits are recoded into the set 0.1.-1.-2 and the final partial product must be right shifted by only one bit instead of two as for the other modules in the multiplier.

To extend the digit width of this multiplier beyond 2 all of the data inputs and outputs from the module must be W bits wide and the internal partial product formation. partial product summation and right shift elements must be W bits wide. The only major change comes in the coefficient storage portion of the multiplier. Only two bits of coefficient are required for each multiplier module but the number of coefficient bits arriving each clock cycle is greater than two for $W > 2$. In order to latch the coefficients into the correct places in the multiplier a two stage latching

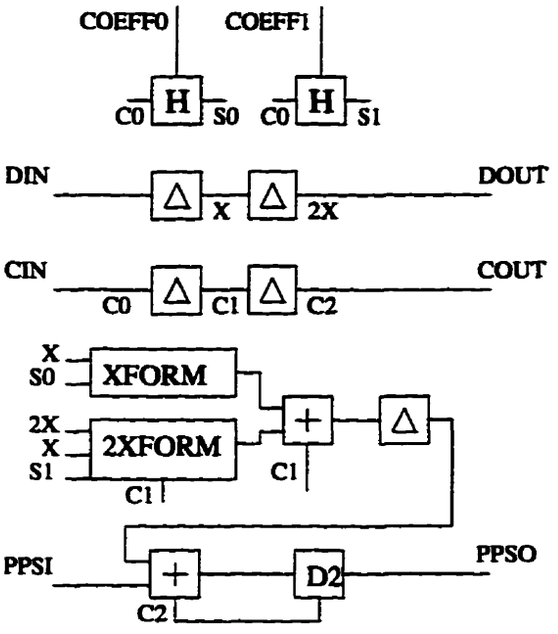


Figure 3.24: Digit-Serial Multiplier Section

scheme is used. The latches which store the coefficients within each multiplier stage are still present but their inputs come from another latching stage instead of a delayed coefficient input signal. The pre-latching stage is effectively a serial to parallel converter which converts the digit-serial input word to a CWL long parallel output word. A small difference occurs on the first two bits of parallel output. these two bits can be connected directly to the multiplier coefficient input lines. A W bit wide non-redundant radix four multiplier module is shown in figure 3.24.

3.15.1 DFIRST

Each of the different DFIRST multipliers can be accessed using the following syntax:

```
MULT [type,CWL] (cin -> cout) data,coeff -> product,deldata
```

The *CWL* parameter selects the number of coefficient bits in the multiplier. The *type* parameter selects which multiplier implementation to use. If type=0 then the original FIRST multiplier described in section 3.11 is used. if type=1 then the non-redundant radix four multiplier described in section 3.12 is used or if type=2 then the non-redundant radix eight multiplier described in section 3.13 is used. If type=3 the non-redundant radix four bit-serial multiplier is used or the arbitrary digit width multiplier described in section 3.15 is used. The digit width of the multiplier in this case is determined by the digit width of the data signals connected to the primitive. the digit width of each data input or output signal must be the same.

If type is two then the CWL must be a multiple of three(radix eight recoding) otherwise the CWL must be even(radix four recoding). The three input signals *cin*, *data* and *coeff* must be valid at the same bit time. The three output signals *cout*, *product* and *deldata* representing the output control signal, product and delayed input data respectively all have the same latency.

3.16 Parallel coefficient Multipliers

The multipliers previously described accept both the coefficient and the data input in serial format. In some cases it may be more appropriate to accept the coefficient input in the parallel signal format. Parallel coefficient inputs can be used effectively when a multiplier is shared between several different multiplication operations, where the different coefficient values are fixed at design time [NT91, Joh92]. The different coefficients can be stored in a parallel look up table and provided as parallel coefficient inputs to the multiplier when needed.

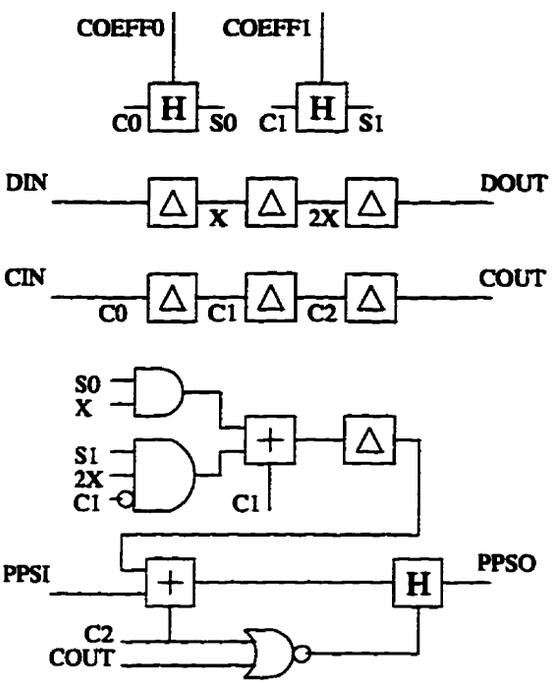


Figure 3.25: Parallel Coefficient DFIRST Multiplier Section

The conversion of the serial coefficient multiplier to the parallel coefficient multiplier is done by removing the serial coefficient delays and providing one external parallel input for each coefficient bit of each multiplier section. The parallel coefficient multiplier is derived from the non-redundant radix four multiplier described in section 3.12. The resulting hardware for a parallel coefficient multiplier module is shown in figure 3.25.

The latency for this parallel coefficient multiplier is the same as the serial coefficient version. 3*CWL/2 +1 for the bit-serial part and CWL+1 for the digit serial parts. Each bit of the parallel coefficient can be set at time zero, the coefficient storage section of the multiplier will load the coefficient bits as needed.

3.16.1 DFIRST

The DFIRST call to instantiate a parallel coeffic

```
PMULT [CWL] (cin->cout) data,coeff0 THRO
```

All of the input signals to the multiplier mu
one multiplier latency later each of the three out
of the multiplier is dictated by the digit width
to the multiplier, each of these signals must ha
significant bit of the coefficient is coeff0.

3.16.2 Extended Multiply

The multipliers discussed to this point all imple
data signal by some number between plus and
number larger than one is required then a left s
the data path before the fractional multiply. F
latency for the multiplication is 3*CWL/2 + 2 b
can be obtained by incorporating the left shift
This is one way to make use of the negative la
using left shift operators as discussed in section
multiply for an arbitrary decimal point placeme
two portions. The first portion, or upper word, is
standard serial multiplier primitive and the seco
from the bits thrown away by the right shift op
product summation. Each of these bits are put t

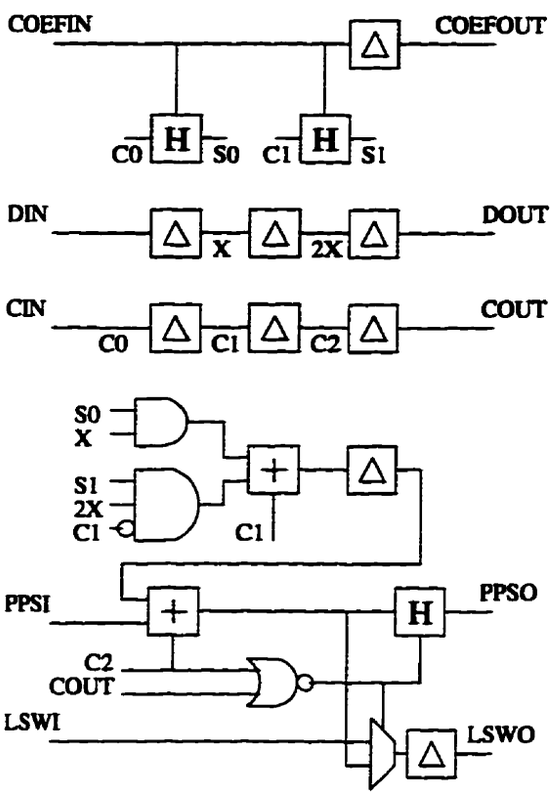


Figure 3.26: Extended Multiplier Section

and the appropriate bit of this data is used as the LSB of the output. The multiplier block to generate and accumulate both the upper and lower products is shown in figure 3.26.

The first stage of this multiplier was modified to remove the partial product summation stage completely and pass the partial product for the first stage directly to the right shift and least significant word accumulation stage. The removal of the pipelining stage speeds up the multiplier by one clock cycle.

In the final multiplier stage the product is not right shifted by one bit as in other bit-serial multipliers. This effectively left shifts the the product coming out of the

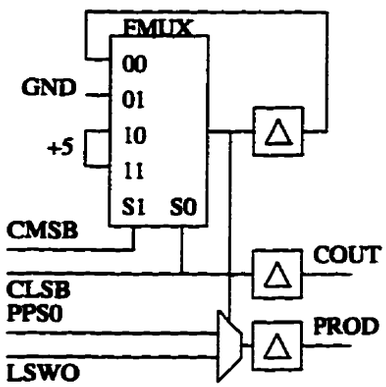


Figure 3.27: Final Product Generation Element for DFIRST Extended Multiply

extended multiplier by one bit when compared to the other bit-serial multipliers. In place of the shifting component a DFF is used to pipeline the most significant word output. In order to combine the LSW and MSW into the final product output one formatting stage is needed as shown in figure 3.27.

The output from the FMUX (four to one mux) is used to select either the LSW or the MSW to form the final product. The *cmsb* signal indicates the LSB time for the MSW output from the multiplier. The *clsb* signal indicates the valid time for the bit of output from the multiplier which is to form the LSB of the final product. The time for this signal is 3*CWL/2-shift. From the arrival of the *clsb* signal until the *cmsb* signal is valid the LSW output forms the product for the multiplier. after the arrival of the *cmsb* signal the MSW output forms the output from the multiplier. A DFF element is added at the multiplexor output to pipeline the operator. In order to save some hardware the WFORM portion of the multiplier can be discarded if the left shift value is one, because the MSW output from the multiplier already contains a left shift by one. The final latency for the extended multiplier primitive is 3*CWL/2+1-shift where the shift value is a positive non-zero integer. The data

signal on the input to the extended multiply requires 2+shift most significant end sign extensions or the product will overflow.

3.16.3 DFIRST

The DFIRST call to an extended multiply primitive is:

```
MULTEX [CWL,shift] (c0->cout) data,coeff -> product, deldata
```

The CWL parameter indicates the CWL for the multiplier and must be an even integer as in all other radix four multiplier DFIRST multipliers. All of the multiplier inputs must be valid at the same clock cycle and the *cout* and *product* outputs are generated 3*CWL/2-shift+1 clock cycles later. The *deldata* output has a latency of 3*CWL/2-1. This primitive is not available in digit-serial.

3.16.4 Constant Multiplication

All previously discussed DFIRST multiplier implementations perform the multiplication of an arbitrary data signal by an arbitrary coefficient. In many instances multiplications by a fixed coefficient are required [Jac89, NT91, Joh92], here a general coefficient multiplier could be used with the coefficient input being set to the desired constant but a smaller fixed coefficient multiplier could also be used.

The fixed coefficient multiplier uses a series of shifts and add/subtract operations to obtain the final product. For this multiplication a Canonic Signed Digit (CSD) [LEL91] recoding scheme is used on the coefficient. In CSD each bit of a number is either positive or negative so a variety of different implementations are possible for most integers. In practice a CSD number has at most half of the CWL bits set and

the other half are zero. With only half of the bits set, at most, only half as many adders are required to accumulate the final product.

To generate a fixed coefficient multiplier three different pieces of information are needed, the precision or CWL of the multiplier and two constants representing the CSD value of the coefficient for the multiplication. The coefficient is broken down into a value portion which indicates which bits of the coefficient are set and a sign portion which indicates which set bits are negative. The strategy to implement the multiplication is to add or subtract the data signal from the accumulated partial product according to the coefficient being implemented. The result of each addition/subtraction operation is right shifted to align the adder output for use with the next data input. The resulting hardware for a 15/32 and a 27/64 constant multiplication is shown in figure 3.28. The 15/32 constant is implemented as (16-1)/32 and the 27/64 constant is implemented as (32-4-1)/64. The SHIFTMULT implementation of 15/32 can be written as $((din >> 4) + din) >> 1$. The SHIFTMULT implementation of 27/64 can be written as $((((din >> 2) + din) >> 3) + din) >> 1$.

3.16.5 DFIRST

The DFIRST call to a constant multiplication primitive is:

```
SHIFTMULT [CWL,value,sign] (cin -> cout) data -> product,del
```

The *CWL* parameter sets the number of bits present in the fixed coefficient, the *value* is an unsigned integer representing which bits are set in the coefficient and the *sign* is an unsigned integer representing which coefficient bits are negative. The effective value for the coefficient can be determined using equation 3.4:

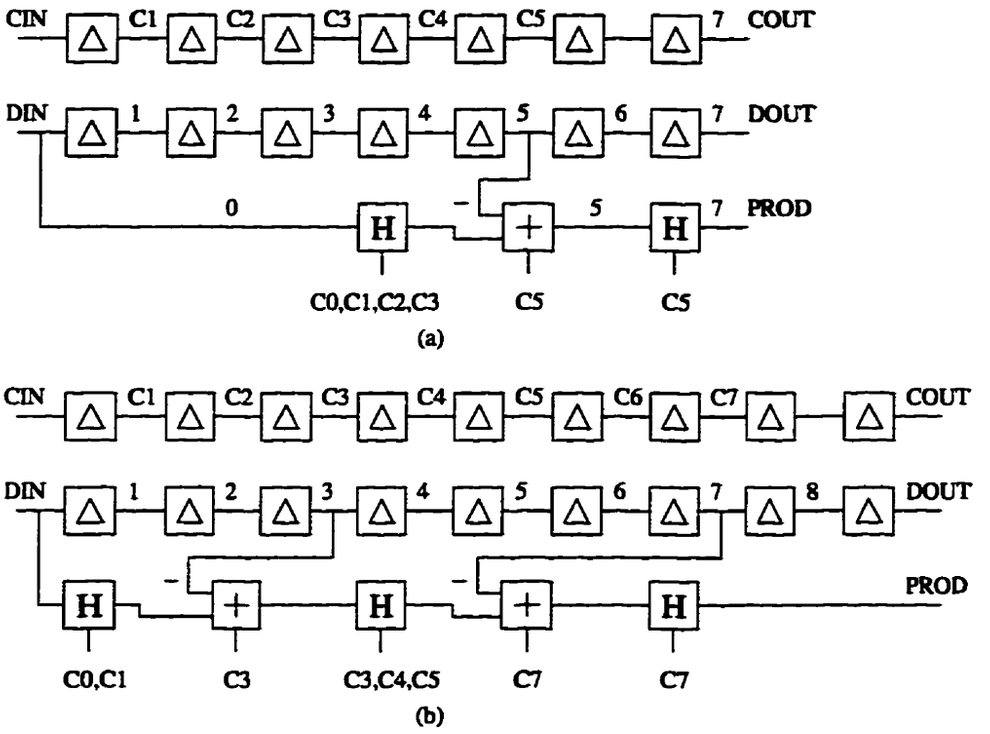


Figure 3.28: Bit-Serial Constant Multipliers (a) 15/32 (b) 27/64

$$P = \sum_{i=0}^{CWL/2-1} (y_{2i-1} + y_{2i} - 2y_{2i+1}) * X * 4^i \qquad (3.4)$$

where only bits present in value are used in *sign* and *sign* < *value*.

For the SHIFTMULT primitive there must be at least one positive bit in the fixed coefficient. This restriction forces at least one input on an adder to be positive, since both inputs on a single adder cannot be subtracted. The *cin* signal and the *data* signal are valid at the same bit time and the latency for the three output signals is the CWL plus the number of bits which are set in the coefficient. The DFIRST calls representing the two multipliers shown in figure 3.28 are:

```
SHIFTMULT [5,17,1] (c0) din -> prod,deldata
SHIFTMULT [6,37,5] (c0) din -> prod,deldata
```

The hardware for a SHIFTMULT operator can be generated using the DFIRST primitives for data delays, additions and right shifting, but the SHIFTMULT primitive is much simpler to use. The circuit designer must only select which constants to use and not be concerned with the exact details of each implementation. The SHIFTMULT primitive is only available in bit-serial.

3.17 Controlgenerator

The CONTROLGENERATOR DFIRST primitive, generates all control signals required within a given DFIRST design and is present in all DFIRST designs. The commands which make up a CONTROLGENERATOR are CYCLEs and EVENTs. The CYCLE command takes a single parameter which describes the division factor

by which the previous CYCLE output is divided. The first CYCLE output is a data framing signal which is high for only one clock cycle per period of the CYCLE (one data word). All subsequent CYCLE commands are steering control signals which are high or low for one or more full data words. Each EVENT command synchronizes one input signal to one cycle in the CONTROLGENERATOR. Once an EVENT is detected on the input signal the synchronized output control signal appears as one full cycle of the CYCLE command just previous to the EVENT command. The first output from a CONTROLGENERATOR is retained from the FIRST language but is not used in DFIRST. this signal is connected to NC. The instantiation of a DFIRST CONTROLGENERATOR is shown below. The resulting control signals are shown in figure 3.29. The hardware implementation of the DFIRST CONTROLGENERATOR is fully discussed in [NT91].

```
CONTROLGENERATOR (ein -> NC, c0, c00, c10, eout)
     CYCLE[16]
     CYCLE[2]
     CYCLE[2]
     EVENT
ENDCONTROLGENERATOR
```

3.18 Other DFIRST Primitives

The DFIRST language also includes a division (DIVIDE) operator, a square-root operator (SQRT). conditional operators which compare to serial signals and generate steering logic depending on the result of the comparison and an ORDER primitive

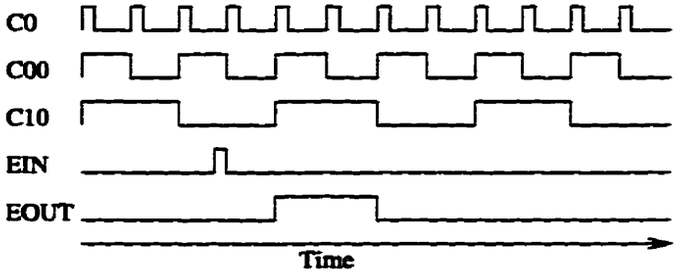


Figure 3.29: Control Signals from Sample CONTROLGENERATOR

which sorts two serial input signals into the larger and smaller values. Each of these primitives is only available in the bit-serial format and are described in [Gra92a]

3.19 Summary

DFIRST primitives are parameterized in terms of precision, digit-width and data wordlength. The DFIRST language primitives include right and left shifters, adders, subtracters, parallel to serial and serial to parallel converters and multipliers. Each primitive is built up from simple logic gates, multiplexors, BLATCHes and delay elements. More complex primitives, such as multipliers, are also built up from the smaller serial adder and shifter components. The digit-serial multiplier uses a non-redundant radix four recoding on the coefficient bits to reduce circuit area and latency and is available in both serial and parallel coefficient versions. The non-redundant radix eight multiplier, the EXMULT (extended dynamic range coefficients) and the CSD fixed coefficient SHIFTMULT primitives are only available in the bit-serial (W=1) form.

Chapter 4

TRANS Hardware Compiler

A high level hardware description language is useful for describing circuits and performing high level circuit simulations but one more step is required to generate a final implementation. The high level language must be converted to a lower level gate or transistor level format to implement a high level circuit description on a real device. In this chapter the TRANS hardware compiler is discussed as it pertains to the DFIRST register transfer level language. The TRANS compiler converts a DFIRST netlist into a format suitable for use with full custom, semi-custom or FPGA devices. In this chapter the netlist transformations and reductions most appropriate for the bit-serial and digit-serial hardware description language will be discussed. The output formats used will concentrate on the XILINX [XIL94] and ACTEL [ACT89] FPGA devices.

4.1 ASIC Architectures

In order to convert a high level language to an implementation in an application specific integrated circuit it is important to understand the nature of the implementation device. Each implementation technology exhibits different strengths and weaknesses which must be exploited or avoided in order to obtain a good implementation. In this section the architectural features of XILINX and ACTEL field programmable gate arrays will be discussed. In particular a short overview of technology resources

for internal logic, device input-output and routing will be presented. The abilities of vendor provided software will also be discussed in order to reveal what is required from TRANS to obtain a more efficient circuit in each technology.

4.1.1 XILINX FPGAs

XILINX FPGAs use static RAM to store the configuration for each routing element and configurable block within the device. The configuration information for a circuit can be loaded from a PC serial port or on power-up from serial or parallel ROMs [XIL94]. Once the device is configured, the functionality of the FPGA does not change until a new program is loaded. The reconfigurability of XILINX FPGAs makes them very useful for proto-typing digital circuits and for use in applications where the requirements of the ASIC may change over time. The routing resources [XIL94] are divided into vertical and horizontal routing channels with a switching matrix being used to connect signals together where the routing channels cross. Local nearest neighbor connections are also present.

The configurable logic elements are divided into two categories. Input/Output Blocks (IOBs) and Configurable Logic Blocks (CLBs). The IOBs are used to communicate with devices external to the XILINX FPGA and the CLBS are used to perform internal digital logic functions. The form for a 4000 series IOB is shown in figure 4.1. This block is configurable as either an input, output or bidirectional I/O pad, in addition each input or output signal can be latched using an edge sensitive DFF as needed.

The 4000 series CLB is a 9 input, 4 output logic block which contains 3 Look-Up-Tables (LUT) and two DFFs. The 4000 series CLB can perform two arbitrary

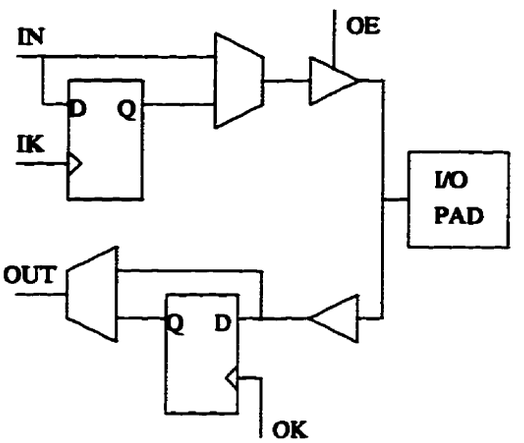


Figure 4.1: XILINX I/O Block Resources

functions of four variables or a single five input function. A very useful feature present in the 4000 series CLB is the ability to convert the two 4 input LUTs into RAM. In this way each CLB can implement a 16x2 or a 32x1 RAM element. The resources contained in a 4000 series CLB are shown in figure 4.2.

XILINX Software

The XILINX implementation software converts a netlist specified in terms of generic logic elements to a final implementation. The job of mapping generic logic elements to CLBs is automatically performed by XILINX tools. Recently other CLB mapping algorithms [Neo94] have performed better than the XILINX provided software but the CLB mapping problem is not addressed within TRANS. The XILINX software also performs the placement and routing [XIL94] operation on the CLBs and IOBs which make up a design. After this step the circuit is ready to load onto the device and be tested in a physical circuit.

The XILINX software performs most of the operations necessary to obtain a good

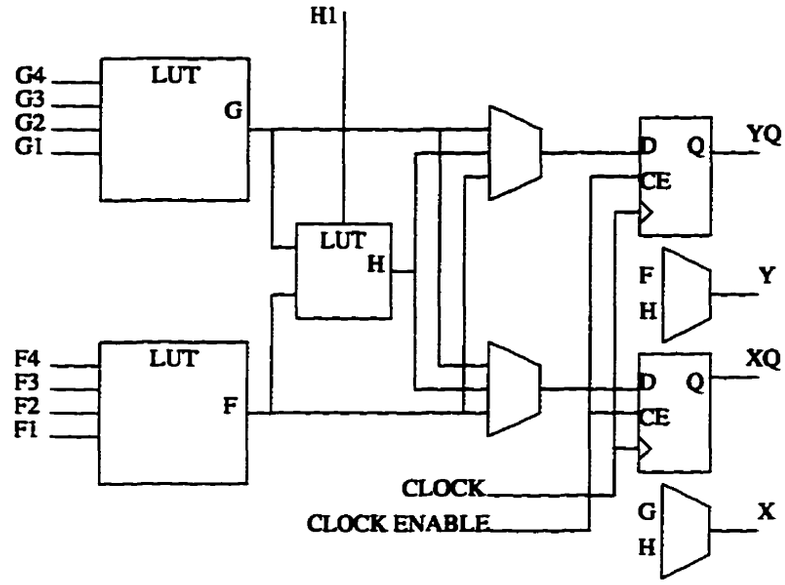


Figure 4.2: Simplified Block Diagram of a 4000 Series XILINX CLB

circuit implementation from a generic gate level netlist. TRANS performs some useful circuit optimizations to improve hardware efficiency in XILINX devices. The redundant hardware removal operation is presented in section 4.5, better utilization of the I/O ring DFFs is discussed in section 4.6 and the use of the 4000 series RAM to reduce shift register size is discussed in section 4.6.2.

4.1.2 ACTEL FPGAs

ACTEL FPGAs are one time programmable devices which use anti-fuse technology [ACT89]. An anti-fuse is a component which before programming exhibits a low resistance and can be 'blown' to become an open circuit. In an unprogrammed ACTEL device all the anti-fuses are present and the circuit is implemented by blowing appropriate elements to create the interconnections desired. The internal logic elements are multiplexor based cells and the I/O pads can be configured to be in-

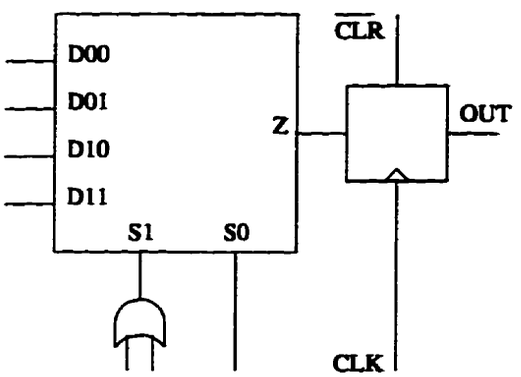


Figure 4.3: ACTEL S-module

put/outputs or bidirectional pads with optional level sensitive latches for input and output signals.

Two different types of cells are present in the ACTEL [ACT89] technology. Half of the cells are C-modules and the other half are sequential cells or S-modules. The S-module combines a multiplexor type cell with a latching element (level or edge sensitive) as shown in figure 4.3.

4.1.3 ACTEL Software

The ACTEL software performs the placement and routing operation but the input netlist must be in ACTEL specific cells. There is no method of converting a circuit specified in generic logic to a circuit which uses ACTEL specific cells other than as a one to one mapping. This operation is performed by the hardware mapping facility of TRANS as explained in section 4.6. Another operation which can be performed to reduce circuit size on ACTEL devices is to remove redundant hardware as described in section 4.5.

4.2 TRANS overview

The TRANS [Gra92c] hardware compiler must perform several operations upon a DFIRST netlist in order to obtain an efficient implementation in the target technology. These transformations begin with the synthesis of the RTL components of the DFIRST language into a generic gate level circuit containing only simple gates (AND. OR. NAND. etc.). D-type flip-flops and input/output pads. Each of the DFIRST primitives is compiled (as discussed in chapter 3) from gate level components and larger primitives provided in a technology library. The DFIRST library contains elements ranging from BLATCHes and bit-serial adders to the bit-serial multiplier sections presented in section 3.8.

The generic gate level implementations generated by the assembly portion of TRANS are not suitable for use with every output technology. Each implementation device whether it be full custom. semi-custom or an FPGA has a cell library which must be used to construct each circuit. The cell library can be a set of multi-gate elements such as two to one multiplexors as in the case of ACTEL FPGAs. or each technology element can be a look up table (LUT) as in XILINX FPGAs. In either case the generic logic elements must be converted to technology specific elements in order to obtain a more efficient implementation. The XILINX LUT mapping procedure is performed by the XILINX software supplied by XILINX. but a cell based mapper is needed for the ACTEL architectures. In order to make the TRANS mapping operation as flexible as possible a rule based system is used which loads an external rule data base file to control the mapping procedure. This rule file is generated once for each technology and can be changed in the light of technology

updates or new cell libraries.

Another mapping which can be used for serial circuits generated by the DFIRST language, is RAM mapping. In some technologies, such as XILINX 4000 series FPGAs, it is possible to implement RAM on the FPGA [XIL94]. The XILINX RAM is completely configurable in terms of RAM depth and RAM width. Using this on-chip RAM together with an addressing unit, an area efficient shift register can be implemented [XIL94]. In order to further reduce implementation size the addressing units of different serial RAMs can be shared. In order to do this effectively size estimates in the target library for serial RAMs and addressing units is needed.

Several other netlist utilities are available within TRANS including, netlist flattening to remove all hierarchical elements, removing redundant hardware elements to reduce circuit size, performing a gate count to estimate size, and computing the critical path length of a circuit. The delay calculation operation requires a data file which describes the expected delay time for each technology element with respect to output loading. The overall flow of the TRANS compiler and the additional information required to perform each step in the process is shown in figure 4.4.

4.3 Technology Files

Each supported TRANS input or output language format has some technology specific primitives. For the DFIRST language these primitives include BLATCHes, FADDs, and bit-serial multiplier sections. For the ACTEL FPGA technology the primitives are multi-gate cells and special purpose input/output pads. Each of these technology specific components must be converted to a format which can be used in

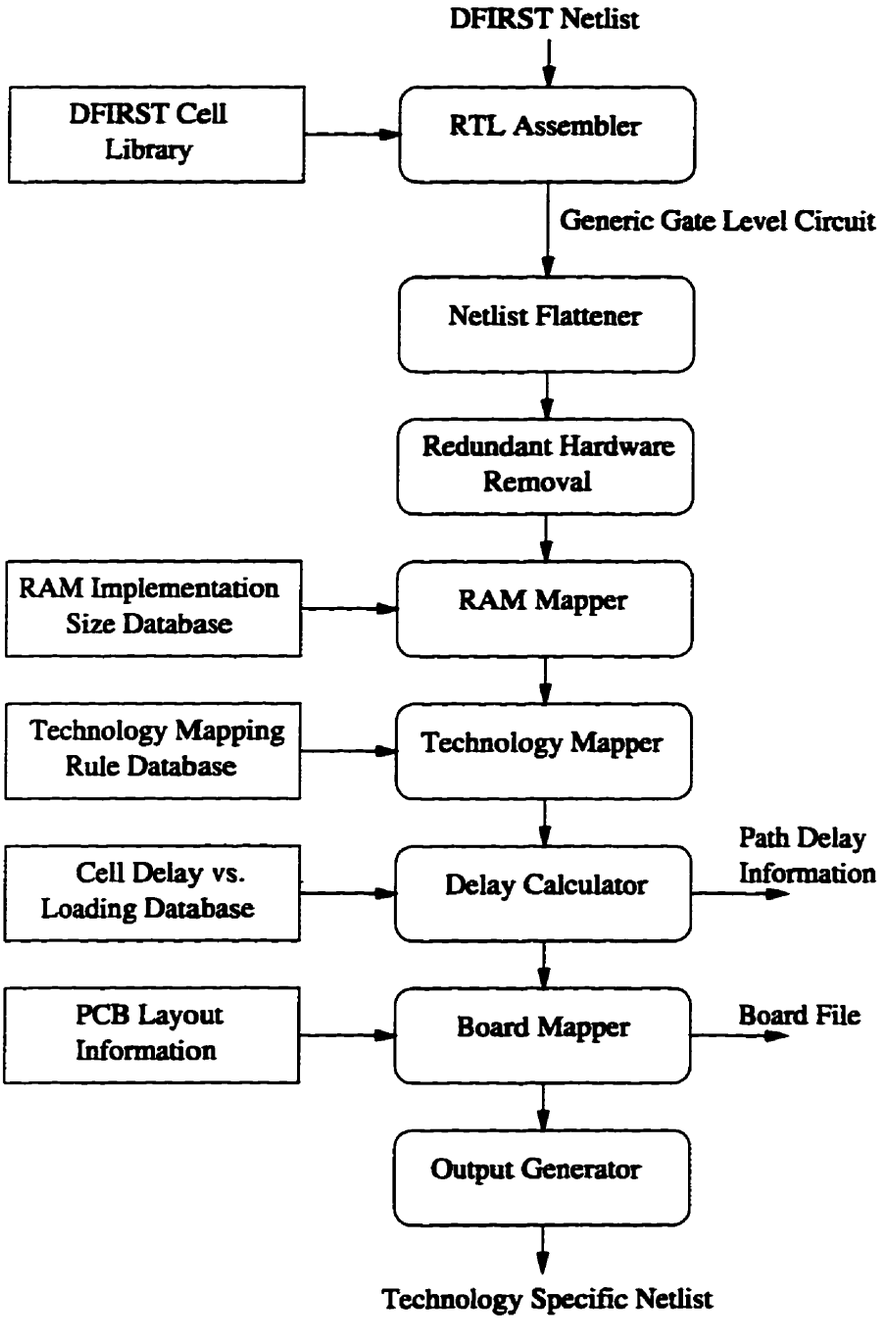


Figure 4.4: TRANS Hardware Compiler Flow Diagram

```
MUX ( (A0 A1 SE) (O) ) @ 1    mux.log
ADDER ( (A B C) (S CO) ) @ 2  adder.log
INDFF ( (D:EXT CLK) (Q) )   indff.log
OUTDFF ( (D CLK) (Q:EXT) )   outdff.log
```

Figure 4.5: Sample TRANS Technology File

other technologies. This conversion can be performed using technology files or 'tech' files.

A tech file is used to define several characteristics for each primitive in the technology database. A tech file is used to define the pinout for any element in the library including direction (input/output) and whether the pin is connected to an I/O pad, making that signal external to an IC. A tech file also defines the gate count for each macro in the library which can be used to estimate the size of a circuit in that technology. Finally the tech file provides the location of a file where a netlist describing the function of each library element can be found. This netlist contains only generic logic elements. A sample technology file describing a 2-1 Multiplexor (MUX) element, a full adder (ADDER) and two chip I/O cells (INDFF and OUTDFF) is shown in figure 4.5.

The signals contained in the first set of brackets are macro inputs and the signals in the second set of brackets are output signals. A ":EXT" modifier on a signal is used to identify that signal as being connected to an I/O pad. The number following the @ symbol, if present, defines the gate count for that library element and can be any integer representing the cell count or area of the primitive. The cell count for any primitive defaults to zero. The string at the end of each macro description identifies a file which contains a description of the cell using only generic logic elements.

4.4 Netlist Flattening

After reading in a DFIRST file the circuit is an interconnection of user defined operators, DFIRST technology specific elements and generic logic elements. The operation of netlist flattening converts a circuit containing several levels of hierarchy into a circuit which contains only generic logic and output library elements. This operation removes partitions which were introduced to provide modularity and make the design procedure simpler. After flattening the design optimization steps can operate on the whole design resulting in improved performance at each optimization step.

4.5 Redundant Hardware

One optimization which is suitable for use with any target technology, is to remove any redundant hardware [Gra92c] present in the design. redundant hardware often results during the creation of large circuits in which several levels of hierarchy are used to partition the design into manageable pieces. As well, the assembly of parameterized operators, as performed by TRANS, may introduce other levels of hierarchy into the design. These large elements may often have components which are repeated resulting in wasted resources.

If two identical components have identical inputs but output signals which are connected to different parts of the circuit, one of the elements is redundant. The redundant element can be removed and the load(s) which were driven by the removed part are merged onto the output of the remaining element. This operation increases the loading on the remaining component and thus cannot be implemented without

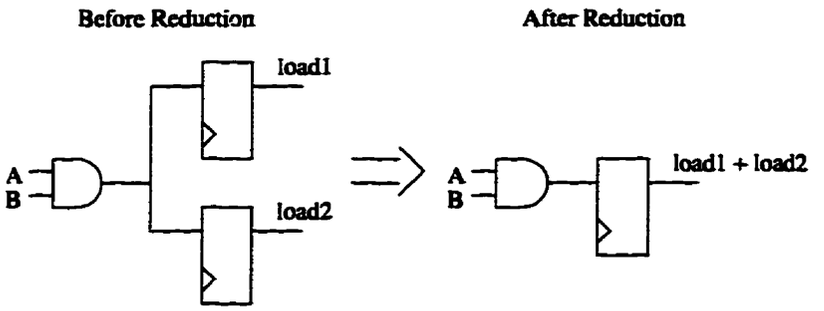


Figure 4.6: Redundant Hardware Removal Operation Saving One DFF

penalty. This loading factor can be controlled through a user defined parameter (FANOUT) which is used to limit the maximum fanout allowed for any node. If a hardware removal operation results in this limit being violated the reduction is aborted and no change is made to the circuit. A hardware reduction operation which removes one redundant flip-flop from the circuit is shown in figure 4.6.

Another form of redundant logic is loadless logic which occurs when a signal is generated but is not used. Several of the DFIRST primitives generate several output signals. If one or more of these signals is not used some loadless hardware is introduced. TRANS can be directed to remove this form of redundant logic.

4.6 Hardware Mapping

To obtain an efficient implementation, the generic logic used to implement the DFIRST primitives must be mapped into the cells available within a target technology. For the XILINX FPGA this process is carried out by XILINX supplied software. However for the ACTEL FPGAs the generic gates of the flattened serial design must be mapped to a unique set of cells available in the ACTEL architecture. This process is carried out using a rule based mapping optimization [Gra92c].

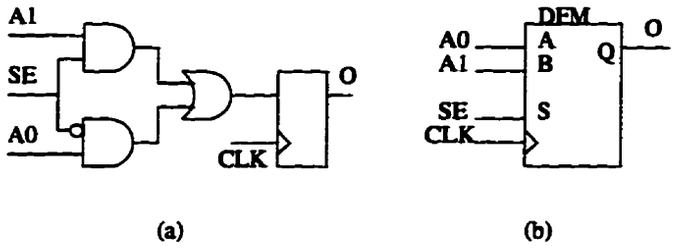


Figure 4.7: Mapping Source(a) and Target(b) for DFM ACTEL Cell

A mapping rule consists of a source logical arrangement to be found within a design, and a target implementation which is used as a replacement for all the hardware identified as a match to the source arrangement. A sample rule is shown in figure 4.7. In this example the logic required to implement a two to one multiplexer driving a D type flip-flop is the source arrangement to be located. Each instance of the source which is found within the design is replaced by the cell DFM, which is a technology specific hardware element. For this example the starting logical arrangement uses six ACTEL cells and the mapped hardware uses only one ACTEL cell.

The mapping for a full adder element into ACTEL logic elements is shown in figure 4.8. Here the original logic for the full adder is replaced by a FA1B ACTEL cell and two inverters to maintain functionality. Both implementations use four ACTEL1 cells but the inverters can be more easily used in subsequent mapping operations, possibly further reducing the size of the full adder implementation.

For a mapping rule to be executed an exact match for the source logical arrangement must be found. This match not only includes functionality but also implementation style. If the two to one multiplexor from figure 4.7a were implemented in the design using NAND gates, this reduction rule would fail to execute. To avoid this difficulty all source arrangements are described in terms of non-inverting logical

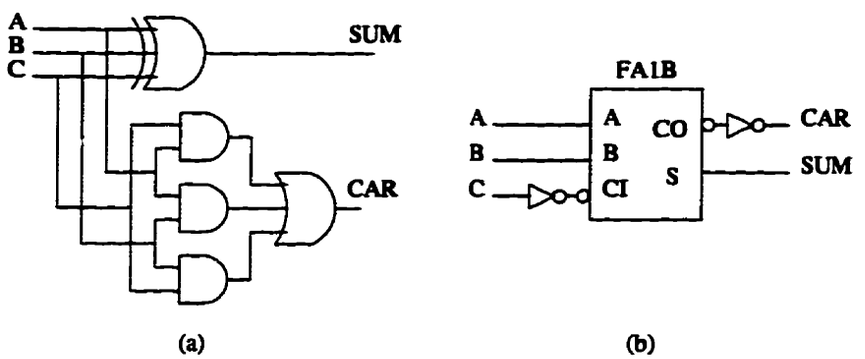


Figure 4.8: Source(a) and Target(b) for ACTEL Full Adder Implementation

elements (AND, OR). At the beginning of the mapping procedure a series of rules are applied which convert inverting logical elements (NAND, NOR) to true logical elements and inverters. Cascaded inverters and demorgan equivalent circuits are used to reduce the circuit further. After this process the entire circuit is described in terms of AND, OR, XOR and INVERTER logic gates. This circuit can be mapped effectively using the mapping rules defined in terms of these four logical elements.

The remainder of the rule file consists of a collection of mapping rules. The rules containing the largest amount of logic are executed before rules containing smaller logical blocks. Precedence is given to rules which can add additional generic elements to the circuit, such as the mapping shown in figure 4.8. By adding the additional generic elements to the circuit early in the mapping procedure the opportunities to combine this logic into other mapping operations is maximized.

4.6.1 Cell Internal Connections

One difficulty which can arise when using rule based mapping is the presence of internally connected nodes within an otherwise exact match to the source logical

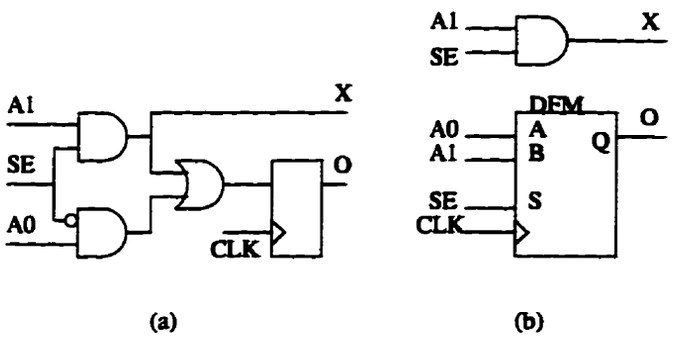


Figure 4.9: Source(a) and Target(b) for DFM Cell with Internal Connection

arrangement. Consider the DFM shown in figure 4.9. the internal node marked X is used elsewhere in the design and the mapping cannot proceed unless X is regenerated. For the circuit shown this means performing the mapping as usual and adding one additional AND gate to drive node X. The number of ACTEL cells for this logical block goes from six before the mapping to two after the mapping shown in figure 4.9. Since a mapping which extracts internal nodes adds additional hardware to the circuit, it is possible that the circuit size after mapping to be larger than the original circuit size. If this occurs no replacement operation is performed leaving the circuit unchanged.

4.6.2 XILINX Mappings

The hardware mapping operation is not needed for XILINX FPGAs since the XILINX software already performs logic mapping to CLBs. However hardware mapping may be used to implement some special purpose reductions particular to DFIRST and XILINX FPGAs. One mapping which can be used within the XILINX architecture is to convert logic to a form which uses the clock enable pins available on DFFs within XILINX CLBs. The BLATCH DFIRST primitive can be implemented

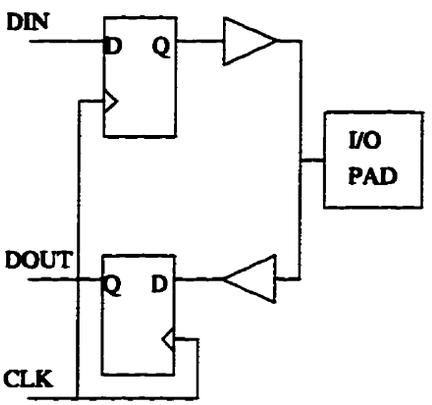


Figure 4.10: Two Bit Shift Register Implemented in one XILINX IOB

directly using a clock enabled flip-flop. By doing this. more of the LUT within a XILINX CLB becomes available to implement other logic within the design. This mapping becomes particularly useful in the 4000 series devices where the LUT output and the flip-flop output from a CLB can be used independently.

Another feature which can be exploited using hardware mapping is the DFFs within the I/O ring of XILINX FPGAs. Each IOB contains two DFFs which can be configured as a two bit shift register as shown in figure 4.10. In this configuration the external connection to the pad must be left unconnected at the board level or the shift register will not function correctly. Since serial designs make extensive use of shift registers. this feature is useful for fitting large designs onto a particular XILINX device. The mapping algorithm finds two bit shift registers and converts them to the IOB representation. To control the amount of flip-flops mapped into the IO ring the user specifies how many IOBs can be used in this manner. Each such IOB frees up two CLB DFF elements which can be used for other operations within the circuit.

4.7 RAM mapping

Serial algorithms implemented in DFIRST typically contain a large number of shift registers for variable storage and timing synchronization. This storage can be implemented using simple shift register chains made up of DFF elements. but a more efficient means of implementing these registers can be used if serial RAM elements are available in the target library.

The RAM based implementation of an N bit shift register requires an N bit deep. 1 bit wide RAM. a modulus N counter to control the address of the RAM. and one final output DFF. During a single clock cycle the data from one RAM location will be written to the output flip-flop. then the next input value to the shift register will be stored at the same address and the counter controlling the address pins of the memory will be incremented. An eight bit shift register implemented using RAM and the corresponding timing diagram is shown in figure 4.11.

The first falling edge (point 1 in figure 4.11) increments the eight bit counter to an address value of n. With the address at location n the RAM begins reading the old data at this location. At the following rising edge (point 2) the data from the RAM is latched into the output DFF element and a new input is presented on the DIN signal. At this point the RAM is writing the new value to location n. overwriting the old value there. At the next falling edge (point 3) the RAM stops writing and the address is incremented to location n+1 and the process repeats.

This implementation of a shift register uses one N bit deep serial RAM. one output DFF and one N bit counter which contains ln(2)N DFFs and some combinational logic. Since only a single input and output are present on the serial RAM. no internal

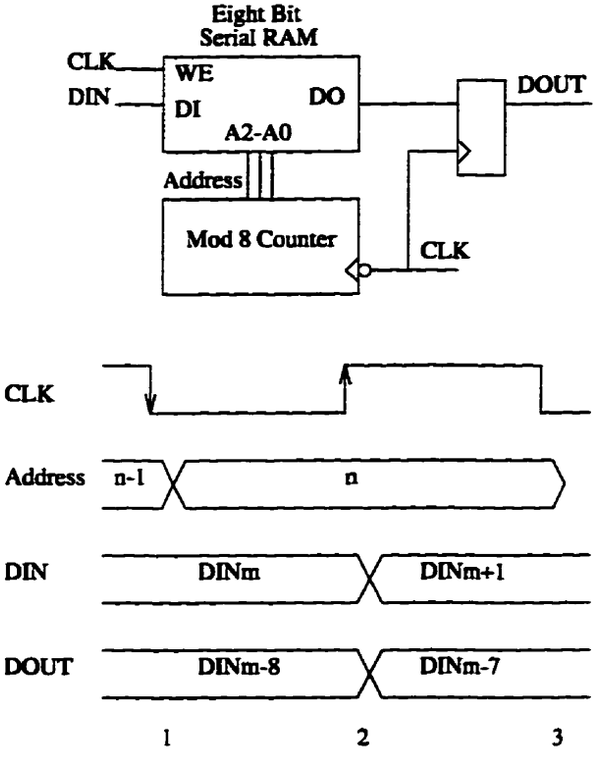


Figure 4.11: RAM Implementation of Eight Bit Shift Register with Timing

values of a shift register implemented in this manner are available. Therefore this mapping can only be used on contiguous (no internal points used) shift registers.

4.7.1 Shared Addressing Units

In many DFIRST circuits several different contiguous Shift Registers (SR) of varying lengths may be present. Each of these shift registers may be implemented directly, with each shift register requiring one RAM and one counter. However a more area efficient circuit is possible if the addressing units are shared between different RAM units. Since several different shift register lengths may be present in a given circuit and a mod N counter must only drive N deep memories, it may be necessary to divide the implementation of some SRs into a combination of RAM and DFF shift registers. For example a nine bit shift register can be implemented using an 8 bit RAM SR as described in figure 4.11 and one additional DFF element.

In order to determine a good address sharing strategy the size of all relevant components in the implementation technology must be known. For the XILINX 4000 series FPGAs the sizes for N bit RAMs, N bit DFF shift registers and N bit counters is shown in table 4.1. Using this information a good mixture of RAM SRs, counters and DFF SRs can be obtained.

The first step of the RAM mapping algorithm is to go through the circuit, which must be flattened, to find all contiguous shift register lengths greater than a designer specified minimum length (LENGTH). The identified SRs are sorted from the shortest to the longest. Starting from the shortest SR, two implementations for the SR are generated. The first implementation uses an N bit serial memory and an N bit serial RAM. The second implementation uses the largest previously generated M bit

Length	RAM	Addressing Unit	Shift Register
1	0.5	1.0	0.5
2	0.5	1.0	1.0
4	0.5	1.0	2.0
8	0.5	1.5	4.0
16	0.5	2.5	8.0
32	1.0	4.0	16.0
64	3.0	5.0	32.0
128	7.0	8.0	64.0
256	12.0	12.0	128.0

Table 4.1: Cost in CLBs for Several Lengths of RAMs.
Addressing Units and Shift Registers

addressing unit ($M < N$). an M bit serial RAM and N-M DFF elements. The size of the two implementations is compared and the smaller of the two is chosen as the implementation for this SR. The algorithm proceeds in this way until all SRs have been mapped. Finally the DFFs used to implement the original shift register(s) are removed from the circuit.

One drawback to a shared addressing unit architecture is the potential loading on each addressing unit. An addressing unit which is driving too many serial RAMs will have a degraded performance in terms of the transition times on the addressing lines. this may have an impact on the critical path of the overall circuit. In order to account for this a user defined parameter (ADRLOAD) can be used to set the maximum number of loads for each addressing unit in the circuit. Any RAMs in excess of this parameter will be driven by a separate addressing unit.

Another TRANS parameter (ADRUNIT) is also available which allows the user to control the addressing units used in a design. The user can insert any number and addressing units of any length into the circuit. This feature is useful if the simple

addressing unit allocation algorithm used by TRANS results in a larger than optimal design.

4.8 Delay Calculation

The critical path of a synchronous circuit dictates the maximum clock rate at which the circuit will correctly function. The critical path is the maximum DFF to DFF logic delay time. This logic delay is made up of logic element propagation delays. routing delays and set-up and hold times on the DFFs. The logic element delay for each component is dependent on the output loading for that element. A higher number of loads means that a higher total current is required from the driving point. also each load and wiring element adds capacitance to the output of any element. The combination of this load current requirement and the load capacitance dictates the time required to drive a node to the required state. It is useful to be able to estimate the critical path of a circuit at an early stage in order to know the maximum clock speed for the circuit and where improvements may be made to decrease the critical path length.

To estimate the delay on any path it is necessary to understand the behaviour of each cell in the technology library in terms of the loading to delay time relationship. Also it is important to account for any routing delays which may be present in the path. This routing delay time is not known until after the circuit has been fully placed and routed. This post layout timing is more accurate than the pre-layout estimate but is still only an estimation. The exact critical path will vary from device to device and over temperature and voltage ranges. To calculate the delay time for

any path it is necessary to include an estimate of this routing delay at some operating point.

The loading factor - delay performance information for a technology is imported into TRANS using a delay file. In this file each library element is described in terms of the input loading factor for each input pin and the delay time relative to load for each output pin. The delay file description for an ACTEL three input NAND gate is shown below.

```
NAND3 ( (1 1 1) ( {[1 102] [2 106] [3 116] [4 125] [8 146]} ) )
```

All the information pertaining to the NAND3 cell is contained within a set of brackets. The first internal set of brackets indicates the loading factor for each input pin on the cell. for the NAND3 element each of the three input pins exhibits a loading factor of one. The second set of brackets contains the load performance information for each output pin from a cell. The curly brackets are used to separate the data for one output pin from another. since a nand gate has only one output there is only one set of curly brackets. Each set of square brackets indicates two numbers representing the loading factor and the corresponding delay time for that output pin on the cell. The loading factors must be in sequence from smallest to largest and both the loading factor and the delay time must be integer values. In this example the units for the delay times are tenths of nanoseconds. The delay time for any loading factor not present in the table is obtained by linear extrapolation.

The delay data for ACTEL devices is available within the ACTEL data sheets [ACT89]. The delay information includes a statistical measure of the expected routing delay based on the loading factor but the exact routing delay will be implemen-

tation and device dependent. The delay figure determined using this information is useful to get an idea of where the critical path is and to compare the critical paths of different implementations.

4.9 Summary

The TRANS hardware compiler reads in a DFIRST netlist and generates all primitives using generic logic elements as described in Chapter 3. This is then flattened by TRANS to remove all hierarchical levels within the design. A series of optimizations can be applied to this flattened netlist resulting in a more efficient technology specific implementation.

XILINX devices are N times programmable Field Programmable Gate Arrays containing Configurable Logic Blocks. I/O Blocks and programmable routing resources. Each 4000 series CLB is a nine input. four output block containing two four input LUTs, one three input LUT and two D type flip-flops. Each four input LUT can be configured as a serial RAM containing 16 one bit memory locations.

ACTEL FPGAs are one time programmable devices containing C-modules. S-modules. programmable I/O pins and anit-fuse based programmable connection resources. The C-modules are single output multiplexor based cells which can implement one function of 8 variables, and many different functions of 7 or fewer inputs. The S-modules are single output cells which can perform one logic function of 7 inputs and many different functions of 6 or fewer inputs. This module also contains a level or edge triggered latching element.

TRANS optimizations include redundant hardware removal. RAM mapping and

logic mapping. The redundant hardware removal optimization removes redundant logic from a circuit. The RAM mapping optimization converts contiguous shift register chains (common in digit-serial designs) to a smaller implementation using a serial RAM, a counter and a single DFF. This optimization can be used with the XILINX 4000 series technology which can implement serial RAM. The logic mapping optimization converts logic implemented using generic logic elements to technology specific logic elements. This operation is performed using a rule database containing a set of rules describing a source logical arrangement to be found and a replacement logical arrangement which leads to a smaller implementation in the target technology.

Chapter 5

Applications

The DFIRST language has been used to design a variety of different circuits including digital filters [GT92, NT91], a digital oscillator [Wor92a], a spread spectrum receiver/transmitter pair [Pat93] and a free field listening on headphones system [Bei94]. These tools have also been used by students in graduate level courses [TG95] to design and implement class projects.

In this chapter the implementation of several DFIRST designs will be discussed. Digital filter implementations for some common filter structures are presented in section 5.1. To generate the DFIRST code for each digital filter the synthesis programs BITSYN [NT91], SNAFU [Joh92], FIRGEN [TGG95] and DIGIPARSE will be used to convert a high level digicap [Tur88] filter description to a DFIRST description. These circuits will be used to investigate the reduction and mapping operations of TRANS for the ACTEL FPGAs and XILINX 4000 series FPGAs.

5.1 Digital Filters

Digital filters are critical components in many products ranging from CD players to Cellular telephones. These filters can be broken down into two main categories. Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters. Several IIR filters will be designed including a fifth order bilinear LDI filter [Bru75], a 7th order wave digital filter [Fet86] and a custom designed digital filter designed using

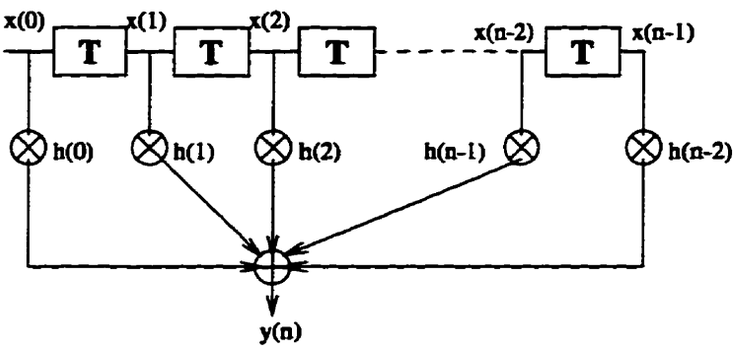


Figure 5.1: FIR Filter Structure

the filter optimization program noisegen [Kac95]. A 61 tap FIR filter will also be designed. Each of these filters will be implemented using different synthesis programs and then a series of applicable reductions will be applied in order to investigate the performance of the TRANS reduction utilities on each design.

5.1.1 FIR Filters

Finite Impulse Response filters are used when absolute stability is required or when linear phase is necessary. The main drawback of FIR filters is the high filter orders required to implement sharp transitions, low passband ripple or high stop band attenuation digital filters [Jac89]. The classical structure of an FIR filter is shown in figure 5.1. For an N tap FIR filter N-1 registers, and N multipliers are needed to implement the filter. Each of the multiplier outputs are summed together using N-1 adders.

The FIR filter shown in figure 5.1 is an arbitrary phase implementation, to guarantee linear phase it is necessary that the coefficients of the filter be symmetrical or anti-symmetrical around the center tap [Jac89]. Taking advantage of this symmetry

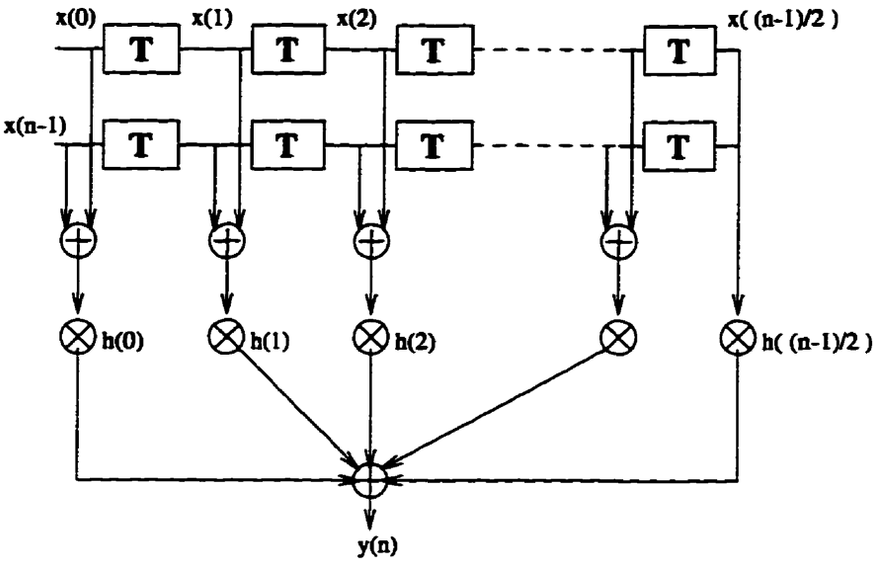


Figure 5.2: Linear Phase FIR filter structure (odd order)

results in the structure shown in figure 5.2. For a symmetrical FIR filter only N/2 multiplications are necessary. the number of registers and addition operations are the same as in the general implementation.

The coefficients. given in appendix A. for this filter are constant values implementing a fixed transfer function. The coefficients for this filter were generated using NOMAD [Svi91] and in order to reduce implementation complexity the CSD coefficient option within NOMAD was used. The coefficient wordlength was selected to be 12 bits and the number of bits for each coefficient was limited to 3 bits. the data wordlength was chosen to be 16 bits.

5.1.2 Bilinear LDI Digital Filter

Lossless Discrete Integrator (LDI) [Bru75] filters are IIR filters which are developed by transforming a voltage-current signal flow graph of an analog proto-type ladder

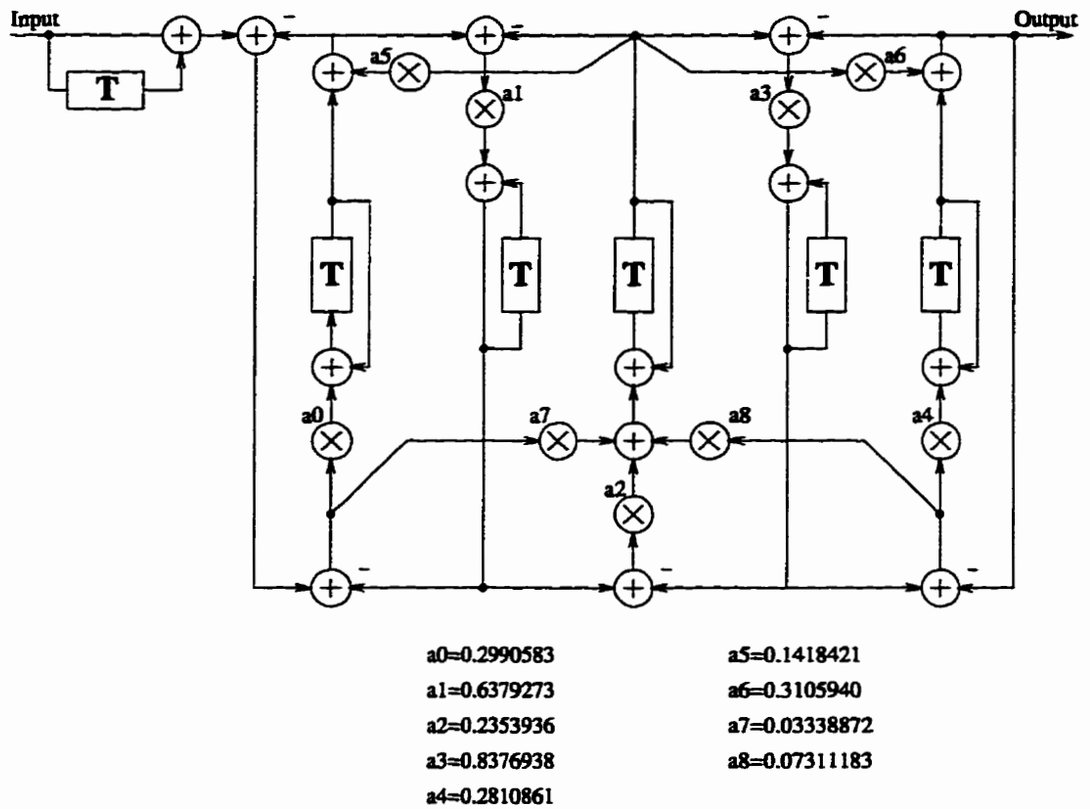


Figure 5.3: 5th Order Bilinear LDI Filter Structure

filter to the digital domain [Bru75]. The test filter is the Bilinear LDI filter shown in figure 5.3. This signal flow graph requires 9 multiplications. 14 additions and 5 registers to store state variables.

For this digital filter a coefficient wordlenth of only 6 bits will be used and the data wordlength will be 17 bits.

5.1.3 Wave Digital Filter

Wave filters are another implementation style for IIR filters. Wave filters are derived by transforming an analog proto-type filter to the digital domain [Fet86]. The sample filter implemented here is the 7th order wave filter shown in figure 5.4. This filtering

operation requires 7 multiplications, 21 additions and 7 registers.

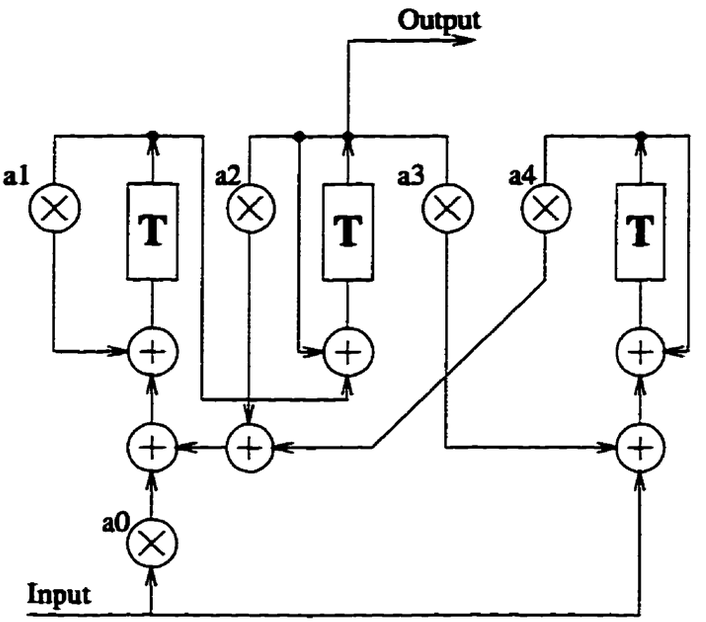
For this filter the coefficient precision was chosen to be 6 bits and the data wordlength is 18 bits.

5.1.4 N2 Filter

The third IIR digital filter used to test the mappings and optimizations of TRANS was derived using NOISEGEN [Kac95], a simulated annealing based signal flow graph optimization tool. NOISEGEN takes as input a DIGICAP netlist describing a fixed coefficient digital filter. This starting netlist is optimized using simulated annealing [Kac95] to obtain a filter which has the same transfer function but is less sensitive to finite precision effects within the filter. The flow graph for the 'N2' filter is shown in figure 5.5.

5.2 High Level Synthesis Tools

To generate a suite of register transfer level test designs from the filter specifications in the previous sections, a variety of high level synthesis tools will be employed. High level synthesis for digital filters is the process of converting a completely specified signal flow graph containing N multipliers and P adders to a register transfer level implementation containing fixed resources which may be shared between the various operations required to implement the digital filter. In this section the synthesis tools BITSYN,SNAFU,FIRGEN and DIGIPARSE will be discussed. The strategy used by each CAD tool to synthesize the design will be presented to better understand the type of circuits which are generated by each synthesis tool.



a0=0.034788
a1=0.21715
a2=-0.40745
a3=-2.00069
a4=0.034788

Figure 5.5: N2 Filter Structure Generated by Noisegen

lated annealing [Kac95] to search for a solution which exhibits the most desirable area and throughput characteristics.

SNAFU only uses the LSB/MSB indicating output from the DFIRST control generator. This signal is delayed as required to generate all timing signals required in the RTL implementation. Multiplexing signals are generated from these delayed pulses and are used to select the correct data and coefficient input to each shared operator. The data wordlength is always the user specified value and is not restricted by the minimum system wordlength of the circuit. Using this control scheme the number of clock cycles per input sample is always the minimum required. One drawback to the SNAFU control scheme is the long control path delays which are present if the cycle time for one complete computation is long. SNAFU, like BITSYN, uses the parallel coefficient serial multiplier together with logic circuit ROMs to implement multiplication by fixed coefficient values.

5.2.3 FIRGEN-DIGIPARSE

FIRGEN or FIR filter GENerator [TGG95] is a program dedicated to the bit-serial implementation of FIR filters. The input to this program is simply the set of coefficients required to implement the desired transfer function and word length parameters to select the data and coefficient wordlengths for the filter. No attempt is made within FIRGEN to share any hardware resources, instead each multiplier is implemented using a fixed CSD recoded SHIFTMULT multiplier.

The process of converting a set of coefficients to a RTL implementation is carried out in several simple steps. The first step is to generate the delay chain of the FIR filter using N-1 data wordlength long registers. Then all the taps which exhibit the

same coefficient are added together, this reduces the number of multipliers in the circuit by at least two for a symmetrical FIR filter. Finally the output of these adders are multiplied by the appropriate coefficient and summed together using a tree of bit-serial adders. The resulting circuit can accept a new input sample every SWL clock cycles where SWL is the data wordlength selected by the user.

DIGIPARSE is the IIR equivalent of FIRGEN. The input to this program is a DIGICAP specified netlist for a fixed coefficient digital filter. DIGIPARSE then implements every operator in the graph as is, using CSD recoded SHIFTMULTs to implement each multiplication within the filter. Like FIRGEN, the user can specify the desired coefficient and data path precisions but the number of clock cycles required between samples is determined by DIGIPARSE. This time will be the longest path within the filter between an input or state variable and the output or another state variable.

5.3 Mapping Performance

To test the hardware mapping operation (section 4.6) of TRANS, the fifth order bilinear LDI filter, seventh order wave filter and the N2 filter were synthesized using BITSYN, SNAFU and DIGIPARSE. The impulse response of all three digital filters was generated using the DSIM simulator and checked against the ideal impulse responses generated by DIGICAP. The DSIM impulse response simulations for all nine test filters matched the DIGICAP impulse response simulations. This verifies that each filter example performs as expected.

	BITSYN			SNAFU			DIGIPARSE		
Filter	B	A	per	B	A	per	B	A	per
LDI5	1222	662	45.8	1434	782	45.5	1055	586	44.5
Wave7	1265	690	45.5	1439	739	46.6	1081	573	47.0
N2	846	419	50.5	1016	520	48.8	493	248	49.7

Table 5.1: ACTEL Cell Counts for Test Filters
Before(B) and After(A) Hardware Mapping

5.3.1 ACTEL Mapping

The nine sample designs (three filters generated using three different synthesis programs) were mapped to the ACTEL technology using TRANS. The gate count (measured in ACTEL cells) before and after hardware mapping and the percentage reduction in cell counts are shown in table 5.1.

The original (before) cell counts from table 5.1 reflect the circuit size without any mapping or reduction operations. In this case a two to one multiplexor is implemented using 2 AND gates. one inverter and one OR gate (4 cells) as specified in the generic logic library. After hardware mapping a two to one multiplexor would be implemented using only a single ACTEL cell. The percent reductions in circuit size due to the mapping operation ranged ranged from 44 to 50 percent for ACTEL designs.

After mapping to the ACTEL technology each design was simulated at the gate level using LOGSIM [BW92] to verify that the test filters still function correctly. For all the test filters it was found that the LOGSIM gate level simulations matched the DSIM RTL simulations.

Another aspect of the hardware mapping operation which can be examined is it's effect on the critical path of each digital filter. The critical path measured by

	BITSYN			SNAFU			DIGIPARSE		
Filter	B	A	per	B	A	per	B	A	per
LDI5	58.2	35.0	39.9	84.5	42.3	49.9	50.1	25.2	49.7
Wave7	50.1	26.2	47.7	64.5	35.0	45.7	50.1	25.2	49.7
N2	50.1	33.0	34.1	65.0	35.5	45.4	50.1	24.7	50.7

Table 5.2: ACTEL Critical Path Lengths (ns)
Before(B) and After(A) Hardware Mapping

TRANS before and after hardware mapping and the percentage reduction in the critical path length is shown in table 5.2.

In the original circuit a two to one multiplexor has three levels of logic and after the mapping operation this same function has only one level. The hardware mapping operation reduced the critical path length of of the ACTEL designs by 40 to 50 percent.

5.3.2 RAM Mapper Performance

To test the performance of the RAM mapping operation each of the nine filter implementations was translated to the XILINX 4000 series technology, first without the RAM mapper and then with the RAM mapper enabled. During the RAM mapper translation a LENGTH parameter (section 4.6.2) of 6 will be used. Here any contiguous shift register of length greater than 6 is converted to the RAM implementation. In each case a more 'optimal' addressing scheme is selected using the ADRUNIT parameter in favor of the addressing unit(s) selected by TRANS. For both the RAM mapped and RAM unmapped translations the redundant hardware removal operation was enabled with a FANOUT (section 4.5) setting of 10.

A major factor in the effectiveness of the RAM mapping operation is the selec-

	BITSYN	SNAFU	DIGIPARSE
LDI5	8, 9, 10, 11, 12x4, 13, 14x3, 16, 17, 18x3	6x4, 7x4, 9x3, 10x8, 11, 12x4, 14, 16x2, 17x2, 18	8x3, 9x3, 10, 12x5, 13, 17x3
Wave7	8, 11x4, 12x3, 13, 14, 15, 16x3, 19x2	8, 10, 11x13, 12x5	6x4, 8x4, 9
N2	6, 15, 16, 17x4, 18, 19	11, 13, 14, 15, 16x7, 17	15, 16, 22

Table 5.3: Test Filter Contiguous Shift Register Lengths

	BITSYN	SNAFU	DIGIPARSE
LDI5	8,14 = 35	6,12 = 57	8 = 38
Wave7	8,14 = 35	8 = 42	6 = 11.5
N2	6,15 = 16.5	11 = 33.5	15 = 8

Table 5.4: TRANS Address Allocations and CLB Cost Estimate

tion of addressing units to control the various serial memories in the circuit. The contiguous shift register lengths required by each test filter are shown in table 5.3. Along with the memory lengths in the circuit, the expected cost for the memory implementation in CLBs is also shown. The addressing scheme used by TRANS is not optimal in terms of the number of CLBs required to RAM map the register lengths of each example. The addressing scheme chosen by TRANS for each filter and the resulting CLB count estimate is shown in table 5.4.

With the ADRUNIT parameter the user can select which addressing units are present within the design. Using this feature an improved set of addressing units can be generated as given in table 5.5. Comparing table 5.4 and 5.5 shows that a user generated addressing scheme can reduce the implementation size of the RAM mapping by up to 50 percent. Each of the nine filter implementations behaves similarly in this regard.

Table 5.6 compares the RAM mapped (user selected addressing units) circuits

	BITSYN	SNAFU	DIGIPARSE
LDI5	8,12,14,18 = 27	6,7,9,10,12,16 = 31.5	8,9,12,17 = 21
Wave7	8,11,13,16 = 23	8,11 = 17.5	6,8 = 8
N2	6,15,17 = 17.5	11,16 = 16	15 = 8

Table 5.5: User Address Allocations and CLB Cost Estimate

	BITSYN		SNAFU		DIGIPARSE	
Filter	FG	DFF	FG	DFF	FG	DFF
LDI5	22.9	38.9	32.6	44.8	28.6	39.6
Wave7	18.4	34.9	14.8	41.4	9.8	15.5
N2	21.1	39.0	12.8	41.3	11.1	24.4

Table 5.6: XILINX 4000 Series RAM Mapping
Percentage Increase in FGs and Percentage Decrease in DFFs

to the unmapped circuits. Table 5.6 shows the percentage reduction in the number of Function Generators (FGs) and the percentage reduction in the number of DFFs used by the nine sample circuits. This table shows that the number of FGs increases as the number FFs decreases. As expected the RAM mapper trades off DFF usage for FG usage.

A useful measure which can be use to investigate this trade-off further is the number of DFF elements which can be implemented per function generator. This FF/FG efficiency factor is defined as the change in DFF elements divided by the change in FG elements affected by the RAM mapping optimization.

The resulting trade-off figure for each of the nine filter implementations is give in table 5.7. The trade-off figures range from 3.8 FFs/FG to 7.6 FFs/FG. The magnitude of this figure depends on the number and length of the different addressing units used to implement the RAM mapping.

Filter	BITSYN	SNAFU	DIGIPARSE
	FF/FG	FF/FG	FF/FG
LDI5	4.5	4.3	3.8
Wave7	4.9	6.0	3.25
N2	4.6	7.6	5.0

Table 5.7: DFFs/FG for XILINX 4000 Series RAM Mapping

	BITSYN		SNAFU		DIGIPARSE	
Filter	ACTEL	4000	ACTEL	4000	ACTEL	4000
LDI5	48.0	41.2	47.7	45.2	53.0	51.7
Wave7	49.9	40.1	55.0	44.3	55.7	33.6
N2	53.1	42.7	51.6	44.3	57.9	39.2

Table 5.8: Overall Area Reductions by TRANS Optimizations

5.3.3 Overall Performance

This section discusses the overall area reductions implemented by TRANS on the nine sample designs for the ACTEL and XILINX FPGA devices. For each test filter the "best" implementation will be compared to the unoptimized circuit for each target technology. For the ACTEL designs both the circuit size and circuit critical path will be used to select the best design. Only the size measured in CLBs will be used to compare XILINX implementations. Table 5.8 shows the total reductions in circuit size and table 5.9 shows the reductions in critical path length.

The ACTEL area reductions ranged from 48 percent to 57.9 percent and the corresponding critical path reductions ranged from 36.0 percent to 57.1 percent.

	BITSYN	SNAFU	DIGIPARSE
LDI5	49.0	55.5	55.4
Wave7	51.8	52.9	56.5
N2	36.0	49.2	57.1

Table 5.9: Overall Critical Path Reductions by TRANS Optimizations

Filter	BITSYN	SNAFU	DIGIPARSE
LDI5	136	85	30
Wave7	64	38	31
N2	72	48	16

Table 5.10: Number of Clock Cycles per Input Sample

Reductions in the X4000 technology can be attributed to the redundant hardware removal operation and the RAM mapping operation. Overall reductions in circuit size of from 33 to 52 percent were observed on the nine test filters. with the majority of filters exhibiting area reductions of 40 to 45 percent.

5.3.4 Filter Implementation Comparisons

In the previous sections three different filters (ldi5,wave7.N2) were implemented using three different synthesis programs (BITSYN. SNAFU. DIGIPARSE). The three implementations of each digital filter can be compared to determine which synthesis tool is the most useful for each circuit. The figure of merit used to compare each filter will be the area time (AT) product [HC90] which equally weights circuit area and processing time.

The comparisons will be performed using the ACTEL gate counts and critical path lengths. The area measure is the number of ACTEL cells required by each design. The time can be obtained by multiplying the critical path length by the number of clock cycles required to process a single sample. For each filter the solution yielding the smallest AT product will be used for this comparison. The number of clock cycles required between samples for each of the nine filters is given in table 5.10.

The total AT products for each of the nine filters is given in table 5.11. For each

	BITSYN	SNAFU	DIGIPARSE
LDI5	7.09	6.19	1.0
Wave7	3.23	2.55	1.0
N2	13.3	11.5	1.0

Table 5.11: ACTEL Area-Time Products for Test Filters

row of this table (filter type) the DIGIPARSE AT products are normalized to a value of one. The remaining elements on the row are normalized to this value. For each filter the DIGIPARSE filter results in the smallest AT product. the SNAFU circuits were next best and BITSYN yielded the biggest/slowest solutions.

The differential in circuit quality was smallest in the wave filter implementation. The reason for this lies in the topology of wave digital filters. which contain several cascaded multiplication operations. This reduces the cost of sharing a general purpose multiplier between several multiplication operations. Since the result from one multiplication must be computed before the next can begin. very little of the processing can be done in parallel. As such the speed (measured in clock cycles) of the resource shared solutions and the non-shared solutions are not that different.

In the LDI and N2 filters more of the structure can be computed in parallel. which results in a higher cost for sharing resources. In a shared multiplier environment an operation which could proceed must wait for a multiplication unit to be free. resulting in extra wait time and additional registers to store intermediate results.

For all of the circuits the DIGIPARSE solutions yielded the smallest number of clock cycles for the computation. the smallest critical path length and the smallest area. These results indicate that filters requiring fixed coefficient values should be implemented using one dedicated CSD multiplier per multiplication in the filter.

5.4 61 Tap FIR Filter

Given the NOMAD generated finite precision coefficients. the bit-serial DFIRST RTL filter description can be synthesized using FIRGEN. This filter contains 31 CSD multipliers. 60 16 bit shift registers for state storage, and 60 adders to sum the multiplier outputs. The input/output signals from this filter are 12 bit parallel values. The format converting parallel to serial and serial to parallel DFIRST primitives are used to interface to external parallel A/D and D/A converters.

The DFIRST filter implementation generated by FIRGEN was translated to the XILINX 4000 series architecture with no optimizations enabled and the circuit occupied 431 percent of the 384 CLB DFFs in the 4005PG156 target device. and 83 percent of the 384 Function Generators in the FPGA. This original circuit would require more than four 4005 devices for a full implementation.

The first reduction which can be applied is to remove redundant hardware elements (section 4.5). The controlling FANOUT parameter is set to 10 for this reduction.

The resulting circuit from this operation requires 347 percent of available DFFs and 80 percent of FGs. a significant improvement over the original circuit containing redundant hardware. The next operation will be to apply the RAM mapping in order to convert all the state storage elements to RAM implementations and reduce the number of DFFs in the design. The minimum contiguous shift register length is set to 6. The resulting shift register lengths and user selected addressing units are given in figure 5.12.

With this RAM mapping optimization the resulting area requirements are 137

Length	Number	Addressing Units
7	7	1
8	4	0
9	2	0
13	3	1
14	2	0
15	14	1
16	13	1
17	9	1
18	12	0
19	2	0
20	2	0
32	3	1
33	1	0

Table 5.12: ACTEL Area-Time Products for Test Filters

percent DFFs and 108 percent FGs. In order to reduce the number of FGs hardware mapping operation presented in section 4.6.2 will be applied. Here all BLATCH type structures will be replaced with the clock enabled flip-flop available within the XILINX 1000 architecture. When this mapping is used 58 separate replacements are performed by TRANS and the resulting circuit size is 137 percent DFFs and 96 percent FGs. Some reduction in the DFF count is still required before the placement and routing can be attempted.

In order to reduce the number of DFFs used within the CLBs of the 4005 another mapping presented in section 4.6.2 is used to map DFFs to the IO ring. For the filter implementation only 12 inputs and 12 outputs are needed for the filter. This leaves a large number (86) of I/O pads which may be used as two bit shift registers. The I/O mapper optimization mapped 84 two bit shift registers to the I/O ring. The resulting circuit size is 106 percent DFFS, 96 percent FGs and 98 percent IOs.

The 84 IO cells used in the I/O ring mapping removed 168 DFFs from the internal

circuitry, this is a reduction in internal DFF usage by 168/384=43.75 percent. The actual reduction in DFF count was only $137 - 106 = 31$ percent. The reason for this discrepancy lies in the reductions which are performed by the XILINX mapping tool PPR before placement and routing. One reduction done here involves removing components which do not drive any internal loads (loadless signals). After this mapping operation some DFFs which previously drove no internal loads are now connected to external pins. These elements can not be removed by PPR. To correct for this the loadless components can be removed using TRANS.

When this optimization is enabled the IO mapping and the RAM mapping operations change. Some loadless DFFs which were previously mapped to RAM implementations have now been removed. The resulting circuit size is now 101 percent DFFs. 96 percent FGs and 86 percent IOs. Only 54 IO blocks are used by the IO mapper function.

This circuit is still slightly larger than what the 4005 can accommodate. To shrink the design further the FANOUT parameter is changed to 12 from 10. The resulting circuit area is now 97 percent DFFs, 96 percent FGs and 84 percent IOs. This circuit can now be placed and routed.

The PPR program terminates indicating that not all DFFs can be placed within the circuit. This problem occurs when using the clock enable pin on the 4000 series CLB DFF. Both DFFs in the CLB must be driven by the same clock enable signal. If this is not possible, one of the two DFFs cannot be used. These lost DFF elements caused this design to be too large for the target device. In order to shrink the design further the FANOUT parameter is increased to 20 The circuit size is 91 percent DFFs. 94 percent FGs and 84 percent IOs. This design now fully places and routes

within a single 4005PG156.

The overall reductions in internal DFF count for this FIR filter is 78.9 percent. At the same time the function generator count was increased by 13.3 percent. The final circuit was implemented and tested on a single 4005PG156. The circuit functioned correctly at 20 MHz. even with the FANOUT of 20 on some internal signals. Since the number of bits per input word is 16 the number of samples per cycle processed by this unit is $20 * 10^6/16 = 1.25$ M samples/second.

5.5 Digit-Serial Circuits

The DFIRST language can describe circuits of variable digit-widths. In the following section TRANS circuit transformations are tested on higher digit-width circuits. The arbitrary digit-width multiplier circuit from section 3.15 will be used to investigate the speed/area tradeoffs of using higher digit width components. As well, one digit-serial digital filter will be designed using SNAFU and mapped by TRANS to the ACTEL and XILINX technologies to investigate the performance of TRANS optimizations on higher digit width circuits.

5.5.1 Digit-Serial Multipliers

To examine the speed/area trade-offs of higher digit width components the DFIRST digit-serial multiplier units will be used. The CWL for the multiplier is eight and the data wordlength is 18 bits. Different digit-width multipliers ranging from W=1 to W=6 will be used for the test. For the 4 and 5 bit digit widths a data wordlength of 20 bits will be used since there must be an integer number of digits per data word.

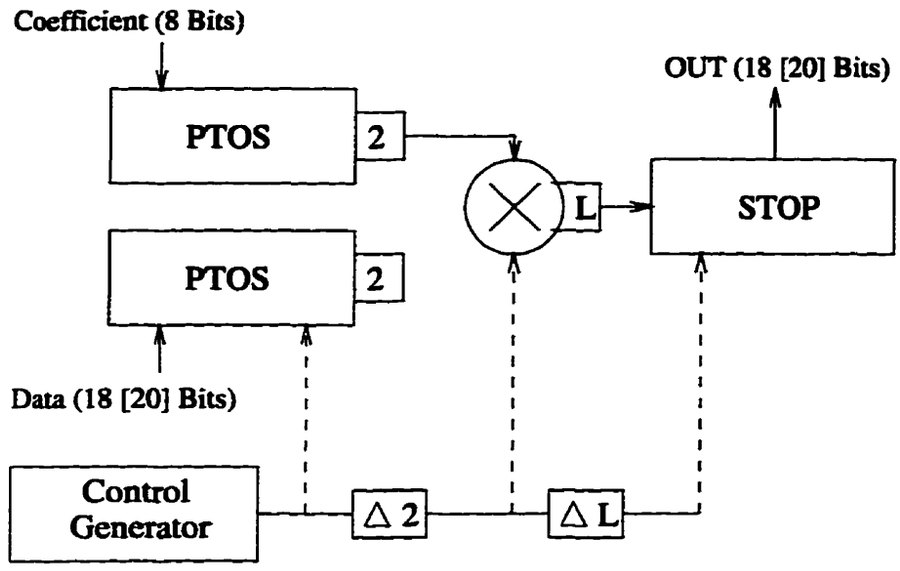


Figure 5.6: Multiplier Test Circuit

The results obtained for these two circuits will be weighted by the appropriate factor to remove the effect of the altered data wordlength.

The DFIRST circuit will consist of two parallel to serial converters. one for the 8 bit coefficient and one for the data signal. a digit serial multiplier. a serial to parallel converter and a control generator as shown in figure 5.6.

Converting the DFIRST code for the circuit given in figure 5.6 between the different digit width circuits is relatively simple. The first step is to change the dfault digit width setting to the desired value using the DIGIT (section 2.3.1) compiler directive. This parameter defines the default digit-width for all serial signals in the DFIRST file. The next step is to change the first CYCLE (section 3.17) count to the appropriate setting. The CYCLE count indicates the number of clock cycles per data word and is set to SWL/W. For the six different digit-width circuits in question the CYCLE parameter is set to 18, 9, 6. 5, 4 and 3 for digit-widths of 1 to 6 bits

DW	XILINX			ACTEL		
	Area	Critical	Total	Area	Critical	Total
	(CLBs)	Path(ns)	Time(ns)	(Cells)	Path(ns)	Time(ns)
1	17	17.5	315.8	91	27.2	489.6
2	25	22.2	200	133	34.0	306.0
3	36	31.3	188	184	40.8	244.8
4	43	38.5	192.3	224	47.6	238
5	50	47.6	190.5	264	54.4	217.6
6	58	58.8	176.5	304	61.2	183.6

Table 5.13: Area and Computation Time
for Different Digit Width DFIRST Multipliers

respectively.

Each of the six multiplier circuits was converted to the XILINX 4000 and ACTEL technologies. The XILINX designs were implemented on a single 4005PG156 device and the critical paths for all the circuits were measured by increasing the clock frequency until the circuits stopped functioning correctly. The delay calculation feature of TRANS was used to calculate the critical path lengths for the ACTEL designs. The critical path length multiplied by the number of clock cycles required to generate the output (CYCLE count) indicates the total time needed for the multiplication. The cell counts, critical path lengths and total multiplication times for each design are given in table 5.13. The times for the 4 and 5 bit digit-width multipliers have been multiplied by 18/20 to normalize these circuits for comparison purposes.

Using the data from table 5.13 the area-time product for each implementation can be computed. The area-time product vs. digit width for DFIRST multipliers implemented in XILINX 4000 and ACTEL FPGAs is shown in figure 5.7. All AT products are normalized to the bit-serial AT value for each technology.

The AT product curve shows that the low digit-width multipliers have a lower

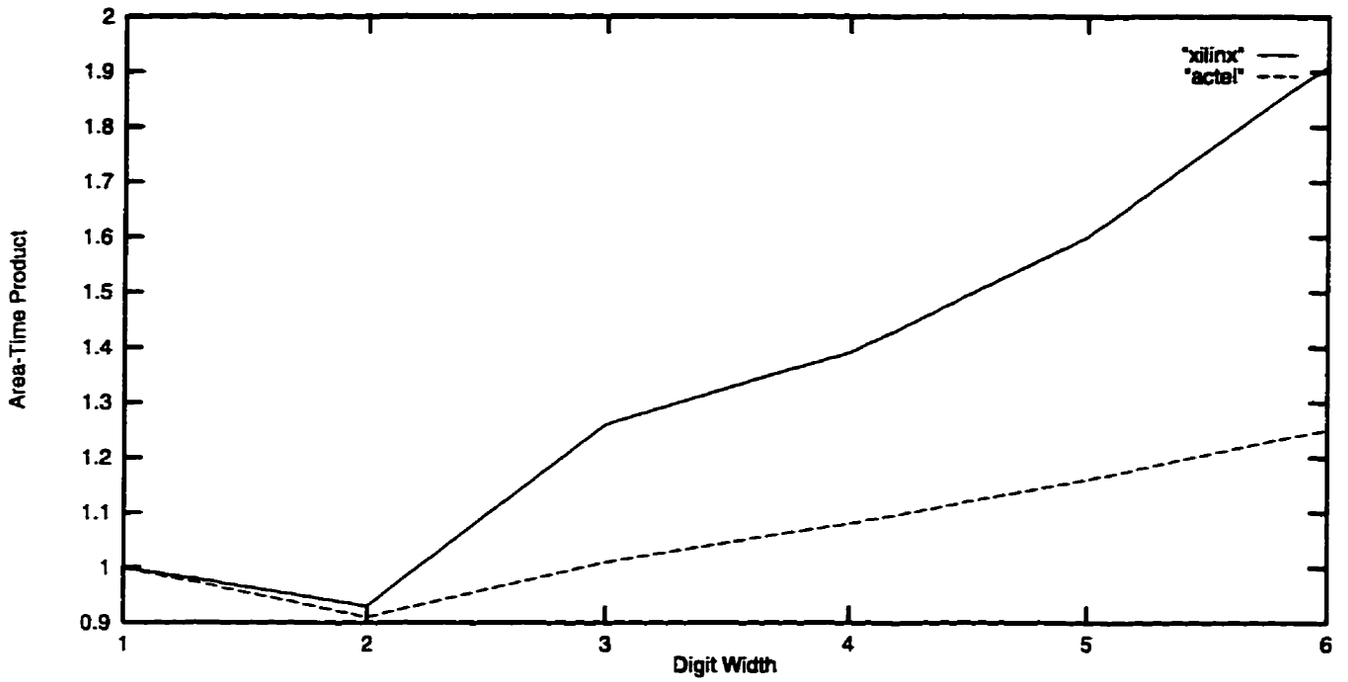


Figure 5.7: Digit-Serial Multiplier Area-Time Products

area-time product than the more parallel designs. A digit-width of 2 results in the most AT efficient multiplier. For the other digit widths the reduction in the number of clock cycles is accompanied by a larger increase in size and critical path length resulting in larger AT products.

5.5.2 Digit-Serial Filters

To examine the performance of the TRANS circuit transformations on digit-serial circuits the SNAFU program was used to synthesize the ldi5 digital filters. described in section 5.1.2. using different digit width architectures.

The filter was synthesized using digit widths ranging from W=1 to W=6. The original data wordlength for this filter was 18 bits but for the W=4 and W=5 filters the internal wordlength was changed to 20 in order to maintain an integer number

DW	Before	After	Percent Reduction
1	1434	760	47.0
2	1784	887	50.3
3	2136	1066	50.1
4	2523	1221	51.6
5	2865	1356	52.7
6	3182	1466	53.9

Table 5.14: ACTEL Area Before and After TRANS Optimizations for Different Digit Width LDI Filters

of digits in each word. The impulse response for each of the different digit-width designs was simulated using DSIM and were found to function correctly. Each of these designs was then translated to the logsim format for a gate level simulation. During this process the redundant hardware remover (FANOUT=10) and the hardware mapper to the ACTEL library were enabled. Each of the filters were found to function correctly and gave identical impulse responses to the impulse responses generated by DSIM.

5.5.3 Optimization Performance

Each of the filters were mapped to the ACTEL technology using the hardware mapper and the Redundant Hardware Remover with a FANOUT limit of 10. The percentage reduction in circuit size is given in table 5.14 and the percentage reductions in critical paths is given in table 5.15.

The performance of the hardware mapper increases both in terms of circuit area and circuit critical path as the digit width increases. The hardware mapper has the greatest impact on the logic elements within the design and a lesser impact on the DFFs in a design. Since the ratio of logic to pipelining elements increases with digit

DW	Before(ns)	After(ns)	Percent Reduction
1	67.9	30.2	55.5
2	69.4	31.2	55.0
3	77.2	34.0	55.6
4	91.3	40.8	55.3
5	105.9	47.6	55.1
6	126.2	57.9	54.1

Table 5.15: ACTEL Critical Paths Before and After TRANS Optimizations for Different Digit Width LDI Filters

width the effect of the hardware mapping operation also increases with increasing digit width.

To test the effectiveness of the RAM mapping operation on higher digit-width circuits the six filters were converted to the XILINX 4000 series technology. One RAM mapping control parameter which becomes increasingingly important as the digit width increases is the LENGTH (section 4.7) parameter. This parameter dictates the minimum length of contiguous shift registers which will be converted to RAM implementations. This parameter is normally set to 6. which results in all shift register chains of 7 DFFs or more being converted to RAM. This works well for bit-serial circuits which make use of long shift register chains but not as well for higher digit width circuits.

Consider the implementation of an 18 bit word delay implemented in various different digit widths as shown in figure 5.8. The bit-serial word delay is a single 18 bit shift register. the W=2 implementation is two 9 bit shift registers and the W=6 implementation is six three bit shift registers. So the word delay implementation results in W N/W long shift registers. The longest delay typically used in SNAFU circuits is one word delay [Joh92]. This means that the LENGTH parameter must be

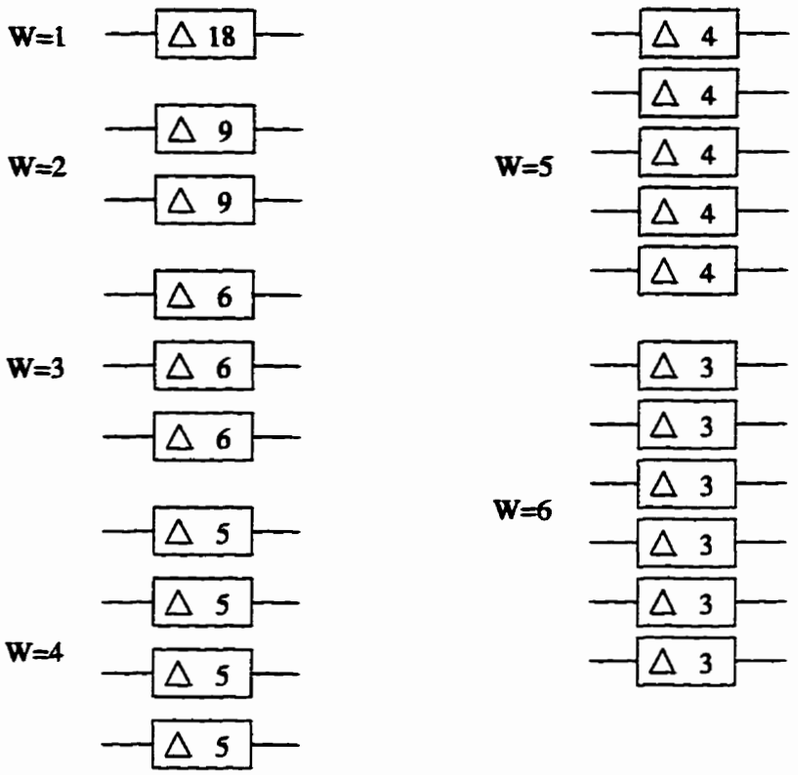


Figure 5.8: Word Delay Implementation for Digit Widths 1 to 6

decreased as the digit width increases. The percentage improvement for each circuit over the unoptimized circuit in terms of CLBs is given in table 5.16. Also presented is the number of DFFs/FG which were implemented in each case.

As the LENGTH parameter decreases the DFF utilization decreases and the FG utilization increases, this is expected since more DFFs are being mapped to RAM. However the efficiency of the mapping operation decreases as LENGTH decreases. A XILINX serial RAM element has four address lines, one data input and one data output, that is up to 16 bits deep and only one bit wide. If only a three deep RAM is required a full 16 bit RAM must be included in the final implementation, with only three elements of the RAM actually being used. The fact that RAM blocks come in

DW	Area Reduction	DFFs/FG
1	0	0
2	43.6	4.71
3	30.5	3.2
4	26.6	3.0
5	20.5	2.9
6	8.1	1.9

Table 5.16: Area Reductions and RAM Efficiency for Different Digit Width LDI Filters

16 bit chunks results in the loss of efficiency for short shift register chains.

As the digit width increases the percentage reductions in DFF utilization decreases. This occurs because as the digit width increases fewer DFFs are found to be suitable for RAM reduction. As. well the efficiency of the RAM reduction in terms of the number DFFs/FG also decreases as the digit width increases. This occurs because the more of the RAMs being used are of the shorter lengths as the digit width increases. until at W=6 almost all of the RAMs are three bits long. As a result the net effect of the RAM reduction optimization decreases in effectiveness as W increases.

Filter Performance

In this section the different digit-width LDI filters will be compared in terms of area and time in order to determine the relationship between digit width and overall filter performance.

The area of ACTEL digit-serial LDI filters is given in table 5.17. Here all values are normalized to the bit-serial settings. As the digit width increases the size of the circuit implementation also increases. A W=2 filter is less than double the size of a bit-serial filter. This is expected since some DFIRST operators such as registers

DW	Size
1	760
2	887
3	1037
4	1221
5	1197
6	1254

Table 5.17: ACTEL Area for Different Digit Width LDI Filters

DW	Clocks	ACTEL	ACTEL
		Critical Path(ns)	Total Time(ns)
1	85	30.2	2567
2	45	31.2	1404
3	38	36.5	1387
4	36	40.8	1469
5	34	52.0	1768
6	32	54.9	1757

Table 5.18: Efficiency data for Test LDI Filter

and format converters do not grow linearly in size with digit width but may remain constant in size regardless of digit width.

The critical path length of these digital filters is shown in figure 5.18. The path length does increase linearly with digit width but only after W=3. before this point the delay curve is relatively flat. The reason for this lies in the type of filters generated by SNAFU. This synthesis tool makes use of zero delay multiplexors in front of all shared components, if the depth of these multiplexors is too large then the these elements form the critical path. In the case of bit-serial circuits this results in critical paths which are longer than expected for this architecture. The depth of these multiplexor elements is relatively constant regardless of digit-width. At W=3 the carry paths within the adder elements becomes longer than the multiplexor delay.

DW	Normalized Area-Time
1	1.0
2	0.638
3	0.737
4	0.919
5	1.09
6	1.13

Table 5.19: Final AT Products for Different Digit Width LDI Filters

The number of clock cycles required between input samples for each of the six filters is given in table 5.18. Here the W=2 filter requires approximately half the number of clock cycles as the bit-serial filter. However the higher digit width filters do not divide the number of clock cycles required by W as might be expected. The reason for this lies in the latency of the digit-serial DFIRST multipliers.

Digit-serial DFIRST multipliers ($W > 1$) have a latency of the CWL+1 while bit-serial multipliers have a latency of 3*CWL/2 +1 so after W=2 no further improvements are possible in the latency of DFIRST serial multipliers. What does decrease with increasing W is the number of clock cycles required to completely multiply or add two numbers together which is N/W. If each multiplier did not have any wait cycles then one new output would be generated every N/W clock cycles. but if this is not possible then the multiplier must wait for the next inputs to be valid before starting the next operation. Since the multipliers can not be fully utilized. the number of clock cycles does not decrease linearly with W.

With the circuit area. circuit critical path and the number of clock cycles per sample known. the final AT products for these filters can be computed as shown in table 5.19. Again all AT products are normalized to the bit-serial values for both technologies. The most AT efficient design is the W=2 digit-serial design.

5.6 Summary

Three fixed coefficient IIR filters, a fifth order bilinear LDI filter, a seventh order WAVE digital filter and a third order 'N2' filter generated using NOISEGEN were designed using using BITSYN, SNAFU and DIGIPARSE. The BITSYN and SNAFU synthesis programs generate resource sharing solutions while DIGIPARSE uses CSD multiplier coefficients and no resource sharing. SNAFU can generate arbitrary digit width solutions while the other methods are bit-serial only. The impulse response for all nine synthesized solutions was obtained using DSIM and the results matched the high level simulations of each filter generated using DIGICAP.

The nine test filters were mapped to the ACTEL technology using the TRANS logic mapping optimization. The resulting reductions in circuit area ranged from 44-50 percent, and critical path length reductions from 40 to 50 percent. The impulse response for the ACTEL implementation was generated using the gate level simulator logsim and were found to match the RTL DSIM simulations.

Using the redundant hardware removal operation (FANOUT=10) and the logic mapper total area reductions for the ACTEL circuits ranged from 48 to 58 percent and total critical path reductions ranged from 36 to 57 percent.

The nine IIR filters were mapped to the XILINX 4000 series technology using TRANS, the RAM optimization and the redundant hardware optimization with a FANOUT of 10. The addressing units allocated to each problem were customized to each application using the ADRUNIT TRANS parameter to obtain the smallest circuit area. The overall reductions in XILINX circuit area ranged from 33 to 52 percent for the nine test filters. The DFF/FG RAM mapping tradeoff figure ranged

from 3.25 to 7.8 DFFs/FG for these filters.

The area-time circuit efficiency parameter for each of the nine IIR filters was calculated. It was found that the DIGIPARSE solutions required the smallest circuit area. the smallest number of clock cycles and had the shortest critical path length for the three test IIR filters.

A fixed coefficient 61 tap bit-serial FIR filter was designed using FIRGEN. which implements all coefficients using SHIFTMULTs. The Data wordlength for this filter is 16 bits and the CWL is 12 bits. and no more than three coefficient bits are set in each coefficient. NOMAD was used to generate the fixed precision coefficients for the FIR filter.

The 61 TAP FIR filter was mapped to the XILINX 4000 technology and implemented on a single 4005PG156 device. The TRANS optimizations used included RAM mapping. redundant hardware removal (FANOUT 20), BLATCH mapping to clock enabled DFF elements and I/O ring mapping (54 I/O pins). The overall reduction in circuit area was 78.9 percent. The final FPGA resource utilization was 91 percent DFFs, 94 percent FGs and 84 percent IOBs. The FIR filter operated correctly at a 20 MHz bit clock rate, resulting in a sampling rate of 1.25 MSamples/s.

Six different digit width solutions, ranging from W=1 to W=6, for the fifth order LDI digital filter were synthesized using SNAFU. Each of these circuits was mapped to the ACTEL technology using the the TRANS hardware mapping optimization and the redundant hardware removal operation (FANOUT=10). Each of these six circuits was simulated using logsim and found to function correctly. The most AT efficient digit width was found to be W=2.

Chapter 6

Summary and Discussion

In this thesis a set of CAD tools for implementing bit-serial and digit-serial digital signal processing systems has been presented. The entry language for these CAD tools is the register transfer level hardware description langauge DFIRST. The DFIRST language is used to describe the interconnection and timing of language primitives such as multipliers, right shifters and adders. The DSIM event driven simulator is used to simulate DFIRST circuits. DSIM performs finite precision simulations and serial signal timing alignment verification on DFIRST circuits. The TRANS gate compiler is used to convert DFIRST circuits to technology specific gate level implementations. The first operation performed by TRANS is to convert DFIRST primitive elements to generic gate level implementations containing only simple gates and D-type flip-flop elements. A series of optimizations such as redundant hardware removal, RAM mapping for shift registers and hardware mapping to technology specific hardware elements are used to reduce final circuit implementation area.

Other bit-serial or digit-serial research efforts include PARSIFAL [HC90], CATHEDRAL [GM86] which has become part of a commercial tool available from Mentor Graphics Corporation, and work by K.K Parhi [Par91]. The DFIRST language is an extension of the FIRST [DR85] hardware description language. Additions to FIRST implemented by DFIRST include full digit-serial support, additional components such as parallel to serial and serial to parallel converters, and extended implementations of serial multipliers.

Bit-serial and digit-serial circuits are not commonly used in industry applications due to a shortage of effective CAD tools to aid in the rapid conversion of ideas to implementations, a longer learning curve as compared to parallel designs due to serial timing alignment complications, and a perception that serial processing systems must have a low throughput rate. One of the objectives of this research was to address these three concerns. The DFIRST language, DSIM simulator and TRANS compiler form an effective CAD environment for the implementation of many signal processing applications.

Serial processing circuits are ideal for low throughput applications where there is ample time between input samples to complete the necessary processing. In this environment the small operator size and simple routing requirements of serial designs can be exploited to create small efficient solutions. For higher throughput applications, bit-serial or digit-serial circuits are not necessarily slow. Consider the 61 tap FIR filter implemented in this thesis. Only 16 clock cycles are required between input samples resulting in a throughput rate of 1.25 M samples per second. Since bit-serial processing elments require a small circuit area, many separate elements can be used in parallel to complete the overall task. Therefore, serial architectures, implement serial data word, parallel operation execution, while parallel architecutures implement parallel data word, serial operation execution. If a large number of processing elements can be used as parallel processors then bit-serial or digit serial archituctures can be used to obtain relatively high throughput rates.

The VHDL [AG93] and Verilog [Pal96] behavioural hardware description languages are often used to describe, simulate and implement digital signal processing systems. Both of these languages use 'if then else' constructs, arithmetic operations,

and conditional operations to describe the behaviour of a digital circuit. Each of these operations are performed on parallel data signals of virtually any signal width. However these behavioural languages are not well suited to bit-serial or digit-serial circuit description. A bit-serial signal is communicated on only a single wire. so all operations within VHDL or Verilog must take place on this single wire. The benefits of high level constructs are lost somewhat, since an add becomes only a single full adder and a multiply becomes a single 'and' gate. As well. the notion of a data word is lost within these langauges. The full data word is broken down into several different signals separated by DFF elements. Under these conditions it is difficult to perform word level simulations and bit-serial timing alignment verification.

6.1 Future Work

The ultimate CAD design tool would convert a design specification. including timing constraints and available implementation technology details. into an optimal circuit implementation. The design specification must be entered in minimal time. the run time for the CAD tools must also be minimal and the created design must meet all timing specifications and occupy the smallest possible area in the technology used.

In every regard present day automated design tools can be improved. Attainable improvements to the tool set discussed in this thesis include design entry (DFIRST) and validation (DSIM) improvements which will expand the capabilities of the present system. The synthesis operation of converting the RTL description to a technology dependent design can be improved to support a greater variety of components and generate smaller more efficient hardware implementations for existing

components. Finally the set of optimizations used within TRANS can be extended to increase final solution quality. A set of possible improvements for the present CAD tools is presented in the following sections.

6.1.1 Design Entry and Validation

There are several short comings in the DFIRST language and the DSIM simulator which can be addressed. The most critical is that DFIRST and DSIM support mixed-mode circuits which combine digit-serial operations and gate level operations. Currently gate level operations occur on serial data words not on individual bits or digits within the word. True gate level simulation within the serial frame-work would remove the need for external gate level hardware to be designed using other gate level tools. This would significantly shorten design cycles for circuits containing a wide range of different hardware elements.

Another feature which could be added is multi-rate data support. Here different data rate circuits. as result from interpolation or decimation operations. can be described in a single file and simultaneously simulated. Supporting different data wordlengths within a single design instead of limiting the entire design to a single serial data word length could also be added.

The next generation of the DFIRST description language could be made completely behavioural. In this language generic operations such as '*' for multiply and '+' for addition would replace the instantiation of specific primitives. The context of each behavioural operation would be defined by the type (parallel or serial) and data width of the I/O signals on each operation. A '=' sign would serve as a format conversion operation. Another benefit of this type of behavioural language would be

that the serial control signals would not have to defined by the user. The compiler would derive the control structures as required. As well. the description of bit-serial. digit-serial or parallel architectures would appear the same. the only difference being the signal declarations for internal signals. The optimal form for this language would be similar to commonly used hardware description languages such as VHDL and Verilog with the only major difference being the inclusion of support for serial signals.

6.1.2 Synthesis

Presently several of the primitives within the DFIRST language such as SHIFT-MULTs. and MULTEXes are available only for the bit-serial data format. These elements can be extended to include all digit-widths. As well. several other primitives such as Dividers. Conditionals. and Square root operators can be made available in any digit width to increase the flexibility of the DFIRST language. Other more complex operations such as sine. cosine. and FFTs can also be added to the language. These more complex operators may require several parameters to fully describe the desired operator but their inclusion would significantly shorten design times for circuits requiring these functions. Memory interface components for external SRAM or DRAM components would also be useful.

Presently only one arbitrary digit-width multiplier structure has been used to implement these elements. This operator uses the non-redundant coefficient recoding in order to reduce multiplier size and shorten multiplier latency. Other recoding schemes should be investigated to improve the quality of arbitrary digit width multiplier primitives. Modified-booth recoding is commonly used in parallel multipliers

and will lead to improved multiplier performance, particularly for higher digit widths. Support of full parallel operators can also be added to increase the flexibility of the DFIRST language.

6.1.3 Optimization

The rule based mapping currently used by TRANS is effective for serial circuits designed with DFIRST since the structures generated are regular and contain mostly multiplexors, blatches and adders. If DFIRST is expanded to include true gate level parts, and full parallel operation the mapper must be improved to accommodate these more complex circuits. A hill climbing optimizer such as simulated annealing may be used in conjunction with the hardware mapper to obtain a smaller circuit. In addition the critical path information specifying input loads and output delay times should be linked with mapper so that the critical path can be shortened at the expense of hardware size or the critical path length can be made longer to allow for a smaller circuit. The path and area constraints must be user controlled. Re-timing which moves register elements forward and backward within the circuit may also be included to increase the effectiveness of the hardware mapper.

The RAM mapper can be improved so that an optimal addressing unit allocation is automatically generated by TRANS. Currently TRANS allocates a sub-optimal addressing unit scheme which must be fine tuned by a knowledgeable user to obtain the best result. As well, the range of addressing schemes can be increased to accommodate multiple addressing units per contiguous shift register chain. Presently each shift register chain is controlled by a single addressing unit. Simulated annealing may also be employed here to generate a more optimal solution.

Another feature which may be added is an automatic signal buffering option which automatically adjusts all internal driving points so that no node is over loaded. resulting in long delays for signal transitions on this node. This operation should be tied in with critical path delay estimates to be sure that the critical path of the circuit is as short as possible.

Bibliography

[ACT89] ACTEL. *Action Logic System Software Reference Manual.* Actel Corporation. 1989.

[AG93] J. R. Armstrong and F. G. Gray. *Structured Logic Design with VHDL.* Prentice Hall Professional Technical Reference. 1993.

[Ann86] M. Annaratone. *Digital CMOS Circuit Design.* Kluwer Academic Publishers. 1986.

[Bei94] E. Beingessner. Localization of sound using headphones. *Department of Electrical and Computer Engineering. University of Calgary. MSc. Thesis.* 1994.

[Bru75] L. T. Bruton. Low sensitivity digital ladder filters. *IEEE Transactions on Circuits and Systems vol. CAS-22. pp168-176.* Mar. 1975.

[BW92] M. Bauer and R. Weatley. Logsim user's guide. *Department of Electrical and Computer Engineering. University of Calgary. Internal Report.* 1992.

[CT88] C. W. Clenshaw and P. R. Turner. The symmetric level-index system. *IMA Journal of Numerical Analysis,* June 1988.

[Dev89] Analog Devices. *DSP Products Data Book.* Analog Devices. 1989.

[DR85] P. Denyer and D. Renshaw. *VLSI Signal Processing: A Bit Serial Approach.* Addison-Wesley Publishing Company. 1985.

[Erc84] M. D. Ercegovac. On-line arithmetic: An overview. *Real Time Signal Processing*, 1984.

[Fet86] A. Fettweis. Wave digital filters: Theory and practice. *Proceeding of the IEEE*, 1986.

[GK83] D. Gajski and R. Kuhn. Guest editor's introduction: New VLSI tools. *IEEE Computer*. 1983.

[GM86] J. Van Ginderdeuren and H. De Man. Application specific intregrated filters for hifi digital audio signal processing. *ICASSP*. 1986.

[Gra92a] P. Graumann. DFIRST Primitive Library. *Department of Electrical and Computer Engineering. University of Calgary. Internal Report*. 1992.

[Gra92b] P. Graumann. DFIRST User's Guide. *Department of Electrical and Computer Engineering. University of Calgary. Internal Report*. 1992.

[Gra92c] P. Graumann. TRANS User's Guide. *Department of Electrical and Computer Engineering. University of Calgary. Internal Report*. 1992.

[Gra92d] P. J. Graumann. Design and implementation of serial multipliers and dividers. *Department of Electrical and Computer Engineering. University of Calgary, Internal Report*, 1992.

[GT92] P. J. Graumann and L. E. Turner. Implementing DSP algorithms using pipelined bit-serial arithmetic and FPGAs. *First International ACM/SIGDA Workshop on FPGAs*, pages 123–128. 1992.

[Hay86] S. Haykin. *Adaptive Filter Theory.* Prentice Hall. 1986.

[HC90] R. Hartley and P. Corbett. Digit-serial processing techniques. *IEEE Transactions on Circuits and Systems.* 37(6):707–719. June 1990.

[HE89] B. S. Haroun and M. I. Elmasry. Architecture synthesis for DSP silicon compilers. *IEEE Transactions on Computer Aided Design.* 1989.

[Ins88] Texas Instruments. *Third Generation TMS320 User's Guide.* Texas Instruments.. 1988.

[Jac89] L. B. Jackson. *Digital Filters and Signal Processing. 2nd Edition.* Kluwer Academic Publishers. 1989.

[Joh92] B. Johnston. DSP system optimization using simulated annealing. *Department of Electrical and Computer Engineering. University of Calgary. MSc. Thesis.* 1992.

[Kac95] R. Kacelenga. Digital filter architecture design using primitives. network transforms and simulated annealing. *Department of Electrical and Computer Engineering, University of Calgary. PhD. Thesis.* 1995.

[LEL91] Y. C. Lim. J. B. Evans, and G. Liu. Decomposition of binary integers into signed power-of-two terms. *IEEE Transactions on Circuits and Systems.* 38(6):667–672. June 1991.

[Lew93] D. M. Lewis. An accurate LNS arithmetic unit using interleaving memory function interpolator. *Proceeding of the 11th Symposium on Computer Arithmetic,* June 1993.

[LM90] T. A. Ly and J. T. Mowchenko. Applying simulated evolution to data path allocation in high level synthesis. *Proceedings CCVLSI*. 1990.

[Log86] LSI Logic. *Data Book and Design Manual*. LSI Logic. 1986.

[Lyo76] R. F. Lyon. Two's complement pipeline multipliers. *IEEE Transactions on Communications*. pages 418–425, April 1976.

[MLD92] P. Michel. U. Lauther. and P. Duzy. *The synthesis approach to digital system design*. Kluwer Academic Publishers. 1992.

[Mot89] Motorola. *DSP56000/DSP56001 DSP User's Manual*. Motorola. 1989.

[Nag91] R. Nagalla. Synthesis of DSP systems using pipelined bit-serial arithmetic. *Department of Electrical and Computer Engineering. University of Calgary. MSc. Thesis*. 1991.

[ND90] J. P. Neil and P. B. Denyer. Exploring design space using SAVAGE: A simulated annealing based vlsi architecture generator. *33rd Midwest Symposium on Circuits and Systems*. 1990.

[Neo94] Neocad. *Neocad User manual*. Neocad Inc.. 1994.

[NT91] R. Nagalla and L. E. Turner. Pipelined bit-serial synthesis of digital filter algorithms. *Proc. of the IFIP TCIO/WG 10.5 International Conf. VLSI*. 1991.

[Obe79] R. M. M. Oberman. *Digital Circuits for Binary Arithmetic*. Macmillan Press Ltd.. 1979.

[OS89] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing.* Prentice Hall. 1989.

[Pal96] S. Palnitkar. *Verilog HDL: A Guide to Digital Design and Synthesis.* Prentice Hall Professional Technical Reference, 1996.

[Par91] K. K. Parhi. A systematic approach for design of digit-serial signal processing architectures. *IEEE Transactions on Circuits and Systems.* 38(4):358–375, April 1991.

[Pat93] E. Patton. Digital hardware rake transceiver. *Department of Electrical and Computer Engineering. University of Calgary. MSc. Thesis.* 1993.

[PGT95] G. Panneerselvam. P. J. W. Graumann. and L. E. Turner. Implementation of fast fourier transforms and discrete cosine transforms in FPGAs. *5th International Conference on Field-Programmable Logic and Applications.* 1995.

[PK87] P. G. Paulin and J. P. Knight. Force-directed scheduling in automatic data path synthesis. *24th ACM/IEEE Design Automation Conference.* 1987.

[PM89] K. K. Primlani and J. L. Meador. A nonredundant radix-4 serial multiplier. *IEEE Journal of Solid State Circuits.* 24(6):1729–1736, December 1989.

[PPM86] A. C. Parker. J. T. Pizarro, and M. Mlinar. Maha: A program for datapath synthesis. *IEEE 23rd Design Automation Conference.* 1986.

[SJJT86] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor. *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing.* IEEE Press, 1986.

[SMD87] S. G. Smith, M. S. McGregor, and P. B. Denyer. Techniques to increase the computational throughput of bit-serial architectures. *International Conference on Acoustics, Speech and Signal Processing*, 1987.

[Svi91] M. J. Svihura. NOMAD User's Guide. *Department of Electrical and Computer Engineering, University of Calgary, Internal Report*, 1991.

[Syn91] Synopsys. *Synopsys Design Compiler Reference Manual Version 2.2*. Synopsys Inc., 1991.

[TG95] L. E. Turner and P. J. W. Graumann. Rapid hardware prototyping of digital signal processing systems using field programmable gate arrays. *5th International Conference on Field-Programmable Logic and Applications*, 1995.

[TGG95] L. E. Turner, P. J. W. Graumann, and S. G. Gibb. Bit-serial FIR filters with csd coefficients for FPGAs. *5th International Conference on Field-Programmable Logic and Applications*, 1995.

[Tur88] L. E. Turner. Digicap user's guide. *Internal Report*, 1988.

[Vie91] ViewLogic. *VIEWlogic Reference Manual.* VIEWlogic Systems, Inc. 1991.

[Vie93] Viewlogic. *Workview plus on Windows: VHDL reference manual and user's guide.* Viewlogic, 1993.

[Wor92a] S. D. Worthington. Limit cycles and sinusoidal oscillations in digital systems. *Department of Electrical and Computer Engineering. University of Calgary. MSc. Thesis*, 1992.

[Wor92b] S. D. Worthington. A new digital sinusoidal oscillator. *Canadian Microelectronic Coprorations. TEXPO presentation*, 1992.

[XIL94] XILINX. *XILINX Programmable Logic Data Book*. Xilinx Inc., 1994.

[YG86] T. Yoshimura and S. Goto. A rule-based and algorithmic approach for logic synthesis. *International Conference on Computer Aided Design*. 1986.

Appendix A

61 Tap FIR Filter Coefficients

61 Tap Filter fixed point coefficients, 12 bit CSD values. The coefficients are centered about the center tap (C[030]) to gaurantee linear phase operation.

```
C[000] = 9     =  0.00439453125
C[001] = -4    = -0.001953125
C[002] = -18   = -0.0087890625
C[003] = 5     =  0.00244140625
C[004] = 16    =  0.0078125
C[005] = -2    = -0.0009765625
C[006] = -5    = -0.00244140625
C[007] = -2    = -0.0009765625
C[008] = -19   = -0.00927734375
C[009] = 6     =  0.0029296875
C[010] = 40    =  0.01953125
C[011] = -9    = -0.00439453125
C[012] = -39   = -0.01904296875
C[013] = 6     =  0.0029296875
C[014] = 14    =  0.0068359375
C[015] = 0     =  0.0
C[016] = 38    =  0.0185546875
```

```
C[017] = -8    = -0.00390625
C[018] = -80   = -0.0390625
C[019] = 6     =  0.0029296875
C[020] = 76    =  0.037109375
C[021] = -4    = -0.001953125
C[022] = 0     =  0.0
C[023] = -3    = -0.00146484375
C[024] = -144  = -0.0703125
C[025] = 10    =  0.0048828125
C[026] = 321   =  0.15673828125
C[027] = -11   = -0.00537109375
C[028] = -464  = -0.2265625
C[029] = 5     =  0.00244140625
C[030] = 519   =  0.25341796875
C[031] = 5     =  0.00244140625
C[032] = -464  = -0.2265625
C[033] = -11   = -0.00537109375
C[034] = 321   =  0.15673828125
C[035] = 10    =  0.0048828125
C[036] = -144  = -0.0703125
C[037] = -3    = -0.00146484375
C[038] = 0     =  0.0
C[039] = -4    = -0.001953125
C[040] = 76    =  0.037109375
```

C[041] = 6 = 0.0029296875

C[042] = -80 = -0.0390625

C[043] = -8 = -0.00390625

C[044] = 38 = 0.0185546875

C[045] = 0 = 0.0

C[046] = 14 = 0.0068359375

C[047] = 6 = 0.0029296875

C[048] = -39 = -0.01904296875

C[049] = -9 = -0.00439453125

C[050] = 40 = 0.01953125

C[051] = 6 = 0.0029296875

C[052] = -19 = -0.00927734375

C[053] = -2 = -0.0009765625

C[054] = -5 = -0.00244140625

C[055] = -2 = -0.0009765625

C[056] = 16 = 0.0078125

C[057] = 5 = 0.00244140625

C[058] = -18 = -0.0087890625

C[059] = -4 = -0.001953125

C[060] = 9 = 0.00439453125