Digital Hardware Design Decisions and Trade-offs for Software Radio Systems

John Patrick Farrell

This thesis is submitted to the Faculty of
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Electrical Engineering

Jeffery H. Reed, Chair
Timothy Pratt
Cameron Patterson

May 21, 2009
Blacksburg, VA

Keywords: Software Radio, Digital Hardware, Signal processing

Digital Hardware Design Decisions and Trade-offs for Software Radio Systems

John Patrick Farrell

ABSTRACT

Software radio is a technique for implementing reconfigurable radio systems using a combination of various circuit elements and digital hardware. By implementing radio functions in software, a flexible radio can be created that is capable of performing a variety of functions at different times. Numerous digital hardware devices are available to perform the required signal processing, each with its own strengths and weaknesses in terms of performance, power consumption, and programmability. The system developer must make trade-offs in these three design areas when determining the best digital hardware solution for a software radio implementation.

When selecting digital hardware architectures, it is important to recognize the requirements of the system and identify which architectures will provide sufficient performance within the design constraints. While some architectures may provide abundant computational performance and flexibility, the associated power consumption may largely exceed the limits available for a given system. Conversely, other processing architectures may demand minimal power consumption and offer sufficient computation performance yet provide little in terms of the flexibility needed for software radio systems. Several digital hardware solutions are presented as well as their design trade-offs and associated implementation issues.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Goals

With the expanding mobile environment evolving in today's society, wireless communications is being seen in a rapidly increasing number of areas and is progressively advancing mobile functionality. Several industries that did not previously incorporate communication systems are now including the functionality in their products and wireless radio systems are continually offering increased data rates to facilitate added capabilities. These advancements are driven by the high computational processing power offered in the latest digital hardware devices. When designing wireless communication systems that utilize these digital hardware devices, engineers are interested in offering a flexible, high-performance solution within their design constraints.

The design constraints given to engineers are based on the conditions in which the radio system will be used as well as the available resources needed to operate the radio. Design constraints generally come in terms of performance, size, weight, power, and cost requirements. When considering digital hardware choices for flexible systems, additional design metrics must be considered such as flexibility, scalability, and portability that directly affect the other design constraints given to the engineer. The goal of this thesis is to describe the available digital hardware architectures, help identify the design metrics and constraints associated with each option, and provide a methodology for making design decisions in a software radio system. Several implementation issues will be described that affect the development process and some design principles will be laid out that help to provide an efficient design. As flexible software radio solutions become more prevalent in wireless communications, these design principles will be increasingly important for the engineers designing the systems.

1.2 Organization of Thesis

Following this introductory chapter, this thesis presents an overview of the digital hardware solutions for software radio systems in Chapter 2. In Chapter 3, digital signal processor architectures are identified and several of the programming techniques are discussed including pipelining and real-time operating systems. The chapter also contrasts digital signal processors

with general-purpose processors for use in software radios. Chapter 4 discusses field-programmable gate array architectures, including structure, processing elements, and design techniques. Chapter 5 summarizes the implementation trade-offs associated with the various digital hardware solutions and the author offers a methodology making decisions on what processing resources to use in a software radio system. Power consumption issues and efficiency techniques are identified in Chapter 6 while the interprocessor communication concerns are listed in Chapter 7. Chapter 8 discusses the techniques for testing and validating digital hardware systems. Chapter 9 presents conclusions and the author offers a prediction of future digital hardware use in software radios.

Chapter 2

Digital Hardware for Software Radios

2.1 Overview

Digital hardware solutions are critical components to the design and operations of software radios and they determine the radio's capabilities and performance. Since radio operations are defined by programmable designs running on digital hardware, greater flexibility is available for implementing various wireless standards and waveforms. Digital hardware is available in various forms on single-chip custom integrated circuits (ICs), of which the most commonly used for software radio are digital signal processors (DSPs), field programmable gate arrays (FPGAs), general-purpose processors (GPPs), and application specific integrated circuits (ASICs). Additionally, hybrid combinations of these devices and multi-core processing structures are becoming increasingly available on single-chip solutions, thereby providing added performance and re-configurability. Analyzing the trade-offs among these options and determining the best digital hardware solution for a software radio system is a complex and challenging task for system designers. Flexibility, portability, scalability, and performance are four main interrelated and often opposing issues important in the design decisions for the digital hardware composition of a software radio.

Flexibility is the ability to adapt to various current and future wireless communication standards and protocols and helps to facilitate short innovation cycles by allowing integration of new features through successive product generations. Additionally, flexibility offers options for integrating system and wireless standard updates for increasing quality and performance, thereby reducing the rate of obsolescence. Run-time adaptation can also facilitate dynamic switching between different standards (i.e. hand-offs between GPRS, EDGE, and/or UMTS) [Blume 02].

Portability is the ease with which a system or component can be transferred from one hardware or software environment to another [IEEE 90] and its goal is the design of software code than can be easily ported to another hardware architecture; non-portable code often requires more design time than fully re-writing the code, and thus leads to increased cost. By designing radio software in a modular format, with a system composed of discrete components, various levels of portability can be achieved for implementation across hardware architectures.

Scalability allows the digital hardware to provide a range of capabilities depending on the software and hardware configuration of the system. Highly scalable digital hardware architectures can satisfy complex system requirements but also handle less demanding, more cost sensitive requirements [Pulley 08]. For instance, a software defined cellular base station can utilize energy-scalable algorithms to provide a variable number of communication channels based on demand, thereby trading capacity for energy cost. In addition, in a handheld device, scalability can be utilized to vary the rate at which equalizer coefficients are updated based on the error rate, trading quality for energy efficiency [Dejonghe 08].

Performance determines how successfully and efficiently the digital hardware implements radio functions. Moore's Law, which states that the number of transistors on a chip will double about every 18 months, has been a driving factor in increasing digital hardware performance. More transistors and higher speed operations leads to more demanding applications with higher data rates. Nevertheless, performance is not only based on computational data throughput but also based a number of factors including energy efficiency, cost, and area requirements. It is directly related to flexibility, portability, and scalability; and trade-offs must be made to provide a balanced implementation.

Each of the four main digital hardware categories offers varying levels of re-programmability and performance. Instruction-based architectures offer the greatest flexibility and can be programmed in familiar high-level languages such as C/C++. GPPs provide the most flexible architecture but guaranteeing execution time in GPPs is difficult, so DSP processors are often used to provide a slightly less flexible yet more deterministic implementation. DSP processors can still be programmed in high-level languages but need architecture specific programming to achieve additional performance. FPGAs offer even better performance using a highly parallel structure with computationally-intensive resources, but require specialized programming tools and knowledge, thereby reducing flexibility for added capability. ASICs offer the most optimized and efficient digital hardware implementation but are often a poor choice for software radio since their design is in fixed silicon with little to no flexibility.

For the latest wireless standards, the four main digital hardware categories (GPPs, DSPs, FPGAs, and ASICs) provide inadequate computational capability and require a combination of technologies for implementation. These new wireless standards require FPGA or ASIC assistance for high throughput simple functions and DSP software for complex control functions [Pulley 08]. However, interconnect bottlenecks between dedicated ICs reduce performance, so hybrid, multi-core, and graphical processing unit (GPU) structures on chip are becoming popular. Multi-core

architectures offer high performance parallel resources with high-level language programming environments. By utilizing these single-chip devices, the new architectures offer a better balance of high speed processing, power consumption, and development effort compared to multi-chip designs.

2.2 Overview of Key Hardware Elements

Modern software radio systems require substantial digital processing power for implementing flexible and highly capable systems. This digital processing can be implemented with various hardware components including DSP processors, GPPs, FPGAs, and ASICs. Each hardware choice provides various levels of performance, flexibility, and programmability that must be evaluated early in the development process when designing hardware for a software radio system. Once hardware choices are made that will satisfy processing requirements, the development process moves to computer-aided design (CAD) of the circuit logic and drawing of a high-level circuit schematic. After the circuit design is considered functional, often after several design revisions, the layout of a printed wiring schematic and manufacturing of the printed circuit board (PCB) can be completed. Following manufacturing, the final design step is to solder the hardware components on the PCB, load the program code into the hardware, and test the final product.

The hardware components used in software radio systems are diverse in their architecture and functionality. Some provide high performance solutions with little in-field flexibility, while others maintain high flexibility at the expense of performance or power trade-offs. Processors are typically considered the most flexible and programmable hardware elements while ASICs maintain a fixed architecture after development. FPGAs fall in the middle of this flexibility versus performance curve, providing a configurable high-performance solution but with mostly static functionality during run-time. However, as technology develops, hardware components are blurring the lines between performance and flexibility. For example, ASICs can contain some partially reconfigurable logic and FPGAs offer run-time reconfiguration options to facilitate flexibility.

DSP processors are microprocessors designed for digital signal processing applications, where high-performance repetitive mathematical operations are the primary function. DSP processors come in a variety of architectures with a trend toward multiple-processing cores on a single chip. They can be programmed through both assembly language and high-level languages (HLLs), making development easier than on hardware requiring more specialized languages, such as FPGAs or ASICs. Since high-performance applications often require significant processing power,

DSP architectures generally include several parallel processing elements to accelerate DSP algorithms. The parallel processing elements typically offer cycle efficient computations, thereby reducing processing time per sample. The multiply-accumulate (MAC) operation, where two operands are multiplied together and added to previous multiplication results, is one common task, and therefore it is often one of the parallel processing elements available on a DSP. Other parallel processing elements include additional arithmetic logic units and hardware acceleration blocks. While significant processing power may be available on DSP processors, their architecture is fixed by the number of hardware blocks. This might provide a less than optimal solution for a processing operation since additional clock cycles may be required to process a data sample. While DSP processors provide one processing solution, other hardware choices are available that may provide a more efficient and optimized design, yet often at the expense of programmability.

FPGAs are hardware devices that provide a matrix of reconfigurable logic resources on a single chip, which can be programmed to implement a user specified circuit. When FPGAs were first introduced in the 1980s, they were designed as a configurable solution to glue-logic, which links discrete hardware components such as an ADC and a DAC together. While they still maintain this functionality in many systems, the architecture has improved to provide processing resources capable of large-scale DSP tasks. Inside an FPGA, multiple hardware resources can be configured for processing various concurrent tasks while maintaining an optimal word size for a particular algorithm. However, since FPGAs are highly configurable, they are difficult to program based on hardware description language (HDL) requirements. This can be a deterring factor when choosing an FPGA for a system since it can increase development costs and time-to-market.

GPPs are a relatively new player to the software radio field because they have historically offered poor performance for real-time signal processing tasks. GPPs are generally designed for high performance computing solutions performing mostly case-based reasoning operations. They are typically not optimized for the predictable and time-sensitive real-time software radio computational tasks, as operation times are often variable. However, as processor architectures have developed, GPPs have gained additional performance capabilities attributed to parallel processing blocks, higher clock rates, and large on-chip memories. Several of the architectural features once only available on DSP processors are now available on GPPs, and as such, GPPs are being used in more software radio applications.

ASICs are generally the most powerful and computationally efficient hardware element designed for signal processing applications, but their use in software radio is diminishing due to limited flexibility. ASICs are designed with HDL tools that implement circuits similar to FPGA

design, but their hardware implementation is configured once at the silicon foundry during manufacturing. While some ASICs can be designed with partially reconfigurable regions, their overall structure is fairly static. Therefore, this is an issue for software radios since the ability to alter the algorithm implemented on the ASIC is unavailable. Custom ASICs are generally only used to provide added processing power when no other option is available due to design constraints or when designing sufficiently high volume systems. Standard ASIC circuits are also available for specified applications such as down conversion on the radio front end or to implement specific standards such as W-CDMA. These standard ASIC circuits are still often utilized in software radio designs but their use may diminish with advancing technology.

In the following chapters, an overview of the hardware architectures of DSP processors, multi-core processors, general-purpose processors, and FPGAs will be discussed along with their associated design trade-offs. ASICs will be mentioned in many of the discussions but due to their vast architectural differences and diminishing use in software radios, a full discussion on the subject is not provided.

Chapter 3

Digital Signal Processors

3.1 Introduction to DSP Processors

Digital signal processors (DSPs) are microprocessors with specialized architectures designed to efficiently implement computational algorithms with high performance I/O. Many application areas, such as wireless communications, audio and video processing, and industrial control systems now require digital signal processing as a core technology. These DSP applications often require higher performance execution than is typically found on standard microprocessors; therefore, DSP processor architectures provide optimized support for the high performance, repetitive, and numerically intensive mathematical manipulation of digital signals [Lapsley 97]. To support these high performance mathematical tasks, DSP processors have several features tailored directly to DSP operations, such as hardware multipliers, dedicated address generation units, and large accumulators. Additionally, special instructions are available for common DSP operations, often with the ability to execute in parallel for faster execution. While these specialized functions can significantly increase algorithm execution, the memory access time for obtaining inputs and storing results can be a bottleneck for the system. Consequently, most architectures offer several memory accesses in a single clock cycle, where instruction and data operands can be obtained. Since numerous DSP processors are available for a variety of applications; it is important to consider factors such as architecture, speed, multi-processing capabilities, MAC units, instruction set, power consumption, and cost before selecting a DSP processor for a particular application.

In this section, several DSP processor architectures will be described, identifying features such as parallel processing units, memory buses and address generation units. The analysis will include a description of the available numeric representation methods and associated arithmetic operations. The pipelining technique will be demonstrated to indicate how processing resources can be better utilized to yield higher computational performance. Since multi-core processing is another method of increasing computational performance and offering several cores on a chip is a relatively new approach, an architectural example will be shown. Development methods using real-time operating systems will be discussed and benchmarking techniques for evaluating DSP processor performance in software radio applications are identified.

3.2 DSP Core

DSP processors can be packaged in many forms, but the main computational data path that includes an arithmetic logic unit (ALU), accumulators, multipliers, shift registers, and data and address buses is called the DSP core [Lapsley 97]. The central processing unit (CPU) inside the DSP core is responsible for performing cycle efficient computational operations on signal data. Many of the computational operations involved in DSP algorithms, such as digital filters, Fourier transforms, and correlations, require a vector dot product. The vector dot product is implemented with a multiply-accumulate (MAC) operation, expressed as $a \leftarrow a + b \bullet c$ [Cofer 06]. To facilitate a fast and efficient algorithm execution, at least one MAC unit is typically included in the CPU's data path and higher performance DSP cores include several MAC units for executing operations in parallel. Other hardware features are also available in the DSP core that facilitate efficient computation operations and will be discussed in the following section.

DSP cores are the building blocks for DSP processors. While DSP cores can be packaged and sold by themselves as DSP processors from companies such as Texas Instruments and Analog Devices, they can also be used as building blocks in other silicon chips. It often may be more efficient for a system to include a DSP core as one element in the overall chip design, thereby keeping as many signals on chip as possible to reduce communication overhead, bottlenecks, and power consumption at the external pins. The overhead for other elements such as interface logic, memory, peripherals, and custom units can all be included in a small efficient package [Lapsley 97]. Designing these custom silicon circuits with DSP cores and other logic on a single chip can be done on a custom ASIC, but it is often more effective on an FPGA implementation for flexibility in a software radio system.

3.3 DSP Processor Architectures

DSP processors have architectures that are molded to efficiently execute DSP algorithms and most of the available features are dictated by the algorithms that are processed on the hardware [Donovan 06]. The performance and price range among devices vary widely, utilizing architectures that range from low-cost serial data path structures to extremely high performance, high-cost parallel configurations [Eyre 00]. In this section, the types of DSP processor architectures will be discussed along with their associated design trade-offs. As technology develops, manufacturers are consistently refining their products for better performance or lower power operation; so the vendor's website should be viewed for the latest available information.

The DSP processor architecture utilizes a DSP core connected to several on-chip resources such as program and data memory. The functionality of the processor architecture varies based on several factors including the instruction set used to execute operations, the available parallel processing units, the organization of the memory sub-system, and the native numeric data format.

The organization of a processor's memory sub-system can have a large impact on performance [BDTI 00]. Shown in Figure 1, early general-purpose microprocessors utilized the von Neumann architecture, which provides a single bus connection to memory, thereby allowing a single memory access per clock cycle. This structure is inefficient for executing many of the DSP operations required in software radio systems. As mentioned previously, the MAC unit is fundamental to many signal processing algorithms. For efficient execution of the MAC unit, an instruction word and two data word fetches must be made to memory in a single clock cycle [Eyre 00]. An additional memory access per clock cycle is also needed to store the result of the multiply operation for an optimal solution. The von Neumann architecture is not sufficient to support this capability; therefore most modern DSP processors utilize the Harvard architecture or a modified Harvard architecture.

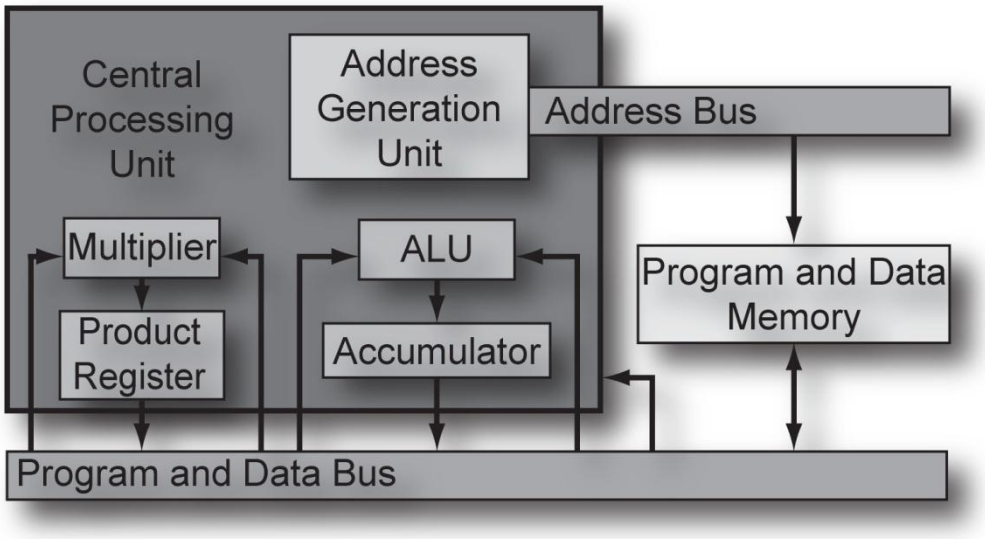

Figure 1: Standard DSP von Neumann Memory Architecture

In the Harvard architecture, separate memory banks are included for program and data memory. The processor can simultaneously access two or more of these separate memory banks through separate communication buses, thereby loading data operands[1] and fetching instructions concurrently. A generic Harvard architecture is shown in Figure 2. Many variants of the Harvard

[1] Operands refer to the both the input data to be processed by an instruction as well as the output data that results from the instruction's execution [Lapsley 97].

architecture are available and most DSP processors actually use a modified Harvard architecture. A modified Harvard architecture allows the instruction memory to be accessed as if it were data memory, and vice versa. Additionally, the independent memory banks and buses allow pipelining to be used to increase program execution speed, which will be discussed in Section 3.6 Pipelining.

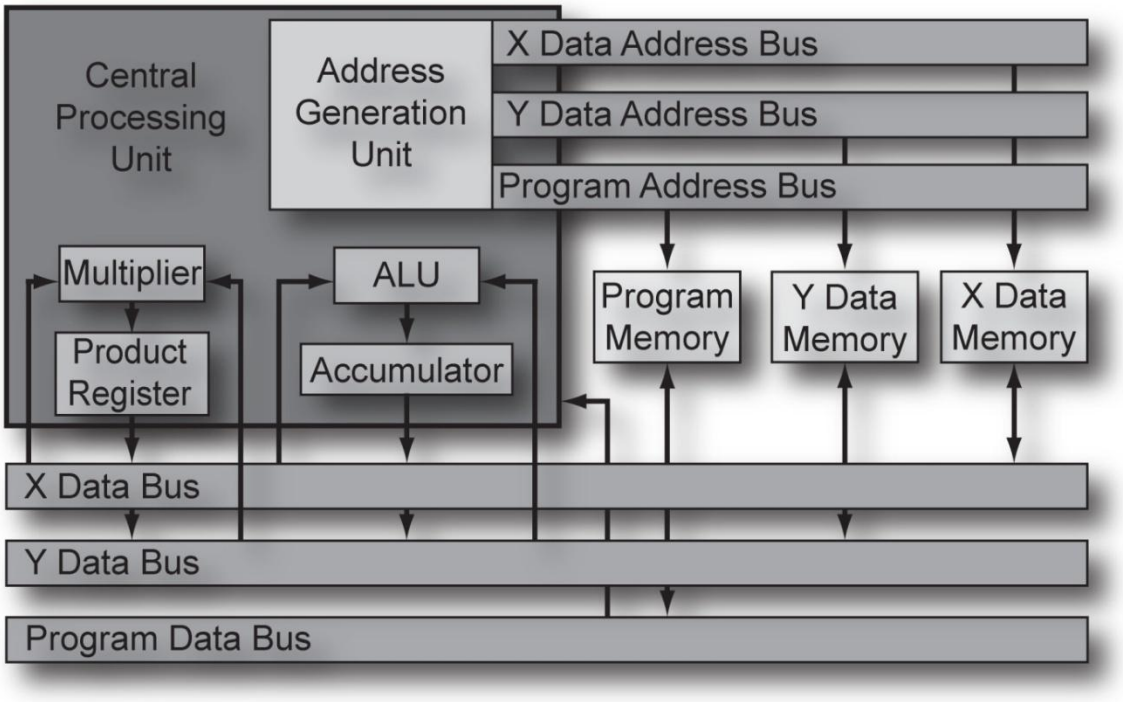

Figure 2: Generic Harvard Architecture using dual-data buses and a program bus

Uniscalar Architecture

DSP processor CPUs that have a small number of processing elements and can execute one instruction per clock cycle are referred to as conventional DSP processors utilizing the uniscalar architecture. The uniscalar architecture uses the modified Harvard memory structure and generally includes a single multiplier unit, an ALU, and possibly a shifter unit. The low-end conventional DSP processors typically have low clock rates and minimal additional execution units while mid-range versions offer higher clock rates, some additional hardware to improve performance, and deeper instruction pipelines [Eyre 00]. Since one instruction can be executed per clock cycle, the maximum processing performance is largely limited by the chip clock rate. For example, as of 2009 the Analog Devices ADSP-21xx family offers uniscalar processors operating between 75-160 MHz, yielding performance between 75-160 million instructions per second

(MIPS), respectively [Analog 09]. These conventional DSP processors provide respectable DSP performance while maintaining modest memory and power consumption requirements.

Very Long Instruction Word (VLIW) vs. Superscalar

Since conventional DSP processor clock rate increases are limited, additional methods must be exercised to gain more useful processing work out of each clock cycle. One approach is to extend the architecture to include parallel execution units and modify the processor instruction set. The VLIW architecture utilizes instruction level parallelism to execute multiple independent operations on separate processing units. VLIW DSP processors typically contain between four and eight independent processing (functional) units in the DSP core. The execution of these processing units during each clock cycle is dependent on one super-long instruction word that contains independent sub-instructions, corresponding to each unit [Eyre 00]. The sub-instruction groupings are done during software development and compile time with the aid of code generation tools; therefore the grouping does not change during program execution. The long instruction words make more uniform register sets and simpler instructions possible at the cost of added code size and memory usage. These added requirements translate into higher memory bandwidths, bus widths, and clock rates that increase power consumption over a conventional DSP processor [Donovan 06]. The Texas Instruments TMS320C6424 is one example of a VLIW DSP Processor. It contains eight independent functional units; six ALUs supporting single 32-bit, dual 16-bit, or quad 8-bit arithmetic and two multipliers supporting four 16 x 16-bit multiplies or eight 8 x 8-bit multiplies per clock cycle [TI 08a]. With a clock rate of 700 MHz, these functional units in the DSP core can provide a total of 2800 million MACs per second (MMACS) with four 16-bit multiply operations or 5600 MMACS with eight 8-bit multiply operations [TI 08b].

Similar to the VLIW architecture, the superscalar architecture also provides parallel execution units. However, instead of grouping instructions during program assembly, the grouping is done during run-time. Superscalar architectures have dedicated circuitry for determining which instructions will be executed in parallel based on data dependencies and resource contention [Donovan 06]. Therefore during different repetitions of the same segments of code, the processor may create different instruction groupings. This dynamic scheduling of parallel operations can be challenging for the programmer since it may be difficult to predict the execution time of a certain segment of code [Eyre 00]. Consequently, the lack of timing predictability has limited Superscalar architecture acceptance to date in DSP processors, despite easier programmability than their VLIW counterparts [Donovan 06].

Single Instruction, Multiple Data (SIMD) and Multiple Instruction, Multiple Data (MIMD)

SIMD and MIMD are architectural techniques used with DSP processors that identify how parallel processing units are controlled and communicate with each other [BDTI 07]. In the SIMD approach, the parallel processing elements are controlled by a single instruction stream and operate concurrently. As shown in Figure 3, during each time instant, a single instruction commands the processing elements to perform the same operation, but utilize their own input data and generate their own output [BDTI 07]. Using this technique, only one hardware unit is needed to decode the instructions for the parallel processing units. Additionally, by utilizing only one instruction word for several parallel processing elements, the instruction memory requirements are reduced over standard VLIW and Superscalar implementations. For instance a 32-bit VLIW processor with four parallel execution units requires a 128-bit instruction; this can be reduced to a 32-bit instruction word in an SIMD implementation, a reduction by a factor of four. These two factors can potentially reduce the required chip size, power consumption, and memory utilization. As an example, an SIMD multiplication instruction could perform two or more multiplications on different input operands in a single clock cycle [Eyre 00]. This approach is most useful for performing the same, repetitive computation across vector data, often found in signal processing systems, such as filtering.

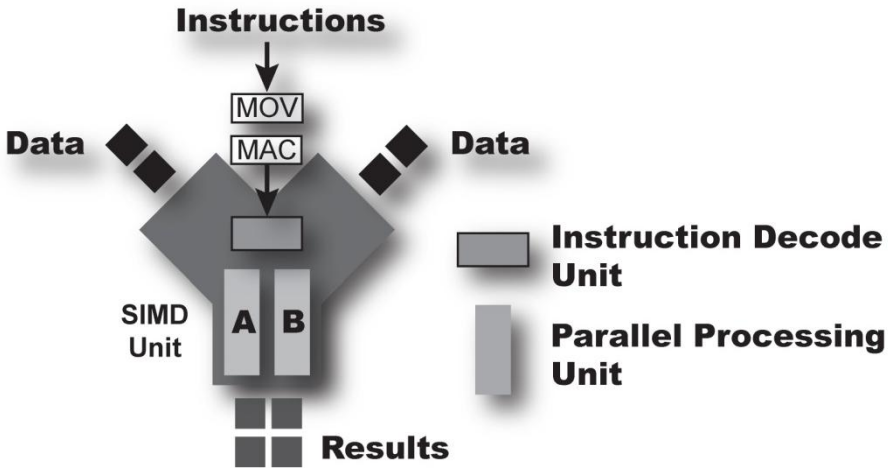

Figure 3: SIMD architectural approach with one instruction word per multiple data words

In the MIMD approach, the processing elements are full processors that execute their own instruction stream. The architecture may follow a homogeneous structure where all the processors are the same or a heterogeneous structure where two or more distinctly different types of processors are available [BDTI 07]. MIMD machines usually operate asynchronously, eliminating

the need for a single clock for all the processing elements. There is often confusion about the differences between and MIMD machine and a multi-processor approach. When considering an on-chip multi-processor approach, the two names are synonymous but when using separate chips it is considered a multi-processor approach. Programming these devices is generally more difficult than SIMD machines since partitioning applications and ensuring correct data flow is a challenge. However, since the parallelism doesn't require multiple, identical operations, it is more flexible than SIMD and can parallelize a wider range of applications [BDTI 07]. The picoArray processor, as discussed in Section 3.7 Multi-processing and MIMD Architecture Examples, is an example of a MIMD machine.

3.4 Numeric Representation

DSP processors and other digital hardware devices implement numerical values and arithmetic operations using either fixed point or floating point numeric representation. Fixed point processors represent and manipulate numbers as integers while floating point processors use a combination of mantissa (or fractional part) and an exponent for representation [Cofer 06]. One of the key differences between the two formats is the available dynamic range, which is the ratio of the maximum to minimum absolute values that can be represented as a number. The floating point format offers a much larger dynamic range and is more high-level language friendly than fixed point but is expensive in terms of silicon area and complexity [Cofer 06]; therefore, most DSP processors utilize the fixed point representation. The format and trade-offs associated with each representation will be explained in the following subsections.

Fixed Point

Fixed point is regularly used as the numeric format in digital hardware for representing numbers as either signed integers or signed fractions. To represent positive and negative numbers in fixed point, sign magnitude, one's complement, or two's complement format is used. Additionally, in the fixed point format, there is an implied binary point (also known as a radix point) that separates integer and fractional parts. In a fixed point integer, this radix point is to the right of the least significant bit where as in fixed point fractions, the radix point is between the sign bit and the most significant magnitude bit [Nelson 95]. A combination of integer and fractional parts can also be represented by shifting the radix point in the fixed point fraction to the right. Since a limited number of possible values are available to represent a number in fixed point, the

desired number must be quantized to fit in the available dynamic range by rounding-off and/or truncating to the nearest fixed point value.

Fixed point DSP processors are popular in wireless communication systems since they typically offer a higher speed, lower cost, and lower power alternative to their floating point counterparts. These benefits result from the reduced silicon complexity required for fixed point arithmetic. However, the advantages are traded off against added design effort required to prevent accumulator overflow and implement fixed point algorithms [Cofer 06]. Consequently, the programmer must pay attention to numeric scaling that will be discussed in the following subsection. The word sizes vary among processors but a 16-bit word using two's complement format is frequently used [Cofer 06]. When performing arithmetic operations, the interim products will often be larger than the original operands. For that reason larger accumulators are attached to the outputs for storing interim results. With a 16-bit word, a 32-bit attached accumulator is common to provide added precision, helping to prevent overflow and saturation conditions from occurring during arithmetic operations.

Figure 4 depicts an example for a 16-bit two's complement fixed point integer. The sign bit is indicated as *S* while *x* denotes a binary value of 1 or 0. In this example, the sign bit is located in bit 15 and the radix point is located to the right of the least significant bit; as a result integer values, N, range from -32768 ≤ N ≤ 32767.

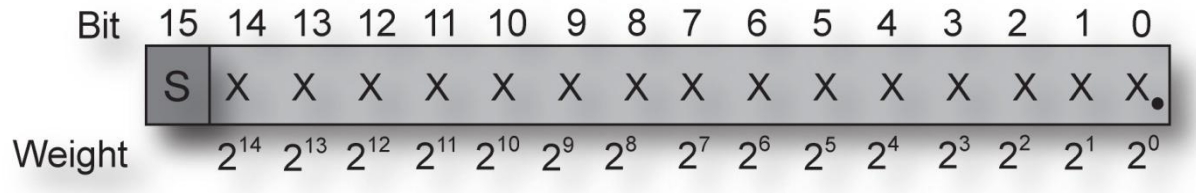

Figure 4: 16-bit Two's Complement Fixed Point Integer Representation

The Q-format is a method for tracking the relative location of the radix point within an arithmetic input value and operation result [Cofer 06]. During some arithmetic operations, such as multiplication, the radix point can shift in the result; thus, to accurately use the resulting value, the location of the radix point must be known. In the Q-format, the number of bits to the left and right of the radix point is indicated. Figure 5 depicts a 16-bit fractional fixed-point representation, also known as Q1.15 since the radix point is indicated to the right of bit position fifteen, between the sign bit and the most significant magnitude bit. When the word length is fixed and known, the bits to the left of the radix point are often excluded from the indicator, so Q1.15 would be shown as Q15. In this Q15 format, the fractional numbers have fifteen bits of precision and a sign bit is included,

thus values from -1 < N < 1 can be represented. However, it should be noted that in the maximum binary fractional value for this representation is 0.999969482 and not actually 1.

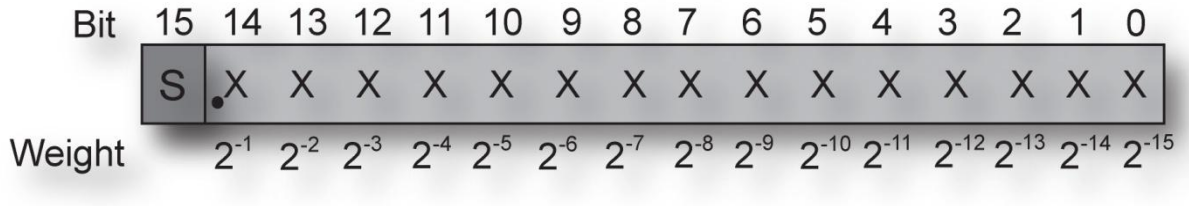

Figure 5: 16-bit Signed Fractional Fixed Point (Q15 Representation)

In Figure 6, a fixed point representation for a 16-bit mixed integer/fractional value is shown. The radix point is to the right of bit 13, thus this is recognized as a Q13 number. In the Q13 two's complement format, there is a sign bit, two bits of integer values, and thirteen bits of fractional representation, thus integer values with fractional portions can be represented from -4 ≤ N ≤ 4 with 15 bits of precision [Reed 02].

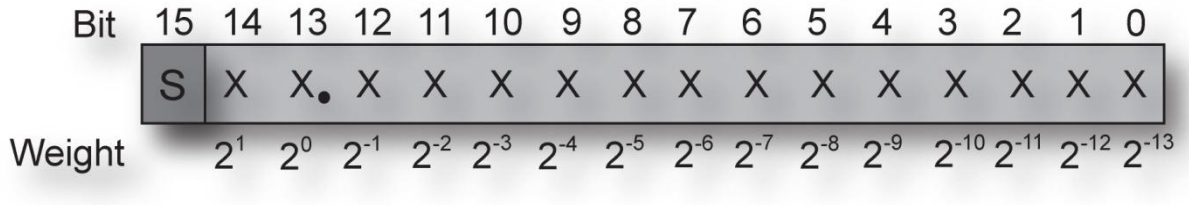

Figure 6: 16-bit Signed Fractional Fixed Point (Q13 Representation)

Fixed Point Numeric Operations

Performing numeric operations in fixed-point representation requires additional programming care. The programmer must ensure that the values remain in the available numeric dynamic range while also tracking the radix point of the operands and results. Multiplication issues arise for fixed-point numbers since the result produces more bits of precision than the original operands. Ultimately, the result will need to be converted back to the size of the original operands, but care must be maintained during the conversion to maintain numeric fidelity.

When operands greater than one are multiplied together, the result is a number greater than the original operands; this can possibly present an overflow situation in a fixed-point DSP processor. However, when two fraction operands are multiplied together, the result is another fractional value with smaller magnitude than the original operands, thus preventing overflow conditions. This principle can be exploited in fixed-point multiplication by scaling the operands

into the range from -1 to 1. As long as both operands, as well as any coefficients used, are scaled equally, the multiplication operation will provide equivalent results. To scale the operands into a fractional representation, the radix point can be shifted. Moving the radix point to the left one position results in a divide by 2 operation, while moving the radix point to the right results in a multiply by 2 operation. Therefore, as long as the binary points are shifted equally, the numbers can be normalized to a fractional representation before the multiplication operation. During arithmetic operations, the goal is to allow the operands and results to span as much of the processors dynamic range as possible [Reed 02].

Since fractional multiplication results in another smaller magnitude fractional value, the least significant bits (LSBs) of the result do not significantly represent its magnitude. Therefore, these LSBs can be truncated to reduce the result back to the original word size without any significant loss of accuracy. For example, when two 16-bit Q15 numbers are multiplied together, as shown in Figure 7, the result is 32-bit Q30 number. This Q30 result has two bits of sign and 30 bits of fractional content, where the MSB is called an extended sign bit [Cofer 06]. While the accumulator can temporarily hold this 32-bit result, it must be converted to a 16-bit word before it can be stored to memory or processed further by the DSP. To convert the Q30 result to Q15 format, the result is shifted to the right by 15 bits and the lower 16 bits are stored to memory. As an alternative method for Q15 conversion, the Q30 result can be shifted to the left by 1 bit and the upper 16 bits stored to memory.

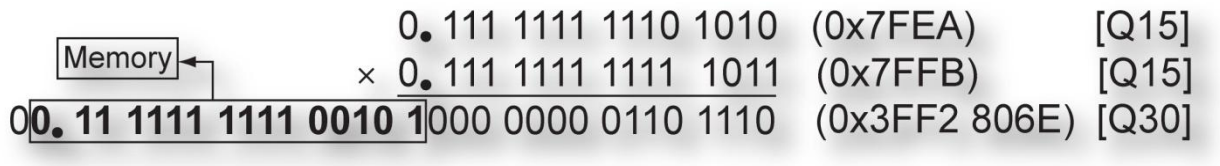

Figure 7: Fixed-point Multiplication

While scaling the operands to fractional values will always result in numeric results less than 1, preventing overflow conditions, the same cannot be said for the addition operation. When two fractional values are added together, as shown in Figure 8, their resulting value can be greater than 1. This causes an overflow condition to occur as indicated by the 1 in the sign bit. In this case, the operation result cannot simply be truncated since the LSBs are required to maintain accuracy. The solution to this issue is to require operand input values to be small enough to avoid any overflow condition. This often requires extra effort on the programmer's part to ensure the operands are small enough before the addition operation is performed. Care must be taken to track

the radix point since the addition operation requires that it be in the same location for both operands.

Figure 8: Fixed-point Addition

Floating Point

Floating point representation uses a mantissa and exponent combination to represent a number, similar to numbers written in scientific notation. The general format for a floating point number *N* is $N = mantissa * 2^{exponent}$. Both the mantissa and the exponent are fixed point numbers where the mantissa identifies the significant digits of *N.* The mantissa is often a fractional value coded in sign magnitude form, with values ranging from -1 < mantissa < 1 [Nelson 95]. The exponent value represents the number of bit positions the radix point must be shifted to the left or right to obtain the correct value. The sign bit included in the mantissa is indicated by *S* in Figure 9, which shows the IEEE single precision floating point format. This format utilizes a 32-bit word size with 24 mantissa bits, 7 exponent bits, and a sign bit. Since the MSB of the mantissa for a normalized number is known to be 1, the storage of this bit is not required. Therefore, the IEEE format suppresses this value by subtracting 127 from the exponent to add an additional bit of precision [Nelson 95]. Several other formats are also available with variable mantissa/exponent coding or increased numeric precision (utilizing longer word lengths).

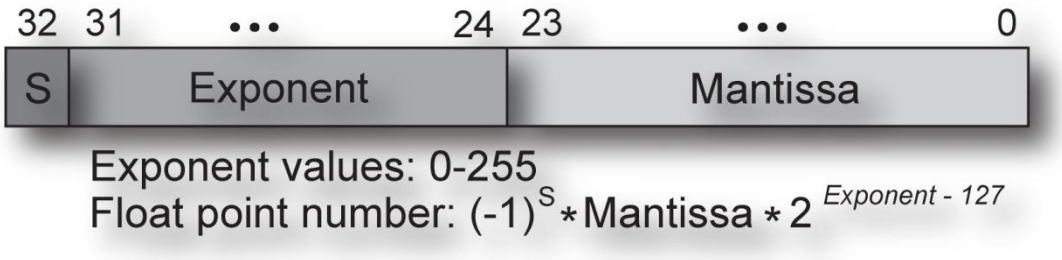

Figure 9: IEEE Floating Point Format

Despite the wide use of the fixed point format, floating point representation offers several implementation advantages in digital hardware. As mentioned earlier, floating point representation offers a much wider dynamic range over fixed point numbers of the same word size [Cofer 06]; however, floating point offers less precision. Floating point processors are often easier to program and they execute floating point algorithms with fewer cycles than equivalent fixed point

implementations [Lerner 07]. The code development is also less architecture aware, enabling increased portability among hardware platforms and simulation environments. Nevertheless, floating point processors have increased cost, silicon circuit complexity, and power consumption. Additionally, the extra dynamic range provided by floating-point representation is often not needed in many applications, thus limiting floating point use in software radio systems.

3.5 Addressing

Addressing refers to how operand locations are specified. There are various addressing modes available and DSP processors will support a subset of the modes described in this section. Some of the addressing modes are commonly found on GPPs while others have been created to efficiently execute DSP algorithms. Additionally, some addressing modes may only be available for a subset of the available processor instructions.

Common to most DSP processor architectures, the address generation unit (AGU), as shown in Figure 2, is a feature included in the CPU for speeding up arithmetic processing. The number of buses the AGU drives will vary depending on the processor architecture. The AGU operates in the background, forming addresses required for operand accesses in parallel with arithmetic calculations in the data path, thus improving performance. AGUs typically support several addressing modes. Five commonly used addressing modes that will be discussed in this section including immediate data, direct addressing, register-indirect addressing, circular address, and bit-reversed addressing. While these modes, or a subset of these modes, are common to many DSP processors, DSP processor architectures may incorporate additional specialized addressing modes; consult the vendor's programming guide for available options.

Immediate Data

Immediate data supplies the numerical value of the operand within the instruction word itself or within a data word that follows the instruction. As an example, immediate data can be used with the Analog Devices ADSP-21xx assembly language to load a register with a constant value 2345. The syntax is:

```
AX0 = 2345
```

This instruction causes the constant 2345 to be loaded into the register AX0. Note that the constant value 2345 is specified as immediate data while the destination register, AX0, is specified with register direct addressing [Lapsley 97].

Direct Addressing

Two common forms of direct addressing are available including register-direct addressing and memory-direct addressing (sometimes referred to as *absolute addressing*). In register-direct addressing, the operand is located in a register; therefore the register is specified as part of the instruction. As an example, the instruction

```
SUBF A1, A2
```

subtracts the register A1 from register A2, and stores the result back in register A1. This addressing mode is often useful when using extended precision registers on the chip with wider data widths than available in memory.

In memory-direct addressing, the operand resides in a memory location accessed by an address encoded in the instruction word or in a data word following the instruction. For instance,

```
AX0 = DM(1FA0)
```

causes the data located at memory address 1FA0 to be stored into register AX0 using the AD ADSP-21xx assembly language [Lapsley 97].

Register-Indirect Addressing

Register-indirect addressing stores data in memory and uses a register to hold the memory address where the data is located. Some processors offer special address registers for holding memory addresses while others use general purpose registers that can store both address and data values. Register-indirect addressing is important for software radio systems since it facilitates working with arrays of data, common to many DSP algorithms. This method is also useful for the instruction set efficiency since it is flexible yet requires relatively few bits in the instruction word [Lapsley 97].

Since register-indirect addressing is one of the most important addressing modes in DSP processors, there are several variations available that help increase performance including register-indirect addressing with pre- and post- increment and register-indirect addressing with indexing. The pre- and post- increment option starts with a base address in the address register but adds an increment to the address value either before (pre-increment) or after (post-increment) the data is used. For a post-increment example, the instruction

```
A1 = A1 + *R1++
```

adds the value in the memory location specified by R1 to the accumulator A1 and then increments the value in the address register R1 to reference the next memory location [Lapsley 97].

Register-indirect addressing with indexing is another option that adds an address register and a constant (or two address registers) to form an effective address to access memory. This

addressing mode is useful when the same program code is used on multiple groups of data since each time the code in run, only the index value must be changed. The pre- and post- increment options can be used in conjunction with the indexing option.

Circular Addressing

Another form of register-indirect addressing is referred to as circular (or modulo addressing). Circular addressing is important in DSP systems since DSP applications often utilize data buffers to temporarily store new data from an off-chip resource or a previous calculation until the processor is ready for further processing. These data buffers are set up in a pre-determined memory space set aside for a specific purpose, such as convolution and correlation, and typically use first-in, first-out (FIFO) protocol. The FIFO protocol reads the data values out of the buffer in the order in which they were received [Lapsley 97].

When using a FIFO, a write pointer and a read pointer are maintained. The write pointer specifies the memory location to write the next incoming data value while the read pointer specifies the memory location to read the next value from the data buffer. Every time a write or a read operation is performed on the data buffer, the write and read pointers are advanced to the next memory location. The pointers are checked on each increment to determine when they reach the end of the memory buffer, at which point they are returned to the first memory location and the process is repeated. However, the process of checking to see if the pointers have reached the end of the data buffer causes a significant performance bottleneck in the system; therefore circular addressing is used to eliminate the bottleneck. To facilitate circular (or modulo) addressing, the processor's AGU performs modulo arithmetic when calculating new address values, creating a circular memory buffer without the need to check the address pointers on every increment.

Bit-Reversed Addressing

Bit-reversed addressing is a unique addressing mode designed to accelerate the use of fast Fourier transform (FFT) algorithms. The FFT converts a time-domain signal into a frequency-domain version, useful for identifying the frequency content of wireless signals. However, after the FFT algorithm has been performed on a set of data, the resulting data is in a jumbled order. Specifically, when using the common radix-2 FFT implementation, the resulting data is in a bit-reversed order. Therefore, if the bits of a binary counter are written in reversed order from least-significant to most-significant rather than most-significant to least significant, the resulting counter order will match the jumbled order of the FFT output data. Many DSP processors support the bit-

reversed addressing mode in the AGU to increase the processing speed of FFT algorithms, where the output of one of the addresses buses is bit-reversed before being sent to the memory address bus [Lapsley 97].

3.6 Pipelining

Pipelining is a technique that breaks a sequence of operations into smaller segments in an effort to execute the segments in parallel. The number of stages that the instruction sequence is separated into is referred to as the pipeline depth. By utilizing a parallel execution, performance is improved since the overall time required to complete the instruction sequence is reduced [Lapsley 97]. However, by parallelizing an execution sequence, the programming becomes more difficult. For instance, some instructions may require additional execution time over other instructions in the pipeline, or some instruction sequences may not be available when using pipelining. The programmer must trade-off performance for development time and programming difficulty when using pipelining [Reed 02].

Pipelining uses the various memory buses available in the modified Harvard architecture to facilitate concurrent fetches for a new instruction while executing the previous instruction. For demonstration purposes, a four-stage pipeline will be used, though in practice deeper pipelines may be available depending on the DSP processor architecture. An example sequence of operations requires an instruction fetch from memory, an instruction decode operation, a read/write operation for the operands, and a MAC/ALU execute operation. In this example, it will be assumed that each stage of the sequence operates in one clock cycle. If each of these operations must be done sequentially, where the next instruction fetch cannot be completed until the current instruction execute operation is compete, the hardware resource utilization efficiency is 25% since it takes four clock cycles per instruction [Reed 02].

In a pipelined implementation, a new instruction is fetched immediately after the previous instruction fetch. During the second instruction fetch, the first instruction is being decoded. The stages have now been overlapped and the efficiency has improved to provide a completed instruction on each clock cycle following the fourth clock cycle when the pipeline is filled, as shown in Figure 10.

`

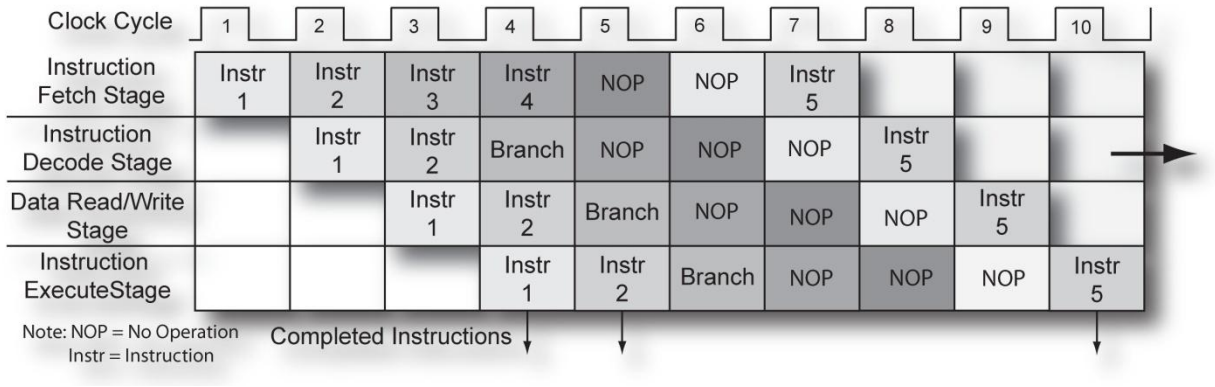

Figure 10: Four Stage DSP Processor Instruction Pipeline

In this example, once the pipeline is filled following the execute stage of Instruction 1, the efficiency of the system has improved from 25% to 100% resource utilization; an improvement by a factor of 4 over the sequential implementation. However, in practice the pipeline can be stalled for a number of reasons; therefore pipelining efficiency will not reach 100% resource utilization. As shown during the decode stage for instruction 3 in Figure 10, conditional branches can stall the pipeline since the processor must begin executing at a new address, yet the next sequential instruction has already been fetched and placed in the pipeline while the branch instruction was decoded. Branching often requires clearing the pipeline by executing no-operation (NOP) instructions until the next instruction is available after the branch as shown by Instruction 5, thereby reducing performance. Also, some instructions will take additional clock cycles to complete an operation. For instance, this scenario can happen for a long operand that requires two memory fetches or when an off-chip memory fetch is needed that takes additional clock cycles to complete. Sometimes there will also be data dependencies where a current instruction will be dependent on a previous execution result, thus stalling the pipeline. These as well as several other conditions, such as interrupts, complicate the programming but there are techniques available to deal with the issues and minimize the penalty. Nevertheless, considerable performance gains will be observed using a pipelined technique over a sequential implementation.

3.7 Multi-processing and MIMD Architecture Examples

With increasingly complex wireless application requirements, there is a need for expanding the computational ability of digital hardware devices. The traditional DSP processors have one or a few processing cores available and regularly execute at a high frequency. One method of increasing the processing power is to connect several of these DSP processors together to provide multi-

processing solution. There are several multi-processing system architectures available but a common approach is shown in Figure 11, where two dual-core DSP processors with separate level one (L1) memory caches and shared L2 caches for each CPU are utilized. L1 cache is a smaller on-chip memory providing fast memory accesses for each CPU while L2 cache is slightly slower memory cache that is typically shared between dual-core processors. The DSP processors are linked together through a shared system memory organization, as will be discussed in Chapter 7. This approach requires applications to be partitioned into separate execution units that can run on the different CPUs in parallel. Optimizing the inter-communication among processors is important for providing an effective solution since the processors must idle as they wait for data to arrive across the bus from another CPU.

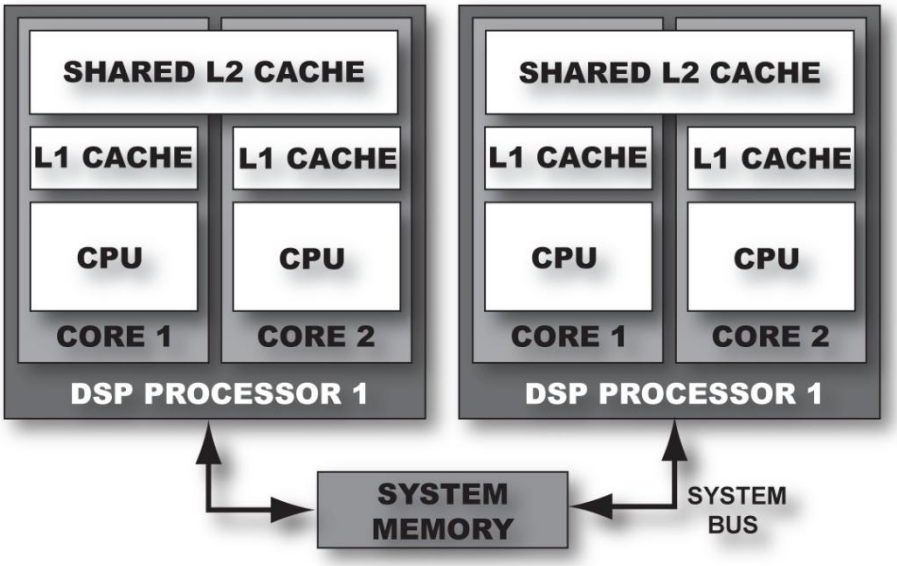

Figure 11: Multi-core processing with two dual-core DSP processors

The trend in multi-processing is moving from a multiple chip organization to creating high performance DSPs that utilize multiple processing cores on the same chip; known as the MIMD approach. On-chip multi-core improves performance by reducing the latency associated with transferring data among processing cores. One example of this trend is the picoArray PC102 from picoChip. The picoArray is a massively parallel, multiple instruction multiple data (MIMD) architecture designed for demanding signal processing applications [Duller 03]. Rather than having one powerful processing core, the picoArray utilizes hundreds of smaller Harvard architecture processing units that can be programmed individually and linked together to implement an software radio system. Each processing unit is called an Array Element (AE) and includes both instruction and operand memory connected to a 16-bit processor, shown in Figure 12. The AE's

come in different forms, offering standard processors optimized signal processing functions, processors optimized for MAC operations, processors optimized for memory accesses, and processors optimized for control operations. The standard and MAC AEs have a small amount of on-chip memory while the memory AEs have a larger amount of on-chip memory. Additionally, several functional acceleration units (FAUs) and dedicated hardware blocks (e.g. Viterbi, FFT) are available to assist the AEs and increase system performance. The units are connected together through a 32-bit data bus network, called the picoBus. By utilizing these smaller programmable elements, the designer can approach the problem with a parallel design approach, helping to facilitate development and integration. Parallel design is possible since units in the design provide deterministic computational operations, where each part of the design will operate the same alone as it does when the units are linked together. The units can be designed to implement different DSP operations such as filtering, mixing, etc., that when linked together build a software radio system.

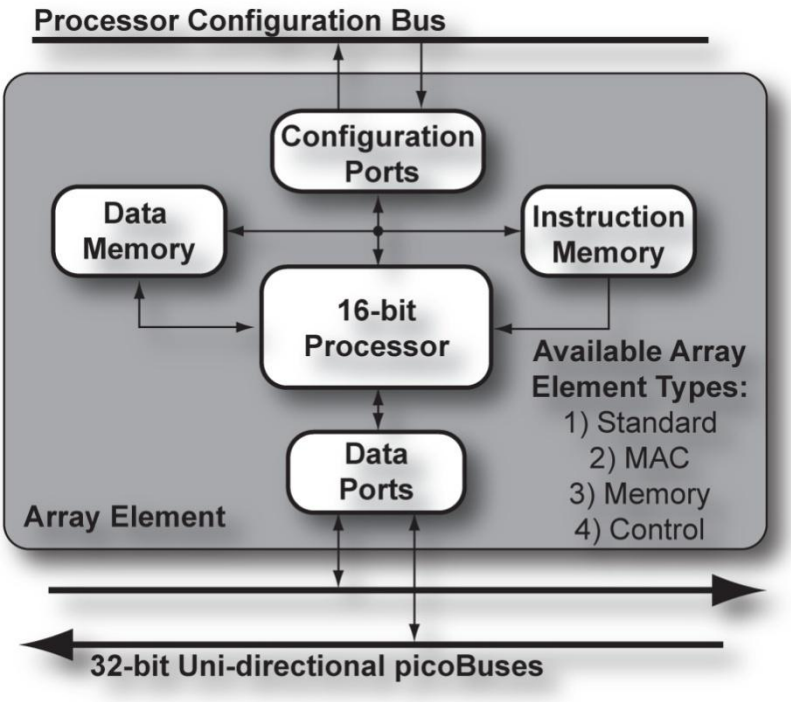

Figure 12: picoChip Array Element[2]

To obtain a high level of programmability, performance, and flexibility, a multi-processing approach is often used with a combination of FPGAs, DSPs, and ASICs. This often presents issues with co-design since integration with different technologies can be challenging, often creating an expensive and power inefficient design. Additionally, co-simulation is challenging since the processing

[2] Figure 12 is adapted from Figure 3, "Parallel Processing – the picoChip way!"

technologies are provided from different vendors with proprietary tools. The picoArray helps solve this issue by providing a powerful homogeneous processing architecture that is flexible for various wireless applications, thus minimizing the need for different technologies. It provides the parallel processing required by many software radio systems in one silicon package and one development tool set.

Multi-processing using Graphics Processing Units

Another emerging trend in software radio development is to use a graphical processing unit (GPU) to implement computationally-intensive algorithms. Traditionally, GPUs were special purpose processors designed specifically to accelerate 3-D graphics on desktop systems and required graphical processing languages, such as OpenGL, for programming. However, recently GPUs have evolved into a programmable architecture for general purpose computing, supporting high-level languages such as C and C++ [NVIDIA 09]. The GPU structure is similar to the MIMD architecture where there are hundreds of processor cores on a chip, focusing on parallel computation requirements with high data throughput. When algorithms are executed on GPUs, rather than separating the pipeline in time like DSP processors, the pipeline is separated in space. By separating the pipeline in space, the output of one stage of the pipeline is fed into another stage of the pipeline operating in a different segment of the GPU [Owens 08].

The GPU programming model is to utilize a heterogeneous architecture with a general purpose CPU connected to a GPU. In this programming model, the sequential programming is implemented on the CPU while the computationally-intensive algorithms are implemented on the GPU. This involves rewriting the algorithm to expose algorithm parallelism but high computational rates can be realized. Some of the available GPU architecture options to utilize in software radio systems are Intel's Larabee and NVIDIA's CUDA architecture. These architecture's are fairly new to the software radio field but there are some signal processing libraries available including fast-Fourier transform algorithms for NVIDIA's CUDA architecture [Owens 08].

3.8 DSP Real-time Operating Systems

Modern DSP applications have demanding requirements yet operate in constrained environments with limited memory and processing power [Kalinsky 05]. The application must respond quickly to external events, perform many operations at the same time, and prioritize processing operations [Oshana 07]. To facilitate this capability, DSP processor based systems are often managed by a real-time operating system (RTOS). An RTOS is a specialized type of operating

system that guarantees a certain processing capability within a deterministic amount of time. By utilizing an RTOS for DSP systems, the application can be built on top of system software that provides the infrastructure for interfacing with the processor hardware. The system software provides a standard set of interfaces for I/O communication and handling hardware interrupts; this increases code portability by developing software for an RTOS that is capable of running on different processor generations and architectures. This hardware abstraction is shown in Figure 13, where the application software is separated from the RTOS system software through application programmer interfaces (APIs).

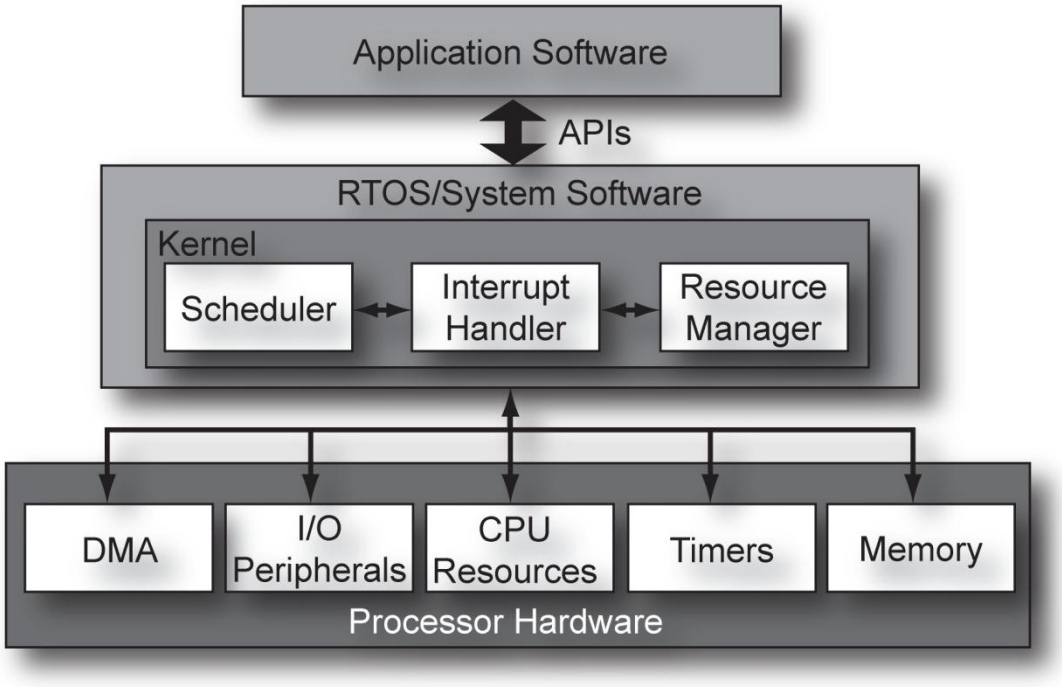

Figure 13: RTOS Hardware Abstraction

The system software in the real-time operating systems manages and arbitrates access to global processor resources such as the CPU, memory, and peripherals through the RTOS core, called the kernel. The kernel's main elements include a scheduler, an interrupt handler, and a resource manager [Oshana 07]. Performance and real-time aspects are managed by the CPU scheduler while the memory manager allocates, frees, and protects the program and data memory. Any external events that compete for kernel services are handled by the interrupt controller. I/O devices, timers, DMA controllers, and other peripherals are managed by RTOS drivers. Since the kernel services run on the same CPU as the application tasks, an inherent performance penalty is experienced when using a RTOS. However, with today's high-performance DSPs, the performance penalty can be minimized and the development benefits can outweigh the penalty cost.

Operating systems split application execution into basic computational units called tasks. The task scheduler determines which tasks should be granted access to CPU resources and in which order they should be executed. To enable real-time applications, the task scheduler needs to be priority based and be given preemptive control; therefore, DSP RTOS task schedulers offer multiple levels of interrupt priorities where the higher priority level tasks run first [Oshana 07]. When a high priority level task is ready to execute, the scheduler will immediately preempt any tasks with a lower priority level. As a result, a DSP developer can schedule various tasks in the system based on a set of rules.

Many RTOS products are available on the market for a variety of DSP processors. When selecting a RTOS for a DSP system, there are many factors involved to make the best decision. The RTOS kernel must provide efficient task execution, but system development and operational factors are equally as important. The RTOS should have a good documentation and tool support as these factors can considerably affect development time. Device support should also be evaluated to ensure the correct drivers are available for any current and future peripheral components needed. Real-time operating systems utilize limited ROM and RAM space, therefore their memory size requirements need to be gauged to guarantee the RTOS will fit in memory while leaving enough memory space available for application operation. To address the memory issue, several RTOSes provide a micro kernel, comprised of only essential services for scheduling, synchronization, interrupt handling, and multitasking [Oshana 07].

Real-time operating systems can be provided by the chip manufacturer or be developed by third-party providers. The Texas Instruments DSP/RTOS™ is an example of a RTOS developed by the chip manufacturer, offering a scalable real-time multi-taking kernel designed for the TMS320C6000™, TMS320C5000™, and TMS320C28x™ DSP-enabled platforms [TI 08b]. An example of the TMS320C5000™ platform will be discussed in Section 6.4 Case Study: TMS320C55x™ Series DSPs. Combined with the standard RTOS elements and additional features, TI's DSP/BIOS also provides a Graphical Kernel Object Viewer and Real-Time Analysis tools to help facilitate application debugging and optimization. The RTXC Quadros is another available RTOS, designed by a third-party company, for use on many processors including the Analog Devices Blackfin® series. The RTXC framework provides a scalable solution, allowing the memory footprint to fit on various memory budgets [RTXC 07]. There are many RTOSs available that will run on a variety of processor so consult the DSP processor manufacturer's website for supported real-time operating systems.

3.9 Benchmarking

Benchmarking allows for an evaluation of DSP processor performance, but making a fair comparison between processors is difficult with many factors involved. Before making final decisions on the DSP processor to use in a software radio design, system designers need a method for making useful performance measurements and evaluating the processor architecture capabilities. These measurements include the time required to complete a task along with the memory and power usage associated with that task. However, the measurements are only useful to the engineer if they can be related to the requirements of the desired application [BDTI 02] and they leave out many factors in the overall system. Factors such as power consumption, word size and memory speed all are variables in the performance of the software radio. Additionally, performance measurements depend on code optimization and compiler efficiency, which can vary widely based on the software code running on the DSP processor.

DSP processor vendors have traditionally used very simple metrics for describing performance in their devices. The MIPS (million instructions per second) metric is a widely used unit but is misleading as different processors can perform different amounts of work per instruction. Compared to traditional GPPs, this is especially true because DSP processors have a specialized instruction set to increase efficiency and perform concurrent operations. For instance, an Intel based GPP could require multiple instructions to perform a MAC operation while most DSP processors require only a single clock cycle. Nevertheless, DSP processors typically have a slower clock rate and newer GPP architectures are including more specialized circuitry. Therefore, without a baseline for identifying instruction set efficiency, the MIPS metric is only useful within the context of a single processor architecture [BDTI 02]. Similarly, MOPS (million operations per second) is a familiar performance metric but is ambiguous since the definitions for what is a considered an operation and the number of operations to accomplish a task vary among processors.

Due to the widespread use of the MAC operation in DSP algorithms, it is also often used as a performance metric by vendors. Most DSP processors can perform a MAC operation in a single clock cycle, helping facilitate efficient algorithmic implementations of FFTs, FIR filters, etc. However, these algorithms also involve several other operations beside the multiply-accumulate, so the MAC operation is not a reliable predictor of performance [BDTI 02]. The specialized instruction sets may also offer the ability to perform several operations in parallel with the MAC, which is disregarded by the MAC operations performance metric.

Despite the downfalls, these simple performance metrics do provide some basic insight into the capabilities of the DSP processors; yet they neglect secondary performance issues such as

memory usage and power consumption. These secondary performance issues are often application specific but can significantly limit a system. Applications with high memory requirements may require slower off-chip memory; thereby limiting the data flow to the DSP core and decreasing processor performance. Additionally, in a mobile situation, processors are unusable if they exceed their available battery capacity. The power consumption varies with different instructions and data values. These two parameters are important factors to consider when benchmarking DSP processors.

To provide a more complete performance evaluation, DSP processors can be benchmarked based on full applications. Application benchmarks help developers understand more about memory usage and energy consumption requirements, but there are four main problems associated with this method. To obtain the best implementation, the code usually has to use vendor supplied libraries and be developed in assembly language. However, this form of benchmarking can be a time intensive process where the benchmark application must be developed to run on each candidate processor. Additionally, the performance is dependent on how optimal the developer is able to code the software. To alleviate some of the development time, code portability is exploited where high-level language implementations are used that can be compiled for each architecture. However, adding portability is time consuming. This method results in benchmarking the compiler as well as the processor, therefore poor compiler efficiency can result in skewed processor benchmarks.

As a compromise between the low-level performance metrics and the high-level applications benchmarks, there are companies that provide third-party evaluations of signal processing technology, utilizing methodologies such as algorithm kernel benchmarking[3]. Algorithm kernels are building blocks of signal processing systems and include functions such as FFTs, filters, and vector additions [BDTI 02]. By utilizing these individual algorithm kernels as a benchmark, they can be focused on the requirements of the desired system rather than an unrelated application. Additionally, these smaller blocks provide for quicker implementations that can be optimized for a specific processor platform. These blocks are optimized for execution time and measures time as well as the memory usage and energy consumption associated with transforming an input data set into an output data set. A control benchmark is also evaluated that executes a sequence of operations, where conditional branches and subroutine calls are made.

While there is no straightforward way of evaluating DSP processor performance and the benchmarking methods may not completely reflect its performance capabilities; they serve as a

[3] A good source for third-party benchmarks is Berkley Design Technology, Inc. www.bdti.com

baseline for making design decisions. When selecting a DSP processor, it is important to consider all the benchmarking factors involved such as MIPS, clock speed, instruction set efficiency, and execution time. Additionally, the register size can have a significant impact on the performance, as high precision processors essentially perform more work which requires more power, memory, and PCB resources. For instance, it may be a better choice to choose a 16-bit DSP processor that uses less memory rather than its 24-bit equivalent if 16 bits provides enough precision for the application [BDTI 02]. Tool availability and the evolutionary development path of the vendor are also important issues to consider.

Architectural Comparison Example

Here a comparison is offered as an example of how processing architectures can be compared. As discussed in Section 3.7, the picoArray provides an innovative solution to multi-core processing, but other companies are also developing other architectural approaches. Tilera is one such company providing another MIMD parallel processor architecture called the *Tile64*, containing 64 independent processor cores connected in a mesh network. The Tile64 and the picoArray have many architectural similarities; therefore it is particularly interesting to compare these two multi-core processor architectures in terms of cost, performance and programmability. The Tile64 employs a heterogeneous array of 64 independent processors, each running at 866 MHz, while the picoArray's larger heterogeneous array of slower processors are running at 120 MHz. As mentioned previously, the picoArray is primarily focused on DSP applications; therefore it contains dedicated acceleration units for communications algorithms. Tilera has designed the Tile64 as a more general-purpose architecture, thus it does not contain any dedicated hardware blocks besides those required to drive the I/O standards. BDTI benchmarks have shown somewhat similar performance despite their architectural differences, yet different trade-offs are found in terms of cost and programmability [BDTI 08].

Both the picoArray PC102 and the Tile64 multi-core processors were evaluated based on performance, cost, and programmability using the BDTI Communications benchmark [BDTI 08]. The communications benchmark is based on standard baseband orthogonal frequency division multiplexing (OFDM) processing found in many current and emerging wireless applications. As a reference, the communications benchmark was also evaluated on the high performance Texas Instruments TMS320C6000 DSP processor and a Xilinx Virtex-4 FX140 FPGA. On the Tile64, Tilera implements a single OFDM channel using four processing cores; and since the processing cores do not share global resources, the design is scaled linearly for a total of 60 processor cores

implementing fifteen OFDM channels. The remaining four processing cores were used for I/O and buffering [BDTI 08].

While Tilera's approach to the benchmark implementation uses a linearly scaled design, picoChip approaches the benchmark implementation using a different strategy. To maximize the picoArray's heterogeneous processing resources, picoChip's communications benchmark implementation uses three different designs. One design primarily uses the hardware co-processors and accelerators to implement OFDM channels; the second implements the OFDM channels using only the AEs, and the third uses a combination of AEs and hardware accelerators. This design was able to implement fourteen OFDM channels on the picoArray, fairly close to the performance obtained on the Tile64, but with superior cost per channel. Since the Tile64 has a unit price greater than the picoArray and implements almost the same number of channels implemented, the picoArray provides a more economical solution in terms of strict hardware costs. Nevertheless, the design time associated with both implementations cannot be ignored. Tilera's implementation scaled linearly from the original design, while picoChip's design required three separate implementations, thus increasing development costs [BDTI 08]. After considering the engineering costs of developing a system on either platform, tradeoffs in time-to-market versus hardware costs can be evaluated when designing a system on either architecture.

3.10 General Purpose Processors compared to DSP Processors

While microprocessor-based software radio systems have historically taken advantage of DSP processors to do the highly computational processing, general-purpose processor (GPP) based designs are gaining strength in many systems. In general, GPPs are not designed to support software radio systems, but their high clock-rates, parallel instruction capabilities, and large on-chip memories allow them to handle substantial digital signal processing applications. By utilizing a GPP in a software radio design, many benefits are apparent, including common development tools, reduced non-recurring engineering (NRE) costs, and familiar architectures. Also, general purpose processors focus on high backwards compatibility. Therefore, even with the significant initial development costs for GPP implementations, the NRE costs drop considerably as code developed for one system can be easily ported to newer generation processor architectures. Despite these added benefits of utilizing GPPs, there is design and implementation issues associated with using a GPP for DSP applications that should be considered and addressed early in the design cycle.

When considering comparisons of DSP processors to general-purpose processors, two variants of these classes are available that includes low-end and high-performance versions. Low-end GPPs are typically cost-sensitive microcontrollers designed for embedded applications such as ARM's ARM7 architecture. High-performance GPPs are central processing units (CPUs) designed for personal computers and network servers such as Intel's Pentium® family and Freescale's PowerPC series. Low-end DSPs have a highly specialized architecture that efficiently executes specific signal processing tasks; enabling low cost, reasonably energy efficient signal processing. Conversely, high-performance DSPs offer considerable processing power for applications such as military or communications infrastructure, but at a high cost and high power consumption.

As processor architectures have developed over successive generations, many of the architectural differences unique to DSP processors or GPPs have converged and can now be found on both types of devices. For the high-performance classes, the parallel processing architectures described for DSP processors are now commonly used on general-purpose processors. For instance, the MPC7448 PowerPC™ microprocessor, available from Freescale Semiconductor, is a high-performance superscalar architecture with an SIMD multimedia unit. It has a seven-stage pipeline, four parallel processing units (named Integer Units), and single cycle execution for most instructions [Freescale 07]. The low-end DSP and GPP architectures are also similar to each other in that they typically execute a single instruction per clock cycle, have limited on-chip memory, and operate at modest clock rates [Hori 07].

Nevertheless, there are also differences identified when considering GPPs versus DSP processors for the digital hardware choice. Low-end DSP processors use compound, multi-operation instructions customized for signal processing operations, where as low-end GPPs use single operation RISC (Reduced Instruction Set Computing) instructions. This means several RISC instructions may be needed to perform the same work as a single DSP processor instruction, thereby reducing performance. In the high-performance market, general-purpose processors typically utilize the superscalar architecture over the VLIW option, where parallel instruction scheduling is done during run-time rather than during code development and compile time. As mentioned previously, this can lead to issues for real-time operation since the execution times may vary during successive iterations. High-performance GPPs also generally use floating point representation rather than the fixed point format found in most DSP processors. As discussed previously, this increases power consumption over fixed point options but provides increased numeric fidelity. These general characteristics for each processor combination are summarized in Table 1 as broad generalizations and are not specific to any processors.

Table 1: DSP Processor vs. General-purpose Processor Characteristics

Low-End Processors	High-Performance Processors
DSPs	**DSPs**
-Compound instructions	-VLIW
-Fixed point representation	-Fixed point representation
-Single instruction/clock cycle	-Up to 8 instructions/clock cycle
-Separate data and instruction buses	-Separate Data and Instruction Buses
GPPs	**GPPs**
-RISC Instructions	-Superscalar
-Fixed point representation	-Floating point representation
-Single instruction/clock cycle	-Up to 4 instructions/clock cycle
-Unified data and instruction bus	-Separate data and instruction buses

The architecture of the on-chip memory subsystem plays an important role in performance and varies among processors. As discussed, the Harvard architecture is the most effective architecture for DSP applications and as such, higher performance GPPs and all DSP processors exploit this design. However, lower performance GPPs, such as Arm's ARM7TDMI, often make use of the von Neumann architecture where program and data memory are combined. The low-end systems also do not have dedicated address generation units, so the processor must spend instruction cycles incrementing the address pointer. These factors limit performance since it will take several clock cycles to perform a single operation.

In addition to the basic GPP processing architectures adopting DSP features, such as the parallel execution units and separate memory buses; GPP vendors are also incorporating DSP instruction set extensions to their devices. For instance architectural modifications to the ARM9TDMI™ RISC core include support for enhanced signal processing instructions with minimal impact on the hardware overhead [Francis 01]. These DSP-enhanced extensions offer options for single cycle MACs, saturation arithmetic, and enhanced addressing modes, similar to the features found on a DSP processor. While these extensions improve signal processing performance, they still only build on top of the GPP instruction set and do not provide a customized DSP architecture.

DSP applications are real-time applications by nature. Therefore, during development, programmers need the ability to predict the execution time of a segment of code to ensure real-time operation. DSP processors as a group have excellent tools for predicting execution time and

developing signal processing systems. While GPPs also have mature development tools, they typically lack in signal processing features. However, as GPPs are used in more DSP applications and vendors strive to have a competitive advantage, the vendor-supplied signal processing tool support for general-purpose processors is improving [Hori 07].

In many applications, GPPs are capable of maintaining real-time processing tasks, but the ability often comes at higher operating costs than DSP processor implementations. Modern high-performance GPPs are mostly designed for speed, utilizing high clock-rates and parallel instruction capabilities. With the exception of GPPs for laptop computers, they are not designed for energy efficiency, which is quite often a significant constraint in software radio systems. Users want mobile radios, and as such, need to incorporate battery power supplies. High performance GPPs can have a large strain on the battery power available and limit the operation time of the system.

Overall, GPPs can provide a viable and cost effective digital hardware solution. However, when evaluating whether to use a GPP or a DSP in a communication system, the designer must take into account all the benefits and issues associated with each device. While GPPs may be able to provide a more cost effective solution in terms of development costs, the excess overhead and power associated with a GPP implementation may limit the capabilities of the system. GPP-based systems can often be better utilized in stationary systems where battery power is not a factor. In a DSP-based solution, more efficient algorithm computation and power efficiency may be available but at the cost of increased development time and reduced upgradability, thereby adding to NRE costs. Therefore, when choosing between a GPP and a DSP, careful attention must be paid to the overall system requirements.

Chapter 4

Field Programmable Gate Arrays

4.1 Introduction to Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) are digital logic processing devices and were introduced in the mid-1980s as alternative to the nearly obsolete gate array version of an ASIC. Their original market focus was for prototyping digital circuits and to achieve first-to-market products where they would be replaced by an ASIC circuit at the earliest opportunity [Ziedman 06]. They achieve this capability with a reconfigurable structure than can be programmed in-house rather than waiting several weeks to fabricate a fixed ASIC circuit. Since the architecture is similar to an ASIC, it provides a parallel-processing structure capable of handling many of today's demanding wireless standards. With advances in speed, logic density, power consumption reduction, and price, they are now being used in products with no intention of being replaced by an equivalent ASIC [Ziedman 06].

There are two major FPGA vendors, Xilinx and Altera, and several smaller vendors, each with varying FPGA architectures based on the same fundamental principles. The structure is pre-fabricated and optimized for multi-level circuits but programmable based on configuration circuitry, thereby adding flexibility to the radio system. The configuration points are implemented using either static random access memory (SRAM) cells, erasable programmable read-only memory (EPROM), or flash based electronically erasable read-only memory (EEPROM). SRAM cells are widely used as the configuration technology but are volatile; therefore the FPGA device must be reprogrammed during each power-up, either automatically from off-chip ROM or downloaded from an external processor [Hauck 98]. Flash based EEPROM configuration technology is also gaining in popularity since these devices retain their configuration when powered off and operate with lower power requirements [Ziedman 06].

4.2 Operation of SRAM Based FPGA Cell

In SRAM based FPGAs, memory cells are distributed throughout the device that provide the configuration circuitry. As shown in Figure 14, a SRAM memory cell consists of a pair of cross-coupled inverters that will sustain a programmed value. An n-transistor (X) is used for writing or reading a value from the inverters. The size ratio between transistor (X) and the upper inverter (Y)

is set larger to allow the value sent through (X) to overpower the inverter. Therefore, if the gate to the (X) transistor is set to 1, the switch is closed and the value assigned to the data input will then be passed to the cross-coupled inverters, where it will be retained until the next data bit is passed. A 0 value closes the (X) transistor and the value cannot pass; this is used for read-back feature to obtain the state of the SRAM memory cell. The control of the FPGA is handled by the Q and Q' outputs. A simple utilization of this SRAM memory cell is to connect the Q output to the gate of an n-transistor; therefore, similar to the (X) transistor properties, a Q value of 1 will close the transistor, allowing the value to pass between source and drain. A 0 value will open the switch, isolating the connection between source and drain. These SRAM cells are then grouped together to form lookup tables (LUTs) that create basic combinatorial logic functions inside the FPGA. The LUTs utilize 2^N programming bits connected to a multiplexer to form an N-input combinatorial Boolean function [Hauck 98].

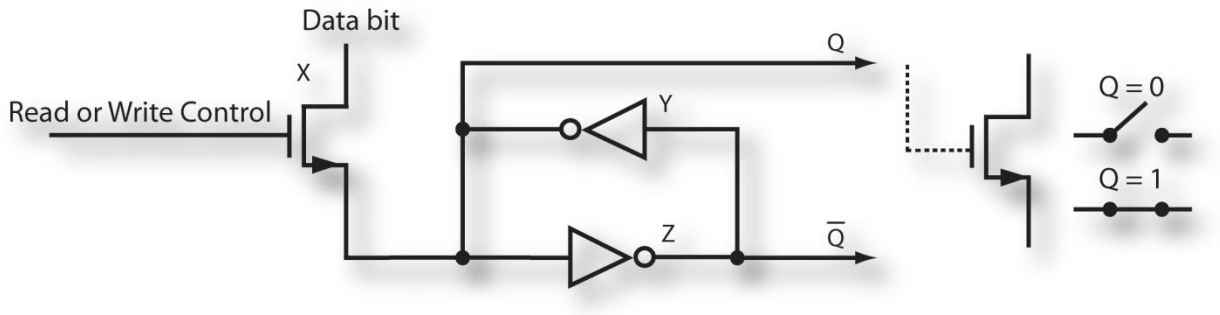

Figure 14: SRAM FPGA Cell

4.3 Applications for FPGAs in Software Radios

FPGAs offer high-speed parallel processing in a compact footprint while providing flexibility and programmability in the final product [Rudra 04]. Their functionality in software radios has expanded as the architectures have followed Moore's Law, exploiting technology developments in terms of speed, size, and capability. This high speed processing capability is often utilized in the physical layer implementation of wireless communications systems. As new wireless networks are developed, FPGAs are playing a more prominent role in the digital hardware choices, offering a scalable high-performance solution. This is especially true in software defined multi-mode base station systems where both 3G and 4G standards need to be supported [Santarini 08]. To support these standards, some typical digital signal processing applications that are implemented in FPGAs include [Dick 02]:

- Digital Down Conversion and Channelization
- Adaptive Equalization
- Error Correction Coding and Decoding
- Multi-user detection and rake receivers
- FFT based systems such as OFDM modulators and demodulators
- Fractional Rate Change Filters
- All digital timing recovery circuits, carrier recovery loops, and frequency locked loops
- Algorithms with high sample rates and non-conventional word length

Historically, FPGAs have been used for glue logic, providing interfaces between different off-the-self peripheral components wired together on a printed circuit board (PCB). Along with providing computational resources, they still maintain this functionality in most systems. The interfaces utilize most of the available I/O pins for on-off chip communications, which are connected to other peripheral components such as ADCs, DACs, external DSP processors, etc. To best utilize the peripherals, concurrent control and data bus communication needs to be provided to each device. FPGAs provide a superior solution for this communication since the parallelism of the architecture can be exploited to drive each bus concurrently (within the limitations of the I/O buses). Since the peripheral components are all centrally connected to the FPGA, data communication among the components can then be implemented within the FPGA fabric.

FPGAs also provide a mechanism for prototyping since a full circuit can be implemented on the FPGA, yet changes are possible as problems are found and enhancements need to be made. While FPGA prototyping code development takes a significant amount of time; once it is complete, it often provides faster circuit evaluation than software simulation. Additionally, the HDL code developed for the FPGA can used to create an ASIC device.

4.4 FPGA Architectures

FPGA architectures vary by vendor but in general are a variation of that shown in Figure 15 [Ziedman 06]. The FPGA hardware architecture is composed a core set of processing resources that define the FPGA and allow for its highly reconfigurable structure. The core resources of an FPGA, as discussed in the following sections, include logic blocks, memory units, I/O buffer cells, multiplexer buffers, clock management circuitry, and dedicated DSP resources. All of these processing resources are organized into a matrix of reconfigurable routing. The arrangement of the routing interconnections is highly programmable and defines the functionality of the device. FPGAs also

include other dedicated hardware resources that help offload processing from the core resources such as embedded microprocessors or transceivers.

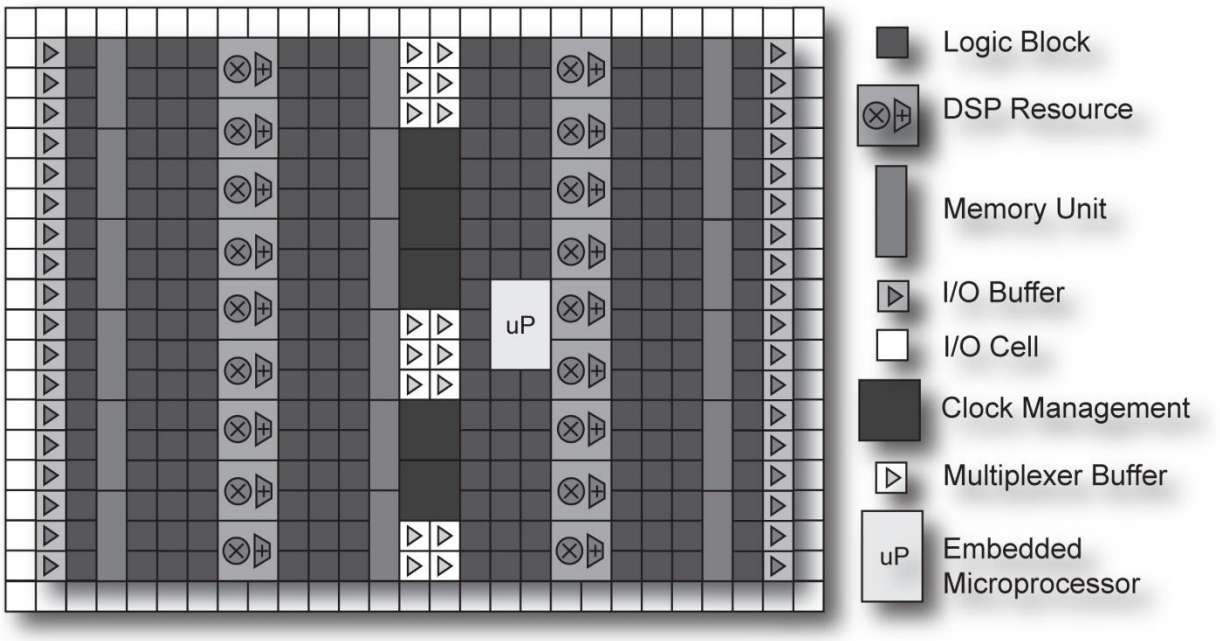

Figure 15: Generic FPGA Architecture for DSP Applications

Logic Blocks: In Xilinx FPGAs, interconnections are made between logic units called configurable logic blocks (CLBs) which consist of look-up tables (LUTs), flip-flop registers, and multiplexers. LUTs are constant delay devices that store combinational logic results, thus reducing the combinational logic requirements of the device. Large processing gains are achievable by using LUTs since the device does not have to do an expensive computation to determine the value of a combinational function. For many years, 4-input LUTs were the industry standard. However, in the latest generations of Xilinx devices, Virtex-5 and Virtex-6 FPGAs have introduced 6-input LUTs. The additional inputs provide added flexibility for implementing more complex functions with a reduced number of logic levels [Cosoroaba 07]. The 6-input LUT is attached to multiplexers and registers to for a LUT-register pair, shown in Figure 16 below, where four LUT-register pairs then connect to form an FPGA slice. Finally, CLBs are made up of two or more slices. The multiplexers are used to implement wide functions, up to 28 inputs utilizing LUT input signals and the cascaded input signals. A four level carry chain is available that, among other functions, helps support arithmetic operations. Outputs from the Virtex-5 slice are presented as registered outputs on signals the register out signals, combinational outputs on the combinational out signals, or

multiplexed outputs on the multiplexer out signals. By cascading CLBs together with these signals, complex logic circuits can be built for software radio applications.

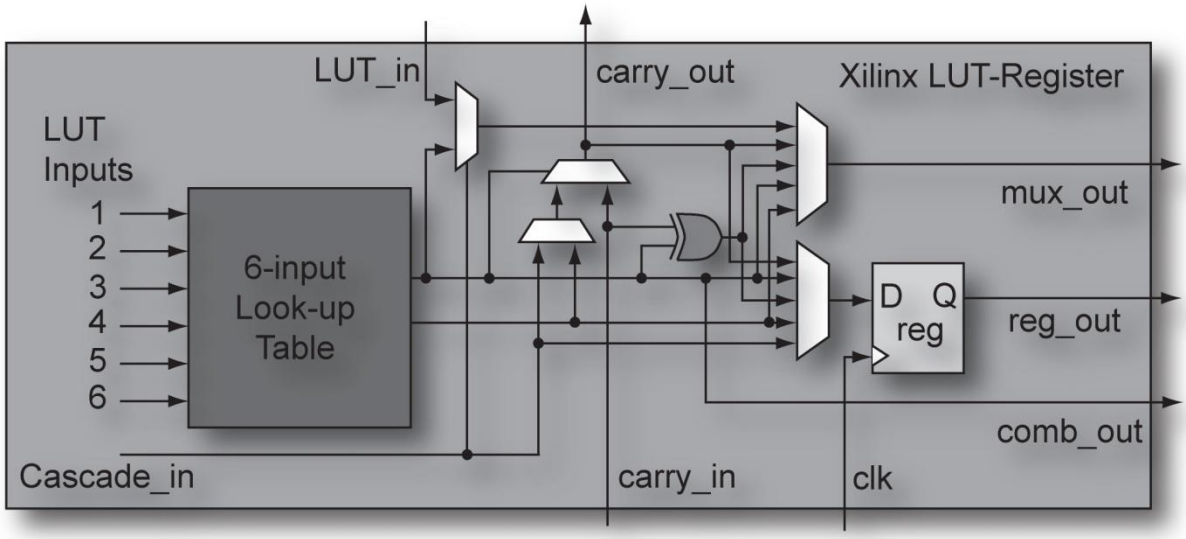

Figure 16: Virtex-5 FPGA Slice and Logic Matrix[4]

In contrast to the logic inside the Xilinx CLB, the Altera Stratix series FPGA architecture utilizes an adaptive logic module (ALM) as the basic logic unit. The ALM also uses LUTs and associated logic to implement functions but their architecture is designed differently than the Xilinx CLB logic. The ALM structure is shown in Figure 17 below; it includes combinational logic, two adders and two registers. The combinational portion is an adaptive LUT with eight inputs that can implement one 6-input function or various combinations of two independent smaller functions; for example, two 4-input functions or a 3-input and a 5-input function [Verma 07]. The two embedded adders inside the ALM allow for two 2-bit or two 3-bit additions without additional resources. The resulting operations from the LUTs and adders can be presented on the combinational output pins or passed through output registers for synchronous operation.

[4] Figure 16 is adapted from Figure 2, "Achieving Higher System Performance with the Virtex-5 Family of FPGAs"

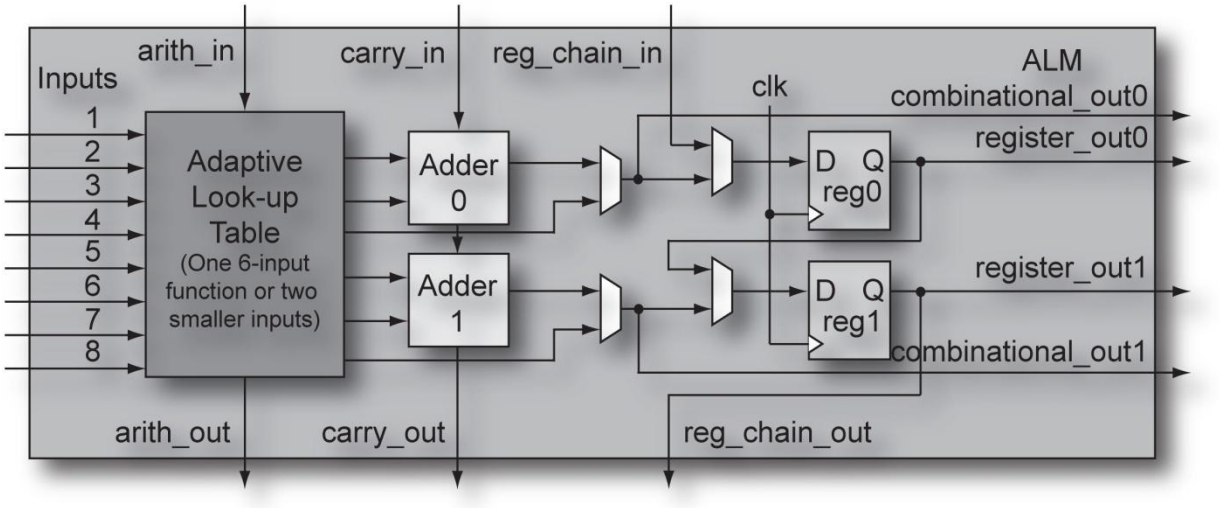

Figure 17: Altera Adaptive Logic Module[5]

The architecture inside Altera Stratix series FPGAs groups 10 ALMs together to form a logic array block (LAB), compared to the 8 LUTs/Flip-flops that form a Virtex-5 CLB. The ALMs are linked together through the shared arithmetic, carry, and register signals that are shown in Figure 17. LABs can be used for a standard logic functions or can be configured as a dual-port SRAM resource.

FPGA Interconnections: While CLBs and LABs provide the computational logic resources for the FPGA, the flexible interconnect routing enables the reconfigurable nature of the device. Signal routing inside the FPGA comes in many forms, including; neighboring logic unit connections, double length lines, long horizontal and vertical connections spanning the length of the device, and low-skew routing for clocking and global signals. Figure 18 shows how logic unit to logic unit connections are made, indicating that wires must either pass through a routing switch box or utilize direct logic block-to-logic block interconnections. Typically, the implementation software tools hide the device routing from the designer, thus simplifying the designer's task. The designer only needs to be concerned with critical timing routes where constraints need to be placed on the propagation delay of a signal through the device. Meeting critical timing routes, as discussed in Section 4.11 Design Principles and Tools for FPGA Development, is an important design concern in software defined radio systems since the system usually must meet real-time signal processing requirements.

[5] Figure 17 is adapted from Figure 8, "Stratix III FPGAs vs. Xilinx Virtex-5 Devices: Architecture and Performance Comparision"

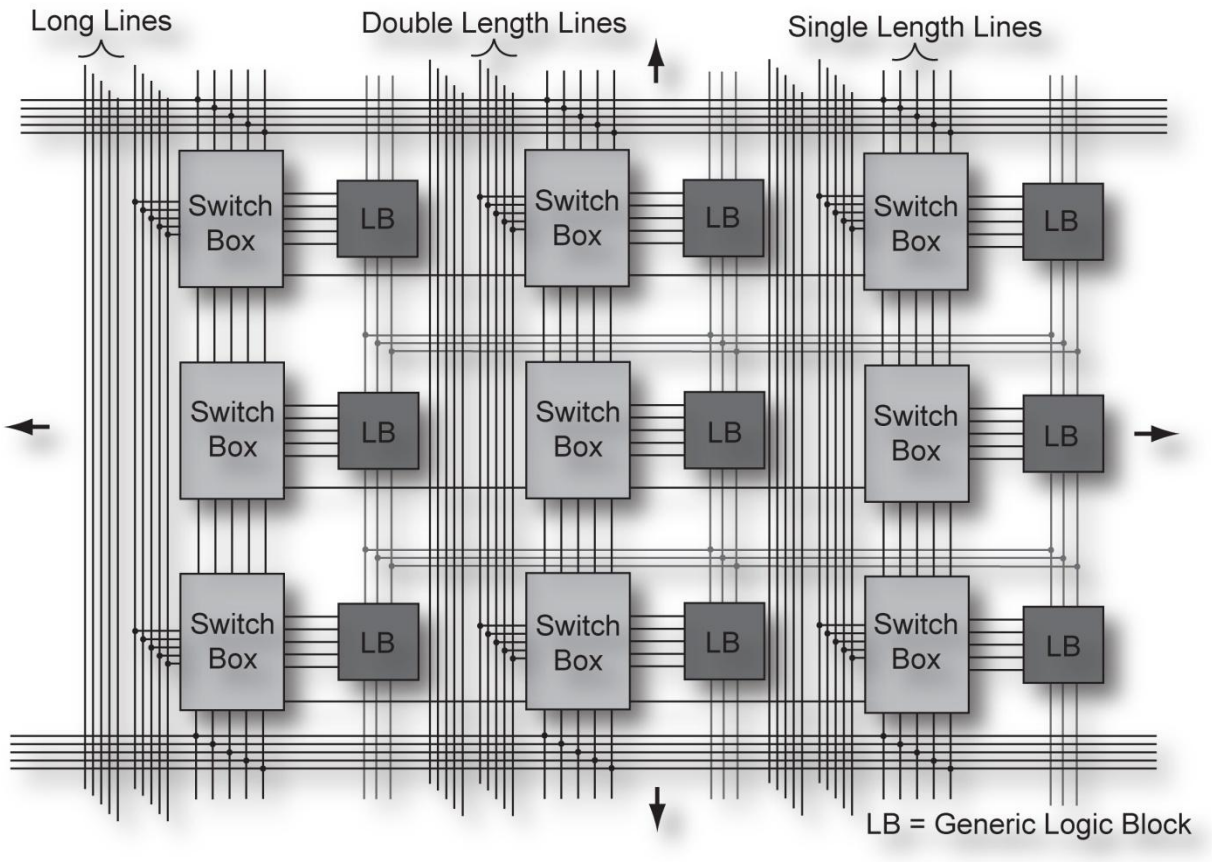

Figure 18: FPGA Re-configurable Routing Matrix

The Virtex-5 and the Stratix III FPGA families introduced a new routing arrangement using a diagonally symmetric interconnect structure. The new interconnection pattern allows more logic unit connections in fewer hops through the use of diagonal connections that are shown in Figure 18. Therefore, a logic unit can directly connect with its vertical and horizontal neighbors as well as its four diagonal neighbors; this reduces timing delay since the number of switch boxes a signal must pass through is reduced. Additionally, the symmetric structure allows the place and route tools to more efficiently route the design, resulting in higher overall performance.

FPGA Memory: Memory is an important element in software defined radio designs and can typically be implemented in two different on-chip architectures in FPGAs. Both Altera and Xilinx offer the ability to utilize dedicated hardware memory resources or to build memory from the FPGA fabric. The dedicated memory modules are called BlockRAMs in Xilinx FPGAs, whereas they are called M9k or M144k blocks in Altera FPGAs. The number of available dedicated memory modules

varies among devices but they offer large memory sizes and when combined together offer between one to two-dozen million bits of RAM. BlockRAM and MLAB resources are dual-port memory units, meaning there are separate address and data lines for both read and write operations. Each port on the memory unit is synchronous and independent, allowing concurrent read and write operations. Concurrent read/write operations allow data exchange over different clock domains. Additionally, memory units have options to control the depth vs. width configurations, utilize error correction coding, and support built in first-input first-output (FIFO) buffers.

On-chip memory can also be built from the FPGA logic fabric; named distributed memory in Xilinx devices and MLAB in Altera devices. This type of memory can only store small amounts of data but has speed and scalability advantages, offering many more data ports for increased memory bandwidth. The disadvantage to this method is that the memory consumes a portion of the FPGA logic fabric, reducing the amount of logic available for other algorithms. Therefore, distributed RAM is used for small and fast memory requirements while larger memory is typically implemented in the dedicated hardware blocks.

Input/Ouput Connections: An important element to the FPGA architecture and often the biggest bottleneck is registering data into and out of the device. Input/Output Blocks (IOBs) are used for providing a link between on-chip and off-chip resources. Their structure consists of a tri-state input buffer and a tri-state output buffer connected to flip-flops. The tri-state buffers offer logic high, logic low, and high impedance results and open collector output controls. Varying logic levels are supported, enabling communication with other off-chip resources on the board. The slew rate can often be programmed for fast or slow rise and falls times that correspond to the off-chip resource requirements. Additionally, buffers usually contain pull-up and possibly pull-down resistors for terminating signals without discrete resistors external to the chip [Ziedman 06]. IOBs are grouped into banks with each bank able to support a particular voltage reference, allowing support for up to 40 standards in the Virtex-5 family.

Clock Management: Software radio systems are implemented with synchronous circuits inside FPGA devices. Providing a clean, low-jitter clock signal is critical for these synchronous systems since the FPGA clock can introduce noise into the signals if the timing is not consistent. Clock management tiles (CMTs) on Virtex-5 FPGAs provide complete clocking solutions for high-speed clock networks; offering options for jitter filtration, phase-shift delay compensation and frequency synthesis. Jitter filtration removes phase noise from the signal, frequency synthesis creates

additional clock frequencies from a reference frequency, and phase-shift delay adds or subtracts pre-determined delays to the input clock signal. Each CMT contains two digital clock managers (DCMs) and one phase-locked-loop (PLL).

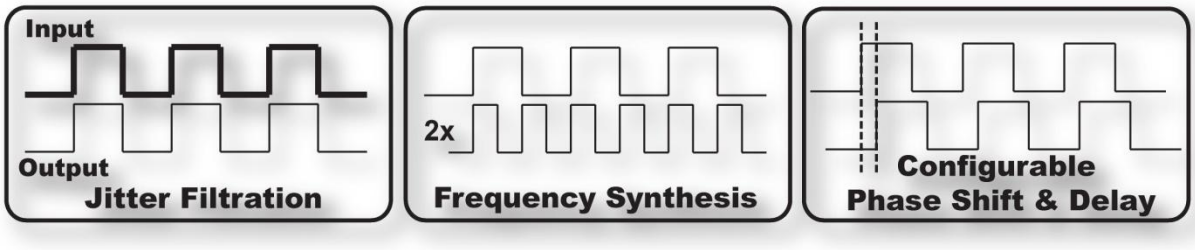

Figure 19: Clock Management Options

Digital clocks suffer from phase distortion based on the length of the distribution paths from the clocks source to the sinks, whether it is registers, memory devices, etc. If the path length varies between two sinks, a phase distortion will be introduced, possibly altering the data. DCMs can use fine-grained phase shifting to compensate for the clock distribution delay, adjusting the clock phase within fractions of a clock cycle. DCMs will also provide 90°, 180°, and 270° phase-shifted versions of the clock, as well as fractional and integer multiples of the clock rate using frequency synthesis. Both of these features are useful in software radio systems for aspects such as DAC and ADC clocking, multi-rate filtering and symbol synchronization.

4.5 FPGA Run-time Reconfiguration

The FPGA reconfigurable architecture is often discussed and utilized for modern software defined radio system, but the reconfiguration is typically done on power-up by loading new FPGA configuration bitstreams. A fairly new design methodology for FPGA design is reconfiguration during run-time. Run-time reconfiguration is often over-shadowed by Moore's Law, with designers having the attitude that if the current design does not fit on the silicon, a newer, more powerful device will be able to support the design in the future. However, this attitude is fading as more increasingly complex wireless standards and requirements are emerging in the marketplace. Many advantages for run-time reconfiguration are apparent, including increased functional density, increased performance, and reduced power dissipation.

Run-time FPGA reconfiguration is the partial or complete restructuring of the implemented architecture on the FPGA fabric while the system is operational. By restructuring the design architecture during run-time, radio systems can quickly adapt to changing wireless standards and

requirements. As FPGA-based radio systems evolve, they are including more and more features and waveform capabilities. However, increased capabilities translate into larger designs and higher power consumption. Run-time reconfiguration can help mitigate this problem by re-using available logic resources or shutting down logic when it is not in use, helping alleviate the issue of static power consumption, which is a significant factor of the power budget.

The major FPGA vendors have run-time reconfiguration abilities built into their devices. Xilinx supports partial reconfiguration, where specific regions of the device, named partial reconfiguration regions (PRR), are devoted to reconfiguration while others must remain static during operation. Partial reconfiguration has the benefit that the static regions of a design can continue to operate during reconfiguration of the PR regions. To adapt a PR region to a different capability, a partial bit stream is loaded into the Internal Configuration Access Port (ICAP). The partial bit stream targets only the PR region specified while leaving the rest of the design to continue its current operation. An important note is that static logic resources of the base design will not be placed in any PRRs, but static logic routing can pass through the PRR. The static routes inside the PRR will not be affected during reconfiguration. The figure below depicts the static regions associated with an FPGA mixed with reconfigurable regions.

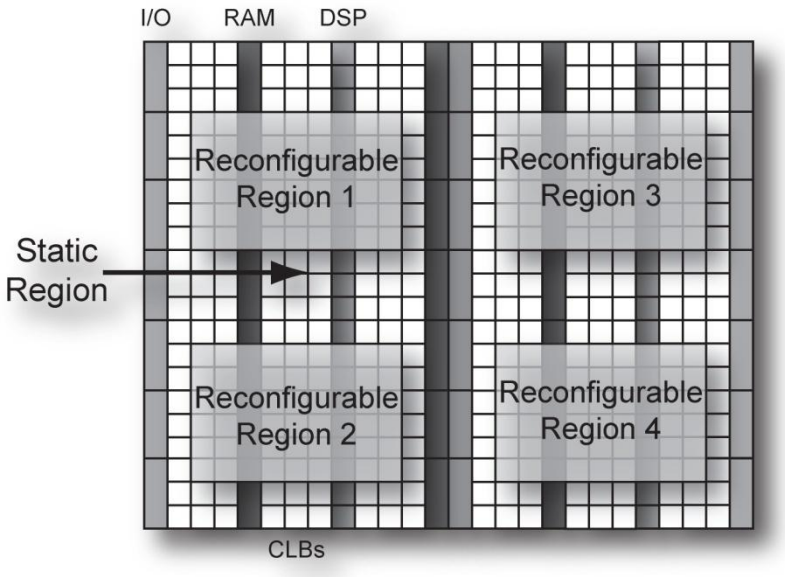

Figure 20: FPGA Reconfiguration Regions

Many methods have been proposed for monitoring and control of the reconfiguration. Often, the FPGA reconfiguration must be monitored by an external processor that determines when new partial bit streams will be loaded onto the FPGA. This requires a significant overheard to the

system if an instruction-based processor is not required for other applications. One fairly new method for reconfiguration control includes incorporating the control into the FPGA itself. By using a hardcore embedded PowerPC or soft-core MicroBlaze processor, the FPGA gains the ability to quickly and effectively modify itself [Becker 07].

FPGA partial reconfiguration has vast possibilities for software radio systems. Software radios strive to accommodate a wide variety of wireless standards, including future standards not yet conceived. While a software radio may be able accommodate many of these various standards, it is not possible, nor practical to have the silicon configured for all standards at once. By utilizing partial reconfiguration, the required hardware architecture for a particular wireless standard can be loaded when required.

4.6 Implementing DSP Functions in FPGAs

Digital Signal Processing (DSP) algorithms are the core of any modern communications system. Emerging wireless standards are rapidly accelerating the signal processing requirements required to support new algorithms. While DSP processors have tried to keep pace with increasing wireless demands, superior signal processing gains are available using FPGA technology. FPGAs provide an inherently parallel architecture, creating an advantage over instruction-based devices. This is especially true for newer communication standards, such as WiMax, where serial processing is insufficient.

Historically, DSP processors have been the traditional approach for implementing DSP algorithms. Instruction-based processors are still heavily used for digital communication systems, but as wireless standards become increasingly complex, FPGA implementations are more attractive to off-load some of the DSP processing. Algorithm performance in an FPGA compared to traditional DSP instruction based processors is usually based on a combination of factors. The most common performance improvements come from the increased data widths and the inherent parallelism. The ability to take advantage of the FPGA architecture and resources is also essential in improving performance. Using the FPGA architecture, the capability to process data on several data channels can be exploited. This provides significant performance enhancements on applications such as Time Division Multiple Access (TDMA) multiplexing, multi-channel communication protocols, and I/Q based waveforms. The data for each channel can be processed at the same time and combined at the output.

The major FPGA vendors have products specifically targeted at DSP algorithm implementation. These specialized FPGAs include various dedicated hardware resources for

improving performance, including MAC blocks and memory structures; since DSP algorithms have a high dependence on the multiply-accumulate function. Multipliers are costly in hardware and consume a significant amount of FPGA fabric resources due to high logic complexity and area usage. The dedicated DSP processing resources that implement the MAC operations help solve the issue; but the problem still remains when there is a large number of MAC operations needed in a design. To help alleviate this issue, DSP algorithms in FPGA are often implemented using distributed arithmetic.

Distribute arithmetic is a method for decomposing the MAC operation into a series of look-up table (LUTs) accesses and summations [Lo 08]. This method is based on partitioning the function into pre-computed partial terms that are stored in LUTs using 2's complement binary representation. The flexibility of the algorithm allows for bit-serial, pipelined, or bit-parallel implementations, thereby trading bandwidth for resource area [Longa 06]. However, while distributed arithmetic offers a multiplier-less DSP algorithm implementation, it comes at the expense of added memory requirements.

4.7 Embedded DSP Functionality

Both Xilinx and Altera include dedicated DSP hardware resources in their devices, each with different names and architectures. These dedicated hardware blocks implement functions in an efficient structure, using only the necessary transistors required for the function rather than using the FPGA fabric structure [Curd 07]. The Xilinx version is called a DSP Slice and utilizes a highly flexible architecture in the Virtex-5 family, capable of handling intensive DSP operations, such as the MAC function. As discussed previously, due to the inherent complexity of the multiply operation, large amounts of logic are required for implementation. By using the DSP Slice, the system designer has the ability to off-load the hardcore multiply and accumulate operations to the slice, saving significant hardware resources in the FPGA fabric and reducing power consumption.

Virtex-5 DSP slices can also be configured to implement other functions such as high-speed counters, logical units, and shift registers to save fabric resources and improve performance. The DSP48E slices include support for Single Instruction/Multiple Data (SIMD) operations and saturation arithmetic. Additionally, the DSP slices can be cascaded together through dedicated signal routes, enabling complex arithmetic and DSP filter operations. The diagram of the Xilinx DSP slice is shown in Figure 21 below. As shown, the DSP48E slice includes a 25 x 18 bit multiplier on inputs A and B, respectively. The result of the multiplier is fed into the arithmetic logic unit (ALU) that is capable of addition, subtraction, bit-wise operations, and other combinations of functions

based on the ALUMODE configuration register. Several other signal ports and configuration options are available for added flexibility and performance improvements; including floating point support [Xilinx 08]. In Xilinx's newest FPGA family, Virtex-6, the DSP48E slice has been enhanced to include a pre-adder that improves performance in densely packed designs, reducing the logic slice requirement by up to 50% [Xilinx 09].

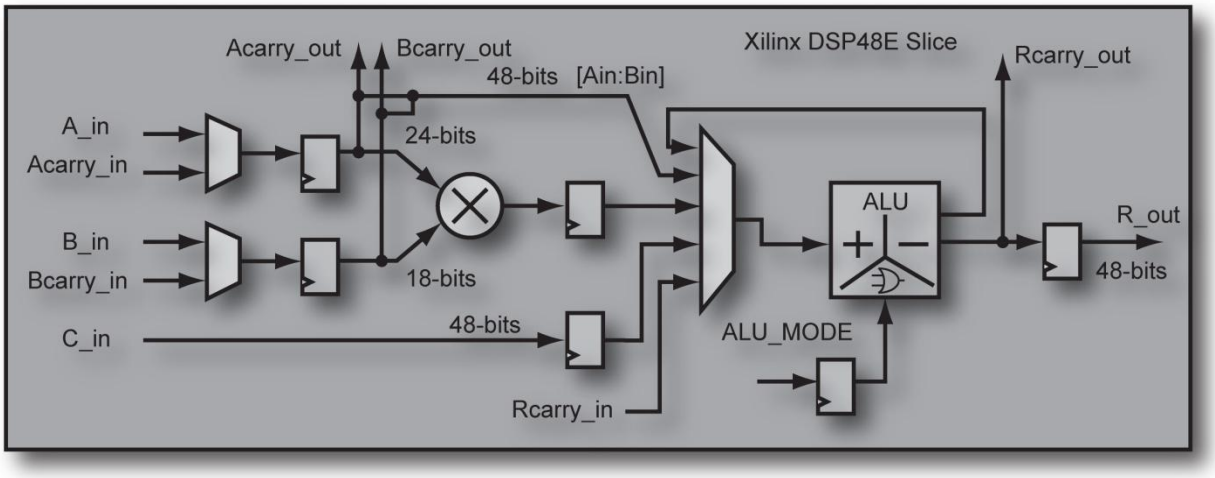

Figure 21: Xilinx DSP48E Slice[6]

The Altera version of the DSP hardware resource is named a DSP block and utilizes a different architecture than the Xilinx DSP Slice. While the Xilinx version provides a large input word width, the Altera DSP Block provides flexible input options. The architecture is designed in such a way that it can implement four concurrent 18x18 bit multiplier operations, or it can be reconfigured to support eight 9x9 operations or one 36x36 operation.

[6] Figure 21 is adapted from Figure 1-1, "Virtex-5 FPGA XtremeDSP Design Consideration"

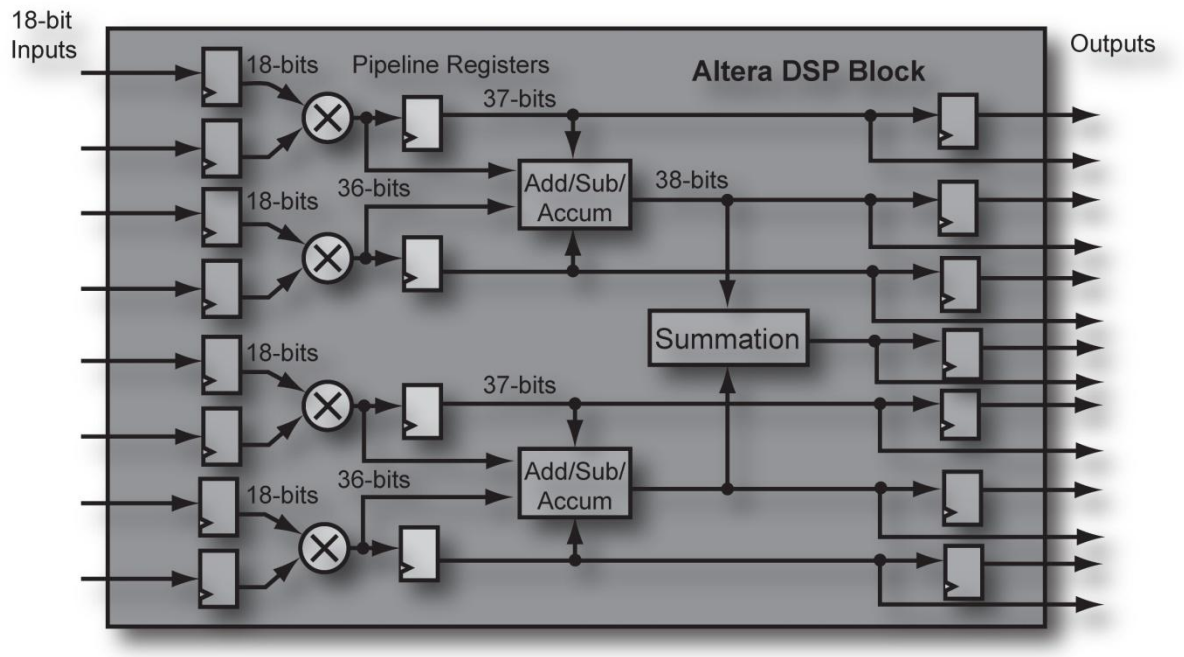

Figure 22: Altera DSP Block[7]

4.8 Exploiting Parallelism in DSP operations

The parallel architecture of FPGAs combined with the dedicated hardware resources make the devices highly efficient at implementing DSP algorithms. Some common algorithms that can be implemented on an FPGA include FIR filters, Fast Fourier Transforms, sample rate conversions, equalization and digital up/down conversion. Parallel DSP algorithm implementation can provide major performance improvements. Assuming a 256 tap FIR filter, a serial implementation would require 256 clock cycles before an output value is available. This is compared to a parallel implementation where the output value is available on the first clock cycle. The trade-off for the added performance available arises with resource usage inside the FPGA. Assuming a FIR filter has N taps, parallelizing the filter will increase the number of million multiply accumulates per second (MMAC/s) by N times, but resource utilization has also increased by a factor of N. Based on the limited number of dedicated DSP hardware resources available and the available FPGA fabric, this may or may not be worth the additional resource cost. Semi-parallel implementations are also available that may strike a better balance of performance versus resource usage. Figure 23 below illustrates this tradeoff.

[7] Figure 22 is adapted from Figure 2, "Digital Signal Processing (DSP) Blocks in Stratix Devices," Altera Corp.

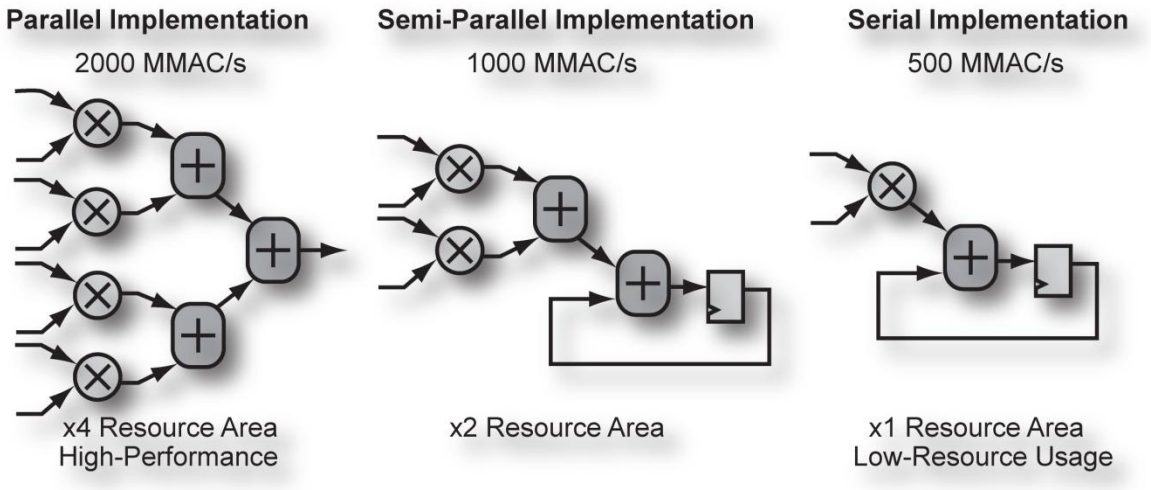

Figure 23: FPGA Speed vs. Hardware Utilization

4.9 FPGA DSP Implementation using IP Cores

DSP algorithm development is a complex and time intensive process. Typically, designers do not want to spend development time on low-level DSP operations, such as FFT blocks, DDS modules, or FIFO memory elements. Not only is there a considerable development time for these blocks, the verification time for the element is also significant. Designers would rather spend their time at a higher level, developing the overall system.

To reduce development time, software radio designers utilize DSP Intellectual Property (IP) available from the FPGA vendors or third-party sources. IP provides access to DSP functionality that has been pre-verified and often optimized for a specific architecture. Not only does using pre-verified code reduce development time, it also increases the reliability and performance of the system. Some categories of available IP cores are shown in
Table 2 below.

Table 2: DSP IP Categories [Cofer 05]

Category	Example
DSP Function	FFT, Viterbi Decoder, MAC, FIR, Discrete Cosine Transform
Math Function	Parallel Multiplier, Pipelined Divider, CORDIC
Logic Function	Accumulator, Shift Register, Comparator, Adder
Memory Function	Block Memory, Distributed Memory
Communication	AES Encryptor, Reed-Solomon Encoder, Turbo Decoder

4.10 Digital Down Converter Implementation in an FPGA

Many software radio systems utilize FPGAs early in the RF process to perform repetitive, computationally intensive tasks. Wireless systems typically sample signals at IF frequencies, which must be translated down to baseband before any data processing can be performed. Digital down converters (DDCs) are fundamental building blocks that perform the translation and subsequent channel filtering. Their introduction revolutionized the communications industry and FPGAs are an effective architecture to use for DDC implementation, partly due to the large number of multipliers required. Additionally, as the number of required DDC channels increases, FPGAs become increasingly attractive in terms of size, weight, power and cost per channel [Hosking 08]. Figure 24 illustrates the basic structure of a DDC block.

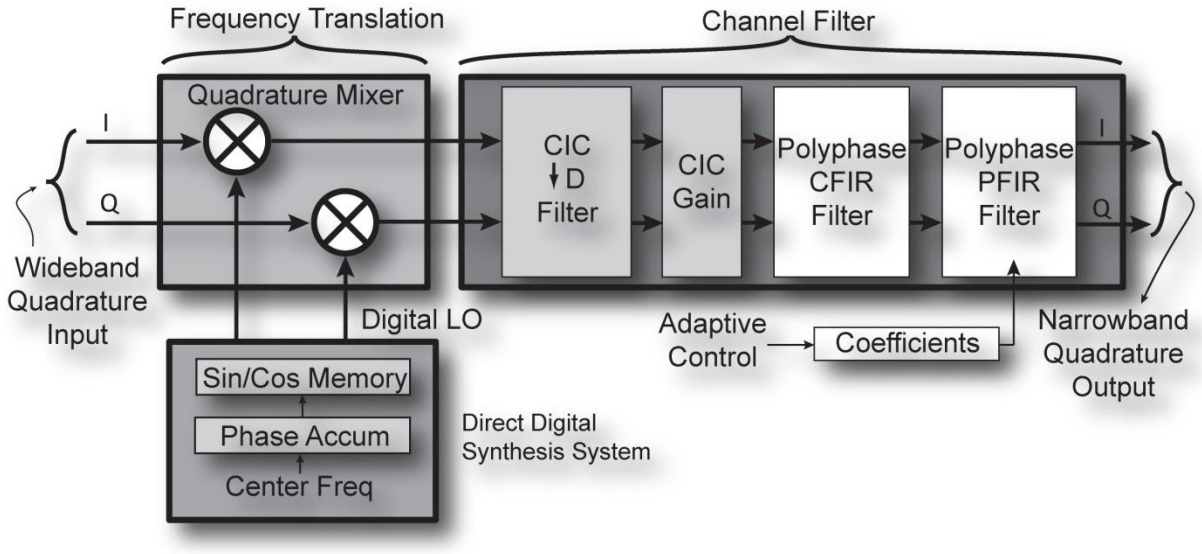

Figure 24: Digital Down Converter Block[8]

As shown in Figure 24, DDCs have three essential elements, the local oscillator, the quadrature mixer, and the filter. The local oscillator (LO) has a direct digital synthesis system (DDS) that includes a digital phase accumulator (an adder and a register) and a look-up take to generate the digital sine and cosine signals for mixing. For the system to function correctly, the phase accumulator must be clocked at the ADC sample rate to provide LO samples at the same rate

[8] Figure 24 is adapted from Figure 2, "New FPGAs Revolutionize Digital Down Converters," by Robert Hosking, used by permission granted by Allan Margulies, COO, SDR Forum

as the incoming data. The frequency of the LO is varied by adjusting the phase increment on each clock cycle. The LO is then fed into the quadrature mixer.

The quadrature mixer performs the frequency translation for the DDC block. It includes two multiplier blocks that operate on each take data sample from the ADC and the LO. As discussed in a previous section, the multiplier block could be implemented using a dedicated DSP slice to save FPGA fabric resources on implementation. If the frequency of the LO is set to the frequency of the desired signal, the result of the multipliers provides an in-phase and quadrature representation of the input signal, shifted to baseband (0 Hz). Various frequency bands can now be shifted to baseband by adjusting the frequency of the digital local oscillator. While the mixer operation has shifted the band of interest to baseband, the signal of interest still must be isolated with subsequent filtering.

The channel filter after the mixer stage accepts baseband I and Q samples. One implementation method is to use a FIR filter architecture that provides a wide range of transfer functions. To adjust the transfer function, the filters can dynamically adjust the number of taps and coefficient values, providing different passband ripples, cutoff frequencies and stop band attenuations. Since FPGAs are highly flexible, the required parameters and coefficient values can be dynamically loaded to vary the transfer function of the desired filter.

The last step in the DDC block is to decimate the signal to a rate consistent with the bandwidth reduction of the filter. The decimate stages takes one out of every N samples to be sent to subsequent signal processing stages, such as equalization, demodulation or decoding. Two important results of decimating the signal out of the DDC block is reducing the number of samples that must be passed into the subsequent processing blocks and matching the system clock rate inside the digital hardware. With a reduced number of samples, the complexity of these blocks is largely reduced.

While FIR filters are common implementation for DDC blocks, they often consume are large number of resources on the FPGA. Shown in Figure 24, the filtering operation can be combined with the subsequent decimation step using cascaded-integrated comb (CIC) filters that offer high orders of bandlimiting and decimation. CIC filters do not require multipliers, thereby greatly reducing the logic resources required for implementation; however, they distort the passband flatness. Compensation FIR (CFIR) filters can correct for the pass band flatness with subsequent polyphase FIR filters. A second programmable FIR (PFIR) filter follows with adaptive coefficients to establish channel frequency characteristics. While this may sound like reintroducing the problem

of FIR filters, the CFIR filter complexity is greatly reduced since the sample rate of the signal has been reduced by the CIC filter.

Digital down converters are typically one of the various IP cores available from FPGA vendors. Xilinx's LogicCore DDC can be configured for various levels of SFDR (spurious free dynamic range), trading off performance for FPGA hardware resources. For example, a complex DDC with 84 dB SFDR consumes approximately 1,700 slices for each channel [Hosking 08]. While this implementation may be practical for systems requiring only a few channels, the FPGA resources will not be sufficient for systems requiring tens to hundreds of DDC channels. In such cases, additional design measures must be taken for altering the algorithm architecture to reduce the number logic resources required. For instance, a channelizer followed by a multiplexed DDC stage can significantly reduce the amount of resources required. In one example, a 256 channel DDC implemented in a channelizer/multiplexer DDC architecture will consume 18,000 slices. The original DDC, requiring 1,700 slices, would require 435,200 slices for the same implementation [Hosking 08]. In this implementation, 256 separate channels are provided on the multiplexed output, therefore 256 separate data streams can be provided by using a time-division multiplexing (TDM) approach on the output, useful for multi-user systems.

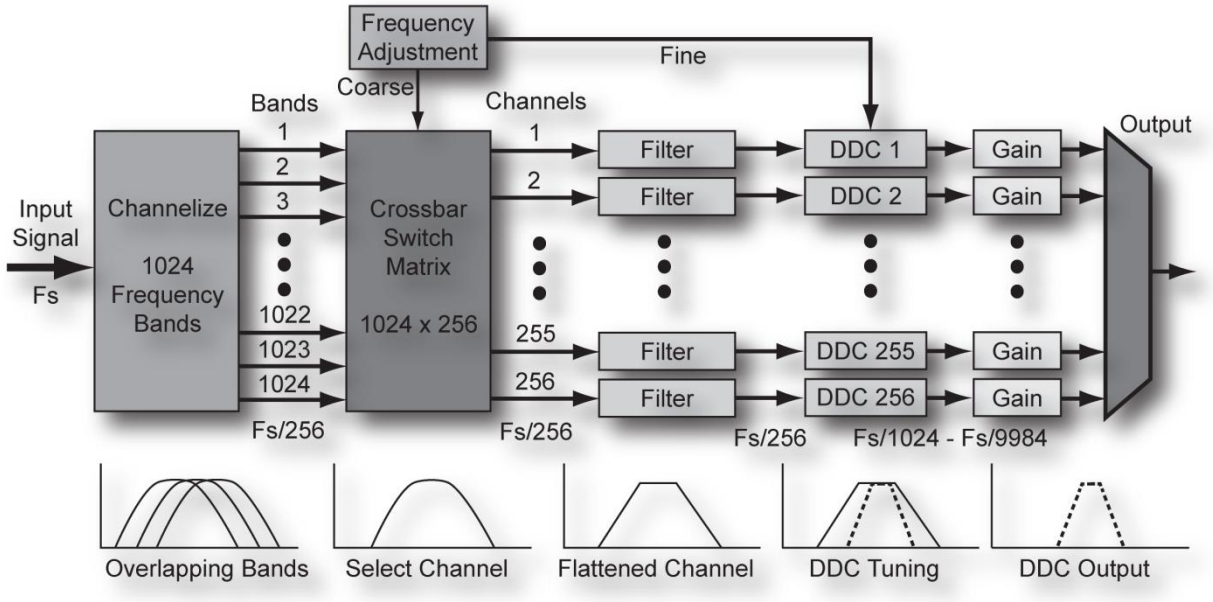

Figure 25: Channelized DDC Block Diagram[9]

[9] Figure 25 is adapted from Figure 3, "New FPGAs Revolutionize Digital Down Converters," by Rodger Hosking, used by permission granted by Allan Margulies, COO, SDR Forum

4.11 Design Principles and Tools for FPGA Development

To facilitate the movement to FPGA based radios, development principles and tools are critical for the design process. The traditional design flow for FPGA development is shown in Figure 26.

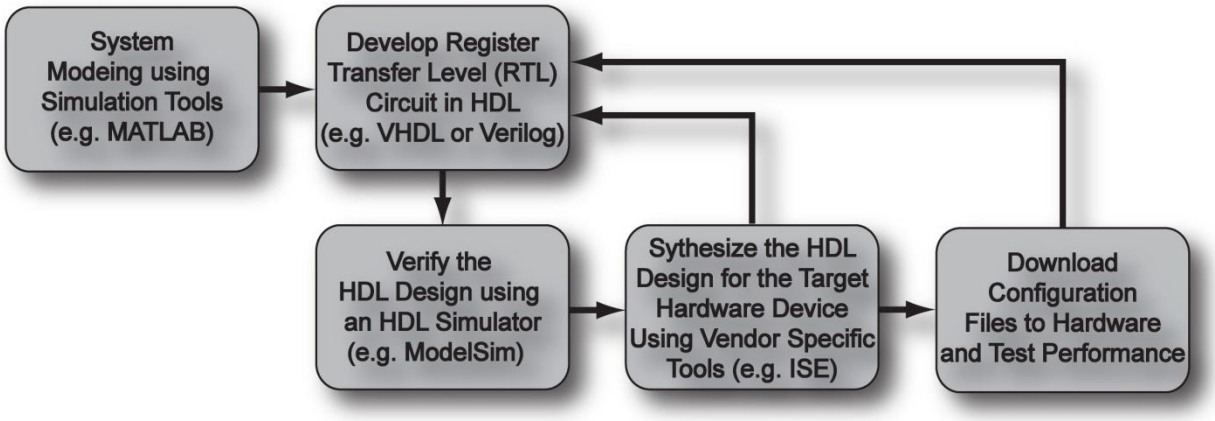

Figure 26: Traditional FPGA Development Flow

Most FPGA designs rely on a hardware description language (HDL) based design for the logical functions. These logical functions are linked to actual physical architecture elements in the FPGA during the synthesis step [Rivoallon 07] using vendor specific software tools such as Xilinx Integrated Software Environment (ISE). During synthesis, the FPGA design tools translate the original HDL description and constraints into an EDIF (Electronic Design Interchange Format) file. The EDIF file is a netlist that contains basic elements such as LUTs, flip-flops, etc., but does not control how they will be packaged together on the FPGA [Rivoallon 07]. The mapping step that follows uses the EDIF file to fit the design to the hardware resources available on the device. Then during place and route, the design is placed onto the FPGA fabric and physical interconnections are identified based on timing constraints. As a final step, a programming file is generated that can be loaded into the FPGA. However, this traditional design flow often presents problems with timing closure as signal timing is only checked at the end of the design flow [Raje 04]. Timing closure issues often arise where the timing fails to meet constraints, thus requiring modifications to the RTL design and/or constraints, followed by re-synthesis. Since adjusting the timing for one fragment of the design can alter timing for another fragment, this process can take significant time to converge on a solution [Raje 04]. Therefore, a new process called physical synthesis is incorporated into modern FPGA design tools that provides information about critical signal paths during the synthesis step, providing some help to resolve timing closure [Rivoallon 07].

FPGA design conforms well to a modular design approach using hierarchal blocks. Modular design allows independent development on parts of a design, often by different developers, that can be joined together to form a cohesive system. In this approach, a top-level design lays out constituent processing blocks, the I/O for each block, and the position of blocks relative to the others in the overall design; where each processing block may contain additional sub-blocks. As an option, the design layout, called the floorplan, can be done before module development with the help of FPGA vendor design tools. Floorplanning provides for resource budgeting and helps to improve timing closure during synthesis. A schematic view of the register-transfer logic (RTL) description aids in viewing the layout of the design modules and the paths connecting them. As shown in Figure 27, the partitions that contain modules have been allocated area on the FPGA fabric and timing constraints have been apportioned along the paths that span the partitions [Raje 04].

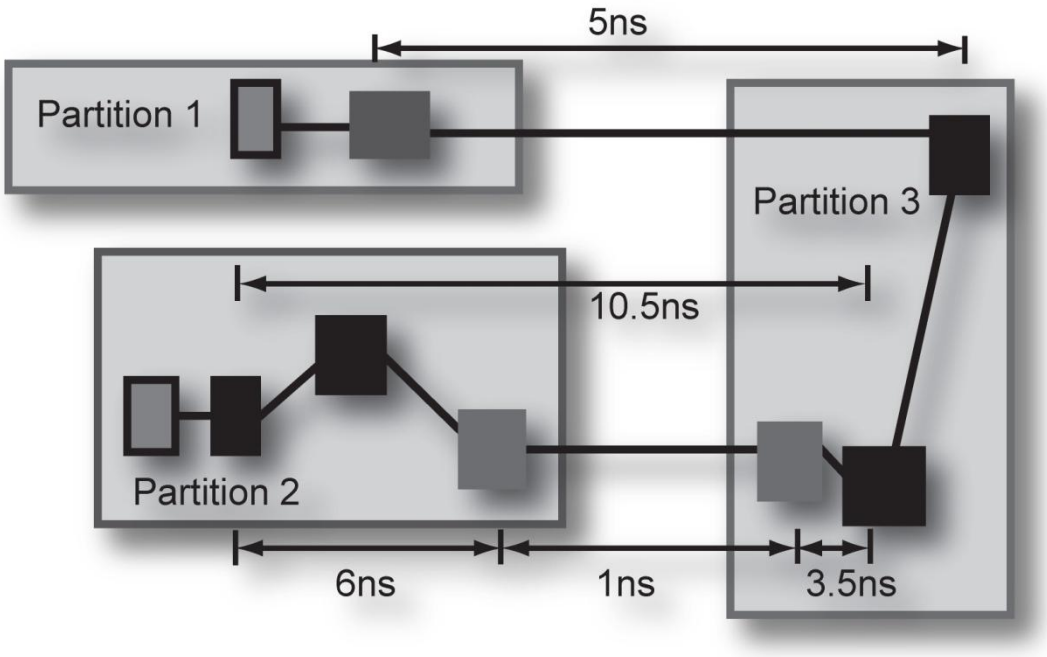

Figure 27: Floorplanning with Timing Budget Constraints

When determining the partitioning for the individual processing modules used in the schematic and floor plan, FPGA-friendly partitioning principles should be followed. These principles will aid in timing closure and substantially reduce the overall design time by simplifying the coding, synthesis, simulation, floorplanning, and optimization phases of the design. These guiding principles include [Scott 06]:

- *Sub-blocks should be synchronous with registered outputs.* Registering outputs helps to eliminate issues associated with logic optimization across design boundaries and allows the synthesis tool to implement combinational logic and registers in the same logic block.
- *Related combinational and arithmetic logic should be in the same design module.* By keeping both forms of logic in the same design module, the synthesis tool can optimize critical paths and share logic hardware resources.
- *Logic with different optimization goals should be separated.* By isolating critical timing paths from non-critical paths, the design tools can synthesis more efficient logic. Therefore, if one path requires area optimization while another path requires timing optimization, these can be placed in separate modules.

After the processing modules are specified using these principles, the abstract graphical floor plan can be used to size and anchor the design module blocks to device resources. Global resources such as clock drivers and external I/O should be assigned first using optimal positions and locked in place. They are often implemented with specialized drivers such as DCMs or serializers/deserializers, which have specific locations on the package the must be accounted for in the floor plan. After global resources have been assigned, the individual module blocks can be placed and allocated resources, including LUTs, registers, memory, or DSP elements. It is important that the module's region be assigned sufficient resources in the floor plan, which are initially based on estimates; but an iterative approach is used to resize the regions during development. The logical connections between blocks are based on the schematic and their associated resources are shown as flywires superimposed on the package in the development tools. Modules that are heavily interconnected should be placed next to each other while providing enough physical resources to accommodate the design logic [Scott 06].

FPGA Graphical Development Tools

The development environment for FPGAs is challenging, requiring designers to think in terms of a parallel implementation. Programmers have desired to have a C-based development environment for FPGA design where software programmers could become FPGA developers without additional training in hardware design. However, C-based FPGA programming has yet to become main stream since the technical challenges of a C-based FPGA compiler are difficult to meet. Language variants that provide a C-like environment for FPGA synthesis are available but often generate inefficient hardware or end up creating a new language that merely adopts the C syntax. One such language that attempts to confront the C-based design aspiration is SystemC.

Nevertheless, SystemC is not straight C programming; rather it uses the C++ class mechanisms to model the hierarchical structure and uses combinational and sequential processes to describe hardware, much like VHDL or Verilog [Edwards 06]. This language structure forces programmers to deviate from the traditional sequential programming methodology and think in terms of concurrent execution. Due to the challenges present in C-based FPGA programming, combined with the desire to raise the abstraction level for development, graphical FPGA design tools are becoming popular for applications utilizing streaming data.

Xilinx and Altera both provide graphical design tools for FPGA design, named System Generator and DSP Builder, respectively. Graphical design flows available in these design environments provide an intuitive way of visualizing the program flow of an FPGA design. The overall goal of the graphical development tools is to bring FPGA development to a wider audience by requiring less knowledge of the underlying FPGA architecture. Each tool provides libraries of common IP algorithm blocks to facilitate fast and easy system implementation. In software radio systems, communications blocks are complicated and the development is rigorous; therefore, these pre-designed libraries are especially useful for signal processing applications where designers want to make use of optimized algorithms, such as Viterbi decoders and FFTs.

As shown in Figure 26, traditional FPGA DSP development requires many different tools, including such programs as MATLAB, ModelSim, and Xilinx ISE or Altera Quartus. The initial system-modeling phase utilizes MATLAB to explore and validate a DSP algorithm, often times providing test vectors for system verification. After the algorithm is determined, initial implementation is done in languages such as VHDL or Verilog, typically utilizing vendor HDL libraries, and verified through an HDL simulator such as ModelSim or ActiveHDL. HDL simulators use the test vectors from the MATLAB system-modeling phase to verify the HDL code; however, since MATLAB and HDL simulator packages are not closely tied, the debug and verification stage becomes time consuming. Lastly, the HDL files are moved to the vendor specific FPGA tools, such as Xilinx ISE, to synthesize and generate FPGA bit-streams. Significant time is often spent in the ISE environment, as synthesis of HDL is more complex than simulation. The designer must ensure the algorithm uses the correct hardware components and meets timing to ensure system performance and stability. Each step in this process is time intensive and requires extensive knowledge of hardware design principles. Additionally, as FPGA technology evolves, the architecture is becoming more and more complex, hindering low-level design principles. Therefore, new tools have been developed to accelerate the process. These tools still require all the implementation steps, including synthesis, place and route, etc., but the designer is not explicitly involved in every step.

Xilinx System Generator for DSP™ was first introduced to the FPGA DSP community in 2000 [Haessig 05] and has since made a significant impact on software radio development. System generator takes a different approach to FPGA design by incorporating the entire design process into one tool. The Mathworks refers to this method as *model-based design*, where all the design, verification, and implementation steps are derived from one central system model [Mathworks 08]. System Generator can be use to develop a graphical system model, simulate the model to verify operation, then derive FPGA bitstreams from the system model through automatic code generation. By using one central system model for each step in the development process, the design productivity and algorithm assurance greatly increase. In this environment, each step in the development process is now closely linked with each other. Therefore, if a change has to be made to the original system model, a new simulation is easily performed. The change can then be incorporated into the hardware implementation without having to individually repeat each step, as in the traditional design approach. Figure 28 shown below depicts the model-based design approach used in *Xilinx System Generator for DSP™.*

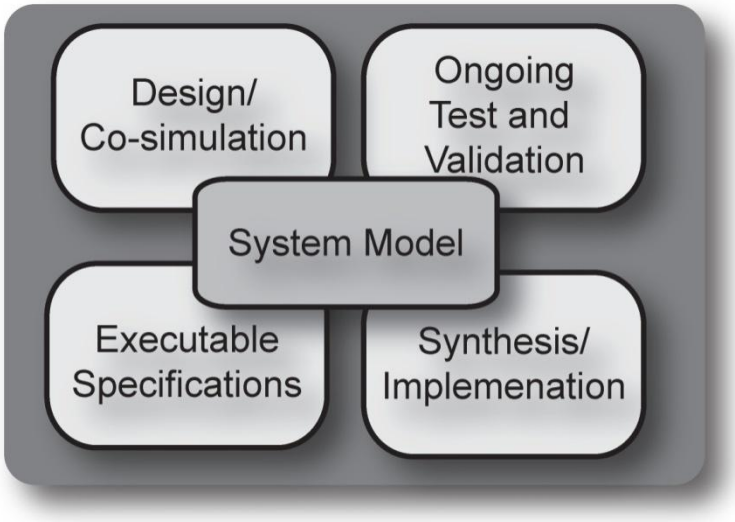

Figure 28: Model Based Design

System Generator uses a graphical development environment to create the model-based design approach. Through this graphical environment, the data flow of an FPGA design is more intuitive than standard HDL, bringing FPGA design to a wider audience. System Generator provides many high-level communications (e.g., Viterbi decoder, demodulators) and DSP (e.g., FFT, FIR filters) blocks that facilitate rapid system development through pre-optimized code. Additionally, System Generator includes lower level building blocks to interface more directly with the

hardware, such as registers and logic units. The custom hardware modules, such as the Virtex-5 DSP48E slices, are also instantiated in the graphical environment with special building blocks available in the Xilinx Block Set. Lastly, custom HDL code can be included through black box modules, adding flexibility to System Generator. Since all these blocks are included in Simulink, simulations can be run on the design for verification and debugging.

Despite the benefits of a model-based approach, there are also disadvantages to using System Generator for FPGA development. The co-simulation between System Generator designs and HDL code requires System Generator to pass simulation parameters to an HDL simulator where it is evaluated and passed back in a clock-cycle by clock-cycle process. The simulation is significantly slower than doing a straight System Generator simulation or an HDL simulation. Therefore, if the developer has HDL code they wish to implement, the simulation process can consume significant development time. Additionally, System Generator prevents some of the low-level control over FPGA functions and debugging a synthesis problem is difficult since it hides much of the synthesis process from the developer. System Generator is best used when the design can effectively use many of the pre-design IP blocks for signal processing, but does not work as well for control logic.

Chapter 5

Selecting Digital Hardware Solutions

5.1 Trade-offs between DSPs, FPGAs, and ASICs

Determining the best hardware solution for implementing a software defined radio is a challenging task for design engineers. DSPs, GPPs, FPGAs, and ASICs each provide their own advantages but it is important to realize that every design is unique and there is no universal solution for choosing among the devices. In fact, most designers use a combination of devices to implement the overall system, a method often referred to as heterogeneous processing. The trade-offs come when considering which device is most efficient for a certain algorithm. Items such as cost, speed, and flexibility, as well as power and optimization, all have to be considered. Another less obvious consideration includes the design team's skill set. The team must be well versed in the design tools available for the project; otherwise the engineering costs could increase.

Historically, DSPs have often been the device of choice as they are a mature technology with strong development tools. A strong benefit of using a mature technology in a software defined radio design is the reduced development risk associated with the project. The design team can be confident that the tools available will lead to a solution within the capabilities of the device. However, DSP processing performance can be an issue for newer, more demanding waveforms and they may not be able to keep up with high data rate processing tasks. Also, DSP processors have a fixed architecture that limits the efficiency and algorithm flexibility since the processor is limited by fixed data widths, fixed hardware accelerator blocks, fixed multiply-accumulate blocks, and fixed memory resources.

ASICs provide much higher device efficiency per unit area at the cost of extremely limited algorithm flexibility and high development costs. Following Moore's Law, manufacturers have been making 60% more transistors available per area of silicon each year. While this provides large computational potential, the complexity of designing algorithms for an ASIC device is large. Sophisticated ASICs requires significant development time and verification, possibly extending beyond the life cycle of the device. Additionally, producing ASIC devices is particularly expensive compared to other alternatives, such as FPGA designs. Therefore, ASIC implementations tend to be better suited for highly complex problems or high volume applications, such as cellular phones.

FPGAs can offer large performance gains on DSP algorithm implementation over instruction set processors. The architectural flexibility allows for algorithm optimization and parallelism,

which enables the device to excel at block computations. For instance, the device can concurrently complete an FFT on one set of data while another block is passing data through a FIR filter. While parallelism can be found in most modern DSP processors, it is not nearly as extensive as that available in FPGAs and the parallel architecture is typically not reconfigurable.

FPGAs are comprised of reconfigurable logic that can be configured based on the desired functionality of the system. Thus, the same FPGA can implement a QPSK transceiver, an OFDM system, or an FSK demodulator by loading different configurations into the device. Current high-density FPGAs, such as Xilinx's Virtex series or Altera's Stratix family, incorporate various silicon resources that allow for complete system implementation inside the FPGA. While FPGA devices can provide significant computational ability with relatively good cost for software radios, they still lack in several areas. FPGA development is time consuming due to the high complexity, often requiring several times longer development cycles than a comparable DSP implementation. The longer development cycles are largely associated with the FPGA development tools, which are immature compared to DSP processor development tools. FPGAs also are limited by their significant power consumption, requiring several times more power than an ASIC implementation. Nevertheless, when compared to ASICs, FPGA cost, complexity and development time are often superior. Table 3 below summarizes the several aspects of the digital hardware performance associated with software radios.

Table 3: Comparison of Hardware Architectures

Device	Parallelism	Gate Reuse And Time Sharing	Flexibility/ Programmability	Power Usage	Speed
General uP	Some	High	High	High	Moderate
DSP uP	Some	Moderate	High	Moderate	Moderate
FPGA	High	Some	Moderate	Moderate/High	High
ASIC	High	None	Low	Low	High

5.2 Design Methodology

The digital hardware design decisions for software radios are challenging but critical decisions of the success of the radio. Not only does the digital hardware define the functionality of the software radio, it also determines how difficult the software radio is to develop. Since there are many choices available, each to their own application niche, an analysis of the pros and cons of each architecture should be performed when considering which to use in the radio design. While DSP

processors have been used in many programmable radio systems to date, the evolution of wireless communications is leaving DSP processors with insufficient resources to implement the latest wireless standards, at least by themselves. 3G and 4G systems that provide added data capabilities are relying on new signal processing techniques such as multiple-input multiple-output (MIMO), orthogonal frequency-division multiple access (OFDMA) and multi-carrier code division multiple access (MC-CDMA) [Altera 07]. These techniques provide faster data throughput at the expensive of intensive computational needs, especially for error correcting algorithms. Therefore, it is important to be able to analyze and determine processing architectures that can support these advanced signal processing techniques.

Providing a fixed methodology for making digital hardware design decisions is a difficult endeavor. While many of the benefits and issues associated with the available architectures are apparent from a high level, further analysis can reveal several implementation issues that are not obvious at first glance. When making a decision on what type of digital hardware to use in a radio system, it is especially important to consider the application requirements of the radio. The design team must identify design parameters for how the radio will be used, which includes factors such as what type of services should be provided, what the size and weight of the radio should be, what type of environment the radio will be used in, how the radio will be powered, and what type of flexibility should be available. These design parameters translate into design constraints that need to be satisfied in the overall radio design. However, several of these design constraints are often conflicting, where improvement on one objective will result in weakening one or more of the other design objective. For example, there is a fairly direct relationship between power efficiency and flexibility; therefore, if additional flexibility is needed in the radio, the design team must tolerate additional overall power consumption. Also, the programmability and maintainability of the processing architecture are equally as important as the processing capabilities of the digital hardware. Even the most powerful processing architecture is useless if it cannot be effectively programmed and maintained. Both the capabilities and the programmability aspects need to be carefully considered since they can lead to high engineering costs and the viability of the software radio.

While it is impossible to provide an exact methodology that would yield the best solution for every combination of application requirements and digital hardware architectures, there are ways to evaluate the trade-offs and help developers make suitable decisions for the digital hardware needed in a software radio. In this section, a design methodology is proposed that helps identify the process that engineers should follow to when considering the available options and

making hard decisions. While this methodology is presented from a high level, it can be expanded to include lower level parameters that are needed in a radio design. It does take into account how to identify when a parallel processing approach is required, but a further analysis of parallel processing theory is needed to determine which combinations of DSP processors, GPPs, FPGAs, and ASICs would yield the most efficient radio design. Nevertheless, there are some general rules-of-thumb that can be followed for utilizing multiple processing devices. Therefore, while this methodology does help explain the process engineers go through when making design choices, it is mostly offered as an educational tool.

As shown in Figure 29, the design methodology includes two phases: the architecture selection phase and the device selection phase. The first step in the architecture selection phase is to determine the requirements of the radio system. These requirements should be identified through an exhaustive list of questions about how the radio will be used and the development effort required. The questions should be separated into sections based on the design parameters. Answers to these questions may come from a variety of sources including a market survey of what type of functionality is desired, the wireless standards (e.g. IEEE 802.16) requirements, the requirements of the overall system that the software radio will be used in, and practical considerations based on engineering experience. By going through the answers to the questions, a further analysis should be done to identify the high-computation processing tasks and the co-processing tasks needed in the system; some of these tasks may include Turbo coding, Viterbi decoding, RAKE receivers, and user interfaces such as LCD displays and keypads.

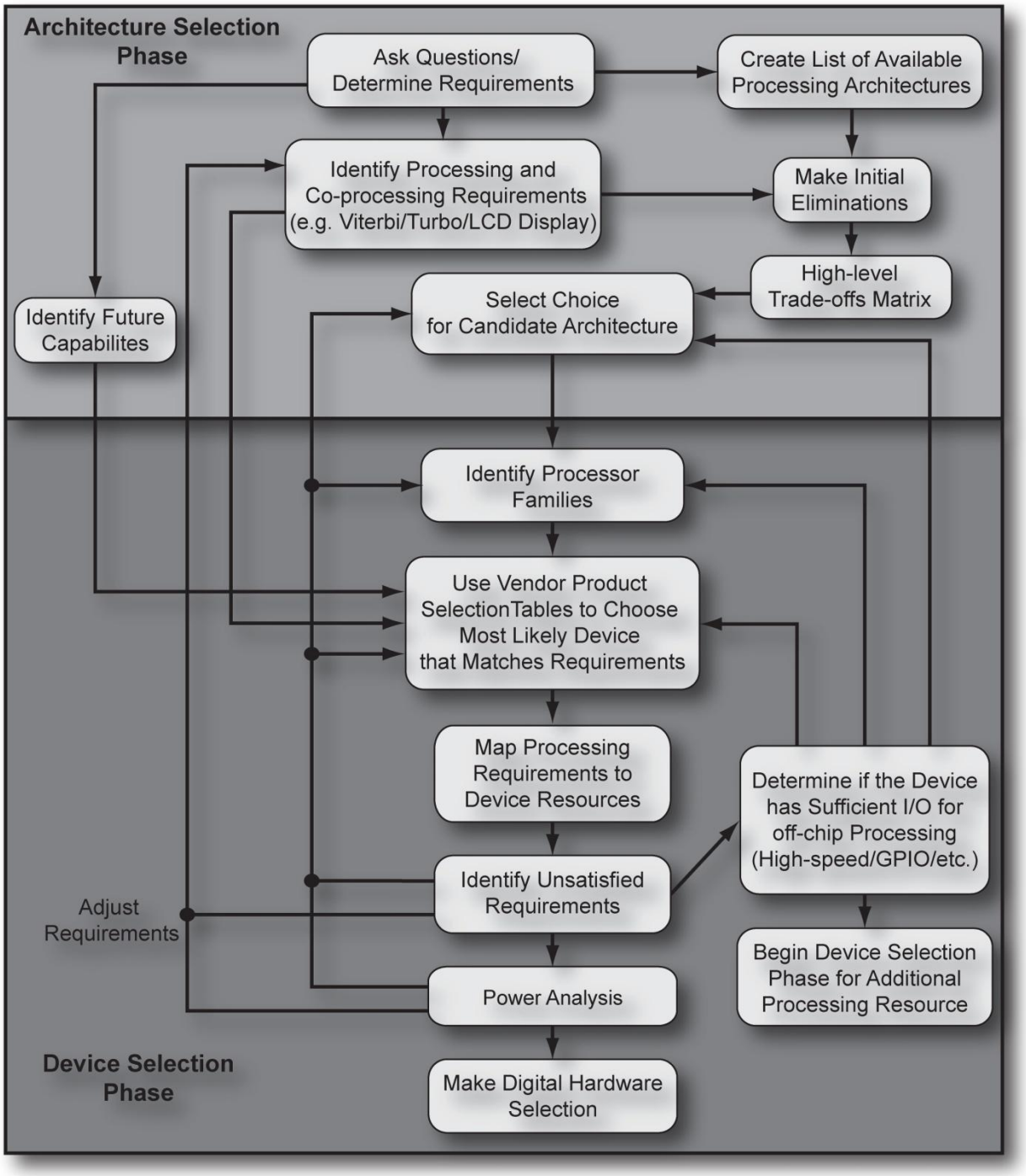

Figure 29: Design Methodology for Selecting Digital Hardware Solutions

After the system requirements have been laid out, a list of the available processing architectures need to be identified, such as high-performance DSP processors, low-power DSP processors, FPGAs, etc. To reduce further analysis, the solution space should be cross referenced with the design constraints and initial eliminations should be made based on inspection; for

instance if a multi-user basestation with high-speed data capabilities is the application, then by inspection the uniscalar low-power DSP processor can be eliminated based on insufficient processing resources.

After all the potential processing architectures are identified, a trade-offs matrix should be created that identifies parameters such as performance, flexibility, power and development time on the rows for each potential architecture on the columns; the parameters should correspond to the sections specified during the question phase. These should be rated based on a signal processing perspective from 0 to 10, where 0 yields poor results for a parameter while 10 yields excellent results. These ratings identify the capabilities of the specific architecture and while mapping numerical values to design parameters is somewhat arbitrary, it can provide a good indication of the relative parameter strengths and weaknesses of the digital hardware architectures. These numerical values are dependent on all the architectural features described in this thesis and possibly several other lower level parameters.

Table 4: Digital Hardware Trade-offs Matrix

Design Parameter	Weights	GPP	High Performance DSP	Low-Power DSP	FPGA	ASIC
Performance	%	5	7	3	9	10
Flexibility	%	8	8	8	6	1
Power	%	1	3	7	2	10
Parallelism	%	3	4	1	10	10
Development Time	%	6	7	7	4	2
Upgradability	%	7	6	6	5	0
Summation						

The resulting numerical values offer insight into the strengths and weaknesses of the design parameters of the hardware architectures relative to other architectures. Nevertheless, applications requirements will desire to emphasize some design parameters over others. For instance, in a mobile radio, power efficiency will be a critical design parameter; therefore power efficiency will need to be emphasized over flexibility in this application. To emphasize certain design parameters over other parameters, weights need to be determined and applied for each of the design parameters. The gains are based on the initial questions used to identify the system requirements and should be fractional weights that add up to 1. These weights should be

multiplied by the corresponding rows in each column and the columns should then be summed where the maximum value will provide the most likely candidate processor architecture.

After the most likely candidate architecture is identified, the first step of the device selection phase is to identify the processing hardware families that correspond to the candidate architecture. These families are available from various vendors, such as Analog Devices's TigerSHARC processors, Texas Instruments' C6000 DSP processors, or Xilinx Virtex-6 FPGAs. Once the desired product families are determined, the product selections tables available from the vendor for each family should be studied; the device that most closely matches the high computational processing tasks and the co-processing tasks found in the first phase should be chosen. After a device has been selected, the required processing tasks should be mapped to device resources. Specific tasks such as Turbo coding and Viterbi decoding should be mapped to any available co-processor functional blocks while user interface tasks should be mapped to any available general-purpose resources that can support the desired functionality. A clock cycle or resource usage estimate, as discussed in the following section, should also be considered to identify if the main computational elements can support required software radio components. The results of this mapping exercise should identify if any of the processing requirements are unsatisfied by the selected device.

If there are any unsatisfied requirements, the developer needs to determine whether off-chip processing can be tolerated for the unsatisfied requirements. If off-chip processing is not an option, the process must return to one of the previous steps and either a different device, device family, or processor architecture must be chosen. If the developer determines that off-chip processing is an option for the unsatisfied requirements, then the input-output (I/O) ports must be analyzed to verify that the device can support sufficient I/O communication for the required tasks. This communication may require general-purpose I/O ports or high-speed interconnects through external memory interface buses. Assuming that sufficient I/O capabilities are available, the device selection phase for the additional digital hardware architecture should be performed. After the device selection phase for all the desired processing devices has been completed, a power analysis should be performed to ensure that the design will perform within system constraints. Once final device selection decisions are made, development of the printed-circuit board can begin.

This two phase methodology is presented from a high-level to yield the most likely processing architectures and devices. It is an iterative process where selections are made and evaluated then either selected or discarded. Also, sometimes during the process, the initial system requirements may have to be adjusted if an initial system constraint is impractical or impossible.

Therefore, the requirements are adjusted and the process is repeated until a solution is found. The following section discuss how to make resource estimations for identifying the processing and co-processing requirements of the candidate digital hardware solutions needed in the architecture selection phase; these requirements are what is mapped to the general purpose resources of the processing architecture.

5.3 Resource Estimation

Since resource usage is typically derived from performance requirements, it is important to have methods for estimating available performance on candidate hardware platforms; nevertheless, these methods often are time intensive and inefficient. DSPs, GPPs, and FPGAs have significant variation in architectural layout and parallelism that complicates the evaluation process. Simply estimating the chip's clock rate is insufficient for performance analysis since the hardware may take advantage of hardcore resources and execute multiple operations in a single clock cycle. To further complicate the process, software radio systems have numerous waveforms that have to be evaluated and significant variations are found in hardware architectures. Additionally, many software radios make use of multiple hardware components, i.e. a DSP and FPGA, where signal processing is split across platforms [Neel 05], thereby introducing partitioning and inter-processor communication issues. Determining the best hardware solution requires analysis of signal processing partitioning, which is quite tedious for numerous available waveforms.

The most accurate way to evaluate performance is use vendor optimized IP libraries or write assembly code for every candidate platform and test the performance. However, vendor supplied libraries may be limited for the desired application and high engineering costs can be associated with this method, including the purchase of test hardware and increased design time when writing assembly code. One evaluation method requiring less design time is to write high-level programming code and compile for each candidate architecture. Issues also persist here, as performance variations between compiled code and assembly code are often significant. Compilers may create inefficient implementation methods or may not take full advantage of the hardware resources available. For example, a FIR filter design compiled for a TI DSP may take advantage of all the available hardcore multipliers on the chip; however, a compiler for an Analog Devices DSP may not recognize the need for hardcore multipliers and use general processor resources. The Analog Devices implementation will be significantly slower only because the compiler does not know to take advantage of its own hardcore multipliers. This may require reformation of the algorithm so it is more compatible with the hardware architecture.

Third party hardware evaluations are available from companies such as Berkley Design Technology Inc (BDTi), but are useful only if analysis has been done on the processor and the algorithm desired. Hardware vendors often develop and use design components that are not available to third party vendors for evaluation. Based on these issues associated with hardware platform evaluation, a more efficient method for analysis is desirable. The method should effectively estimate the number of required cycles, the execution time, the energy required, and the memory used for an algorithm on a given platform. It must not only be usable for currently available wireless standards, but extensible so the analysis can be expanded for future waveforms [Neel 08]. By accurately evaluating these core aspects, useful design decisions can be made during the design process.

A method for evaluating hardware components for making useful design decisions is listed as follows [Neel 05]:

1) Identify initial pool of candidate processors, containing GPPs, FPGAs, DSPs and ASICs.
2) Survey the required waveforms to identify the required components[10] and component execution times
3) Estimate resource utilization for each component as implemented on each processor
4) Using the results from step 3, form the set of processor solutions for examining all possible processor and waveform partition combinations
5) Form the feasible processor solution space by removing all combinations of processors that violate constraints or have insufficient resources.

After an initial pool of candidate hardware platforms are identified, Step 2 in the methodology, estimating the required number of clock cycles for an algorithm is a critically important step for evaluating hardware performance. Figure 30 depicts the design methodology for cycle estimation. Two separate paths are used in this design methodology. In path 1, existing vendor profiled code libraries are used to estimate cycles. While this will provide a reasonably accurate estimate, pre-existing code libraries are often limited and not available from every vendor for the DSP blocks desired, therefore a more generalized method is also needed.

[10] A component is an individual processing element used as a building block for creating a software radio waveform.

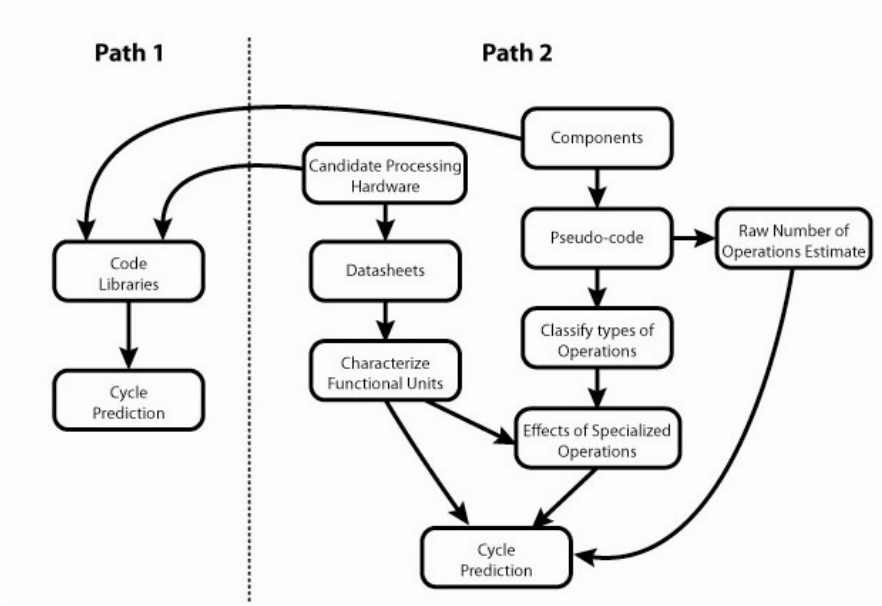

Figure 30: Cycle Prediction Methodology[11]

The second path in this methodology attempts to describe the resources used by a component through parameterized implementation equations [Neel 08]. It is primarily focused on instruction-based architectures but can also be expanded for use with FPGAs. To generate the parameterized equations, both the processor and the component have to be characterized. The processor characterization identifies several parameters by examining the processor's instruction set and datasheet. The peak clock rate, peak power consumption, and native word size are noted along with characterizing the functional units, namely the special parallel architecture blocks. For VLIW architectures these parameters include the total number of unique instructions that can be completed in one clock cycle, and similarly, the total number of memory, arithmetic, and multiplication instructions that can be completed in one clock cycle. For SIMD architectures, the number of data words that an instruction can be applied to is noted. Lastly, any special single cycle instructions, addressing modes, and numeric representation formats are distinguished.

Characterizing a component requires a more involved process than processor characterization. An initial operations estimate of the component is done through the use of pseudo-code in a serial instruction format, capturing all of the relevant operations, including memory accesses and loop control operations. Any assumed processor characteristics for word width, addressing modes, and numeric formats need to be noted. After all relevant operations have

[11] Figure 30 is adapted from Figure 1 in "PCET: A Tool for Rapidly Estimating Statistics of Waveform Components," by James Neel, used by permission granted by Allan Margulies

been listed, each line of the pseudo-code should be labeled and the loops identified with a unique numbering scheme for each loop. From the pseudo-code, equations are created for the number of data memory operations, the number of multiplications, and the number of arithmetic operations; the sum of these equations creates a total estimate for all the operations required by the waveform component. While this provides a worst-case clock cycle estimate for a serial implementation, the parallel functional units and dedicated processing blocks will reduce the number of clock cycles required. Therefore, the pseudo-code is reviewed to identify where specialized operations may be utilized and these are subtracted from the total estimate. However, some specialized operations will target the same line of code, thus these duplicate eliminations must be identified and added back into the estimate [Neel 08]. As a final step, any known profiled code for a component on a particular processor should be included in the analysis as a comparison.

Compared to instruction-based devices that use fixed processing units, FPGAs use configurable processing fabric components, such as logic blocks, memory units, and embedded functional units to implement waveform components. Therefore, resource estimation is generally more difficult than doing a cycle estimate. Vendor libraries provide a useful resource estimate for waveform components but are not available for every element. For instance, if some components are available from Xilinx and others from Altera, conversions between processing fabric elements can be approximated [Neel 05]. A rough equivalence can be found between the basic computation elements, Xilinx slices and Altera ALM's, and used to estimate a waveform component on the other architecture.

After obtaining the cycle predictions and the resource utilization using steps two and three of the evaluation methodology, a feasible set of processor solutions can be identified. Using the cycle and resource parameters, all possible partitions across hardware platforms based on area and cost constraints should be found first. Then using the timing constraints and processor resource requirements that were found in the previous step, all hardware architectures that cannot support the waveforms based on insufficient cycles, insufficient memory, insufficient multipliers, or limited FPGA fabric elements should be removed. Lastly, the remaining hardware architectures should be evaluated based on power consumption requirements and any implementations that violate power constraints should be eliminated.

5.4 Cycle Prediction Methodology Example

To illustrate an example of the cycle prediction methodology, an arbitrary length real-valued FIR filter that calculates results on a sample-by-sample basis is demonstrated. As discussed

in path two of the methodology, the component is characterized by pseudo-code, listed in Figure 31. This FIR filter implementation uses a tap length referred to as “N” and assumes the processor has a circular addressing mode available. Furthermore, to allow for high code portability, the simplest loop control method is used, requiring three separate instructions: one to decrement the loop counter, one to compare the counter with zero, and one to branch based on the comparison result [Neel 08].

```
y=fir(coefficients, data, length, offset)

//Set circular buffer parameters
1  (instruction to store previous setting in local register)
2  (instruction to store buffer length)
3  (instruction to turn on circular buffer)
4  (instruction to set buffer length)

//Move input parameters to local registers
5  R1 = coefficients (address)
6  R2 = data (address)
7  R2 = data + offset              // needed for circular buffering
8  R3 = length (actual #)

//zero accumulator (typically done by subtracting a register from itself)
9  acc = 0

//Note inherent assumption that length > 0
//Note for loops are implemented as conditional branches in assembly

L1  (loop label) R4 = *R1++      //postfix assumption
L2  R5 = *R2++
L3  R6 = R5 * R4                 //Multiply
L4  acc = acc + R6               //Accumulate
L5  R3 = R3 - 1
L6  flag = cmp(R3,0)
L7  if flag (R3==0), branch to loop

// Move result to output register
10  R_out = acc

// Restore Operations
11  (instruction to turn reset addressing mode)
12  (instruction to reset buffer length)
13  (instruction to branch back)
```

Figure 31: Pseudo-code for sample-by-sample FIR Filter[12]

As discussed, each line of the pseudo-code is labeled with a number and looped lines are uniquely labeled with "*L#"* values. The lines are grouped based on memory, multiplication and arithmetic operations, then summed to create raw parameterized equations listed in Table 5 for the FIR filter serial implementation.

[12] Figure 31 is adapted from Figure 2 in "PCET: A Tool for Rapidly Estimating Statistics of Waveform Components…," used by permission granted by Allan Margulies, COO, SDR Forum

Table 5: FIR Filter Parameterized Equations[13]

Group	Equation
Memory	2·N
Multiplication	N
Arithmetic	4·N + 13
Total	7·N+13

Since these parameterized equations are the result of a serial implementation, the pseudo-code needs to be reviewed to identify options for specialized instructions. Several relevant instructions can be used to eliminate clock cycles, for example, BDEC is branch and decrement, BPOS is branch if positive, ZOL is zero-overhead loop (often called block repeat), MEM2 represents double wide memory loads, and NOREG is used to favor direct data memory accesses rather than local registers [Neel 08]. These are applied to the pseudo-code instructions and Table 6 shows the impact on reducing the number of required operations. However, as discussed several of these instructions target the same line of code, therefore the effect must be undone so the line is only removed from consideration once [Neel 08], as shown in Table 7. By using this clock cycle estimate and considering the clock frequency of the hardware, an execution time for the component can be identified. Combining the execution time of this component with the other required components will help identify the feasibility of implementing a waveform on the processor architecture.

Table 6: Special Instruction Cycle Eliminations[14]

Instruction	Impact	Modifier Equation for Reduced Cycles and Memory Accesses
BDEC	L5, L6 eliminated	Processor Cycles – 2·N
BPOS	L6 eliminated	Processor Cycles – N
ZOL	1 cycle to set register, L5, L6, L7 eliminated	Processor Cycles +1 –3·N
MAC	L4 eliminated	Processor Cycles – N
MEM2	Memory accesses cut in 1/4	Memory Accesses – (2·N)/2
MEM4	Memory accesses cut in 1/4	Memory Accesses – 3·(2·N)/4
NOREG	Memory accesses eliminated	Memory Accesses – (2·N)

[13] Table 5 is borrowed from Table 1 in "PCET: A Tool for Rapidly Estimating Statistics of Waveform Components…," used by permission granted by Allan Margulies, COO, SDR Forum

[14] Table 6 is adapted from Table 2 in "PCET: A Tool for Rapidly Estimating Statistics of Waveform Components…," used by permission granted by Allan Margulies, COO, SDR Forum

Table 7: Adjusted Modifier Equations[15]

Instruction	Impact	Modifier Equation
Remove BDEC	L5, L6 added back in	Reduced Cycles + 2·N
Remove BPOS	L6 added back in	Reduced Cycles + N
Remove MEM2	Undo MEM2 effect	Reduced Memory Accesses + (2·N)/2
Remove MEM4	Undo MEM4 effect	Reduced Memory Accesses + 3·(2·N)/4

[15] Table 7 is adapted from Table 3 in "PCET: A Tool for Rapidly Estimating Statistics of Waveform Components…," used by permission granted by Allan Margulies, COO, SDR Forum

Chapter 6

Power Management for Digital Hardware

6.1 Power Management Issues

One of the critical issues facing software radios is power consumption of the digital hardware. As wireless standards evolve to support increased multimedia content capabilities, the processing requirements on software radios are growing and translating into increased power consumption. The goal is to design these capabilities into a small form factor, which typically have limited power constraints and battery life. Additionally, since software radios offer added flexibility over application specific products, their implementations are generally less power efficient. Therefore, methods for designing low power systems to mitigate power requirements are important for factors such as battery life, heat dissipation, and reliability in the final product. Also, the battery size is a big part of the system cost so reducing the power consumption aids in reducing the overall cost of the radio.

Low power software radio systems are desirable for many applications. In base station systems, low power corresponds to smaller, more cost effective solutions requiring less external cooling. Software radios are also often used in mobile devices and it is generally desirable for their form factor to remain relatively small and lightweight. This often presents an issue since high computational capability translates into high power consumption, which converts to heat during operation. The computational components, the radio modules, and other system components generate heat and this heat has to be dissipated to avoid system failure. Since mobile devices have insufficient space for fans, the heat must be passively dissipated through the casing while maintaining an acceptable temperature for holding the device. Therefore, in average size 300-cm^3 radios, the highest amount of acceptable power dissipation is approximately 5W [Arrington 06]. Additionally, weight is a concern and since batteries are often the heaviest component in a software radio, it is desirable to use the lightest battery possible to power the hardware. Light batteries generally do not provide as much stored energy, so powering high performance digital hardware can be a challenge. To satisfy these thermal and battery related issues, strategies for minimizing the power consumption of the digital hardware must be utilized.

The level of flexibility available in digital hardware choices is typically directly associated with device power consumption. Application specific designs, such as those for consumer wireless devices, normally implement only one wireless standard in a relatively power efficient design. In a

software radio approach, additional flexibility is needed to support multiple wireless standards; however this flexibility corresponds to increased power consumption. Figure 32 explains the trade-offs between performance, flexibility, and power efficiency, indicating that the more flexibility and computational power offered, the less power efficient the design.

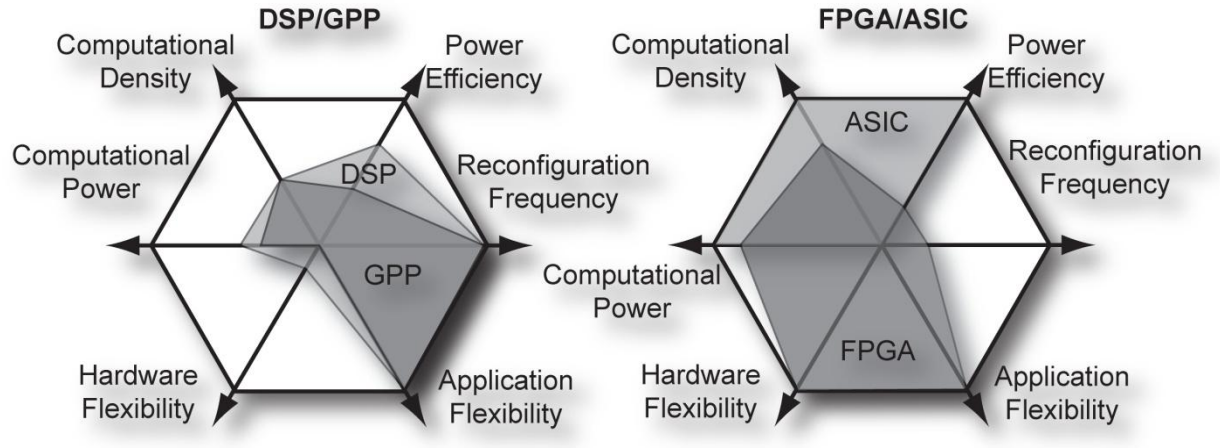

Figure 32: Flexibility vs. Performance

When designing for low-power software radios, a combination of factors must be considered to achieve the best balance of performance and power consumption, including additional factors such as how densely packed the computational elements are and how quickly the device can be reconfigured. Several design techniques are available for low-power operation including utilizing granular control of power modes, exploiting hardware and algorithmic parallelism, minimizing off-chip communication, and optimizing algorithms. The trend toward massively parallel architectures helps to minimize off-chip communications through the use of several parallel processing units. Since on-chip communications operate quicker and more efficient, keeping the data on-chip as long as possible is beneficial for the power budget. This also provides system level power optimization since algorithms that were previously executed on several chips can be implemented in one device, eliminating interface power.

6.2 Low-power VLSI Design

Digital hardware for software radios is implemented with Very Large Scale Integration (VLSI) processes that currently combine millions of transistor-based complementary metal oxide semiconductor (CMOS) circuits on a single chip. VLSI chips have three types of power consumption that comprise the overall operational system power: static power, dynamic power, and interface power [Pelt 05]. Each of these power components is dominated by the silicon process technology used in the manufacture of the VLSI chip and its operating conditions. In previous generation

silicon processes, the dynamic power was the dominating power consumption contributor, but as the transistor size has decreased (e.g., move from 90 nm to 65 nm), the static power consumption is becoming an increasing factor. Static power is a function of silicon area and is often referred to as the leakage power, the power consumed when no signal activity occurs. It is a function of the leakage current present from both the source and the gate to the drain of the transistors. Decreasing transistor size has reduced the transistor channel lengths and gate thickness, thus making leakage current more prevalent in newer digital hardware. Shown in Figure 33 below, the primary leakage current flows across the channel from the source to drain and is the primary static power contributor [Curd 07].

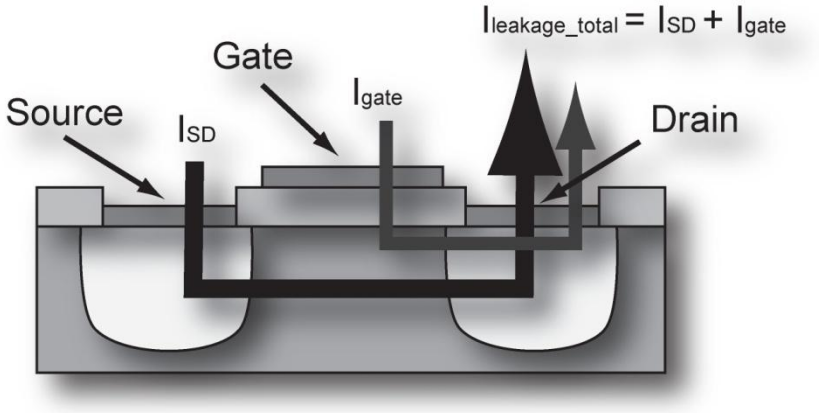

Figure 33: Transistor Leakage Current

While static power is becoming the significant factor, dynamic power is still a large concern for digital hardware designs. Dynamic power is a function of operating conditions and is governed by the following equation:

$$P_{dynamic} = \alpha \cdot C_{eff} \cdot V_{DD}^2 \cdot f_{clk}$$

Equation 1: Dynamic Power Dissipation [Liu 98]

Where C_{eff} is the capacitance of the node switching, V is the supply voltage, f_{clk} is the switching frequency, and α is the average toggle rate [Liu 98]. Since the dynamic power consumption relies on two easily adjustable parameters, voltage and frequency, it is often the target for power-efficiency in digital hardware. Reducing the supply voltage will decrease the dynamic power consumption proportional to the square of the voltage; however this will increase delays in the CMOS logic. The delay of a CMOS logic device can be approximated as:

$$T_D \approx \frac{C_L V_{DD}}{I} = \frac{C_L V_{DD}}{\epsilon (V_{DD} - V_t)^2}$$

Equation 2: Delay of CMOS logic [Liu 98]

6.3 DSP Power Management

The digital hardware is the prime target for power management in a software radios since it is consumes a significant percentage of system power, yet is a component that provides high power flexibility. Selecting a DSP that can provide sufficient computational power while maintaining the overall power budget is a challenge for designers. Low-power DSP processors are designed to help facilitate energy efficient software radio designs through various active power management techniques, VLSI structures, and architectural approaches. These low power approaches involve several factors including voltage and frequency scaling, CPU activity levels, memory utilization, and capacitive loading.

Most newer DSP processors offer on-chip power management capabilities for adjusting the power profile, through both automatic sensing and software control. On-chip methods fall into two categories: active system power management and standby power management [Musah 08]. Vendors may provide different names for their power management strategies, but they all operate under same basic principles. Active power management utilizes three techniques for reducing power consumption; dynamic voltage and frequency scaling (DVFS), adaptive voltage scaling (AVS), and dynamic power switching (DPS) [Musah 08]. Standby power management is also important and must reduce leakage power though system low-power modes during idling.

DSP processors provide ample processing power, yet an application may not always require the full performance available. Newer DSPs separate the chip into various independent power domains. Through DVFS, the voltage levels and clock rates in each of these power regions can be lowered to levels that meet system computational requirements without significant overheard. Since frequency is limited by voltage, the chip has pre-defined processor operating performance points (OPPs) available to ensure sufficient voltage levels for desired clock frequencies. OPPs can be selected dynamically in software based on application processing and peripheral communication requirements. Compared to legacy technologies that operated at 5 V or 3.3 V, new DSPs can operate at 0.9V or even lower. These lower voltages and frequencies provide significant power savings, as can be seen using Equation 3 [Analog 08]. A 25% clock frequency reduction results in a 25% power savings, while a 25% voltage reduction results in a 40% power reduction. Since both these reductions compound for additive power savings, DVFS is an effective power management technique.

$$\text{Power Savings Factor} = \frac{f_{clkRED}}{f_{clkNOM}} \times \left(\frac{V_{DDINTRED}}{V_{DDINTNOM}}\right)^2 \times \frac{T_{RED}}{T_{NOM}}$$

$$\text{Percent Power Savings} = (1 - \text{Power Savings Factor}) \times 100\%$$

Equation 3: DVFS Power Savings [Analog 08]

In Equation 3, f_{clkRED} is the reduced frequency, f_{clkNOM} is the nominal frequency, $V_{DDINTRED}$ is the reduced internal supply voltage, $V_{DDINTNOM}$ is the nominal internal supply voltage, T_{RED} is the duration running at f_{clkRED}, and T_{NOM} is the duration running at f_{clkNOM}.

Adaptive voltage scaling is similar to DVFS in that it adjusts the driving voltage levels, but it is a technique for optimizing performance based on variations in the semi-conductor device during manufacturing and changes over its lifetime. Each chip produced during the manufacturing process has on-chip component variations that determine the overall chip performance. Therefore, some chips may be considered a hot device where a given clock frequency can be achieved at lower voltages than in a cold device. The cold device must provide a higher voltage level to achieve the same performance. Using AVS, the processor can sense its own performance level and adjust voltages accordingly [Musah 08]. This method is helpful to even out power consumption differences among semi-conductor devices, thereby giving designers a better power consumption figure that does not fluctuate in different radio units of the same design.

Dynamic Power Switching is the third active power management technique, operating differently than both DVFS and AVS; it places on-chip components into low-power states during periods of idle activity. Since DSP processors have various power regions, DPS can operate independently on each region depending on the chip architecture. Therefore, if the CPU domain senses that a computational task is complete, it can enter a low-power mode while another power domain performs a DMA transfer. After the DMA transfer is complete, the CPU domain returns to operation within microseconds [Musah 08].

All DSP power management methods require control mechanisms for operating the active and passive power-savings functions to adjust the power profile. Semiconductor vendors typically build in automatic on-chip control mechanisms that do not require programmer intervention, such as voltage and frequency scaling during chip idling. Nevertheless, increased power efficiency can be obtained through more refined software control methods. Low-power DSP architectures give the programmer access to the internal on-chip BIOS, where control registers can be set for DVFS and DPS power management [Patterson 06]. However, command and control of power profiling scheduling is complicated and time sensitive. Abrupt software changes in voltage and frequency can have unanticipated effects; therefore BIOS controlled power management is handled through

an on-chip operating system (OS) scheduler. The programmer has access to the OS scheduler through a Power Scaling Library (PSL) of application programmer interfaces (APIs). Through the APIs, the programmer can gate clocks, activate sleep modes and safely manage transitions between operating performance points for voltage/frequency scaling, thereby enabling safe software control of the on-chip low-power techniques [Patterson 06]. Since the PSL executes safe transitions between OPPs, DSP vendors can provide power management control to programmers while still guaranteeing operation. Figure 34 demonstrates the power management techniques with three operating performance points for user code operations and the PSL transitions between OPPs; note that this figure only considers dynamic power consumption and is governed by Equation 1.

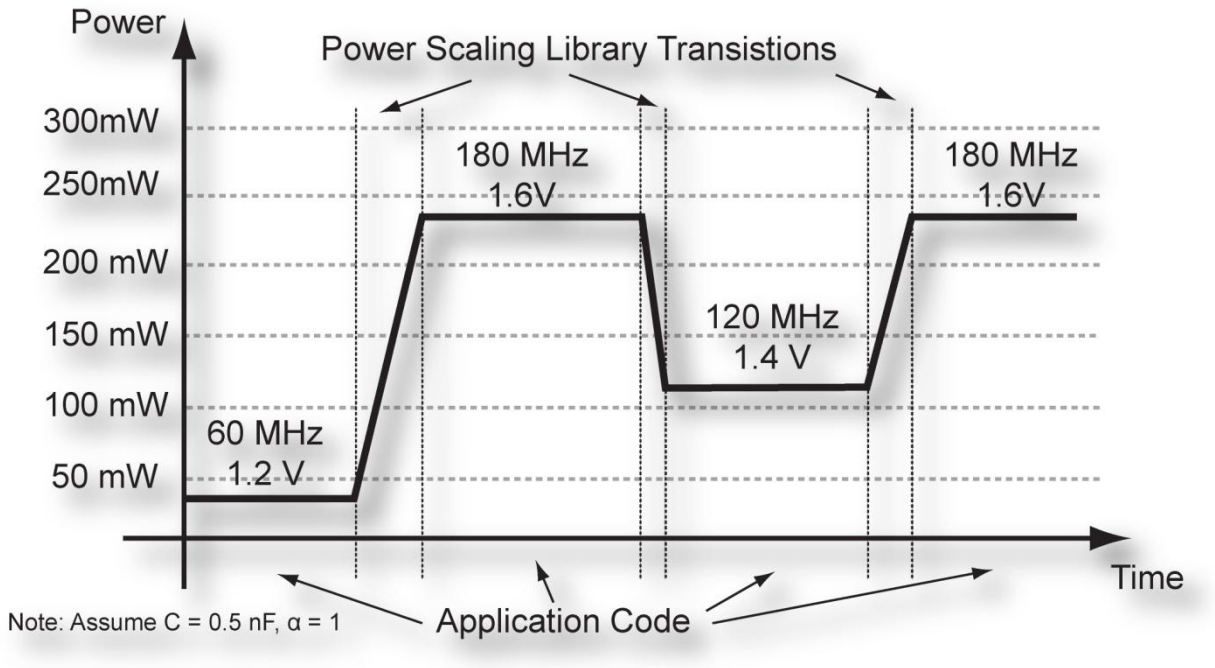

Figure 34: Voltage and Frequency Scaling with PSL transitions

With the variety of software controlled on-chip power management techniques, it is often difficult to determine the required operating power and best approach for reducing power consumption. Power efficient software radio design takes significant effort on the part of the programmer during development. Therefore, chip manufacturers have developed software design tools that give the programmer insight into the power efficiency of a design. With these tools available, the developer has feedback indicating how alternative implementations will affect power consumption. These design tools are integrated into the development software so the programmer

can consider power consumption at each step in the progress and interface with measurement hardware for testing and verification [Patterson 06].

The on-chip power management capabilities available on the DSP processor should be used in combination with additional techniques to provide the most power efficiency design. On-chip DSP memory should be utilized as often as possible, rather than interfacing with external memory, since it draws less current. However, since boot-up instructions only need to be available at power-on, boot-up sequencing is a good use of external memory as it can be shut down during system operation. Also, boot-up sequencing has traditionally powered up all the hardware in a system, but it can be configured to only power up the required components. External peripherals can be shut down when not in use and capacitive loading should be minimized as much as possible. Additionally, algorithmic operations can be optimized for low-power operation through various techniques.

6.4 Case Study: TMS320C55x™ Series DSPs

Texas Instruments is one of the major vendors in the DSP processor market and provides a wide range of processors for different application requirements. Their product line includes fixed-point and floating-point processors, processors designed for high performance, processors designed for low power, and processors designed for specific applications such as digital video [TI 09]. The low-power TMS320C55x™ family of DSPs is specially designed for wireless applications and well suited for the digital hardware in software radios since mobile applications require low-power [Wang 00]. This TI family of processors has a specialized architecture for high performance operation with several power optimization features available to reduce power consumption in the final design.

The CPU in the C55x family is composed of four separate units that interface with various program and data buses for executing efficient and pipelined DSP algorithms. The units are the instruction buffer unit (IU), the program flow unit (PU), the address data flow unit (AU), and the data computation unit (DU) [Wang 00]. Each unit helps to improve computational performance while also helping to improve power efficiency. The IU stores recurring instructions, thereby reducing interface time and power consumption to external memory. Additionally, the IU determines which instructions are executed in parallel to improve performance; this information is then passed to the other three units. The PU generates program memory addresses for fetching instructions needed by the DU. Since many DSP algorithms are cyclic, the PU offers looping capabilities to cycle through repeated instructions. The AU generates data memory addresses for

reading and writing the I/O data processed by the DU [Wang 00]. By utilizing these three units, the DU can focus solely on data computation, thus increasing performance over a standard microprocessor. The C55x CPU architecture, including program and data buses, is shown below in Figure 35.

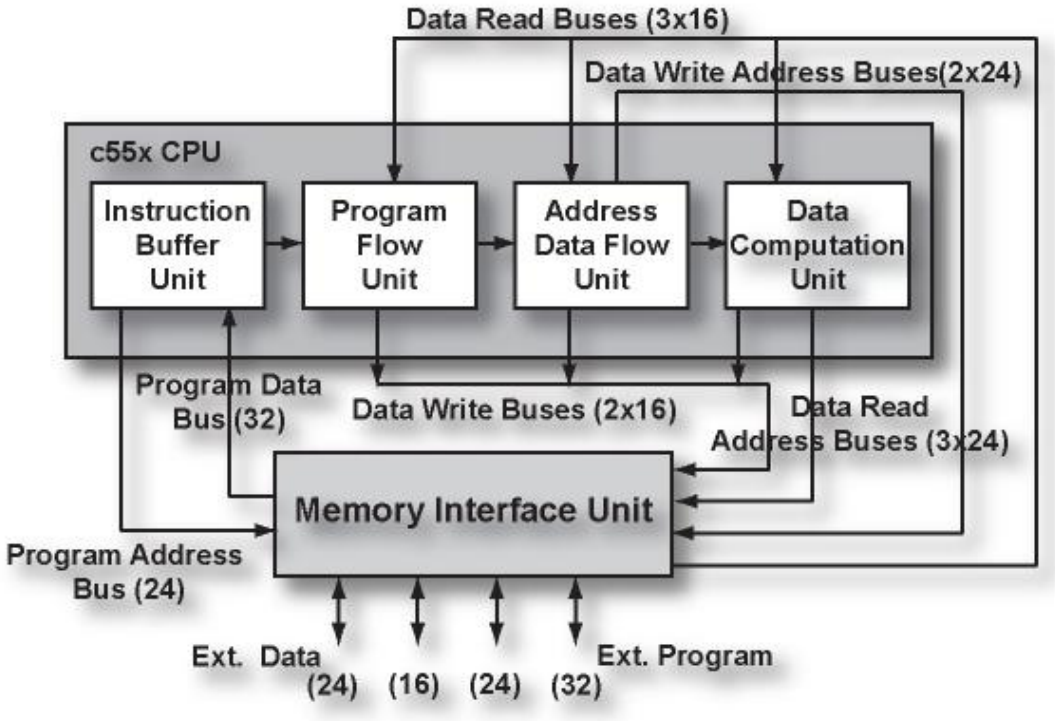

Figure 35: Texas Instruments c55x CPU Architecture

Texas Instruments (TI) helps system designers achieve high performance, low power operation in C55x DSPs through a combination of chip design features and manufacturing processes. The fundamental CMOS technology used during the manufacturing process affects the overall power consumption; therefore TI uses primarily low voltage transistors with low leakage current in C55x DSPs to improve power efficiency [Patterson 06]. While the transistor technology inside the DSP does play a critical role, the majority of the power efficiency comes from the various architectural instruction execution features and power mode options inside the DSP. The specialized CPU architecture with a variable instruction set and parallel processing units helps to facilitate energy efficient DSP algorithm execution. Also, granular power mode control of the hardware components is available and since the power consumption on digital hardware is primarily a component of operating frequency and voltage, throttling of these parameters is possible in different regions of the chip [Patterson 06].

Power-efficient C55x DSPs have several power control options because the more granular power utilization management available, the more the designer can customize the power consumption profiles. The C55x is sectioned into six clock domains, each containing hardware for different chip functions; including CPU, peripherals, cache, clock generation, external memory interfaces (EMIF), and master ports. Each of these domains can be powered-off separately when not in use, yet awakened quickly for operation [TI 08c]. Therefore, the CPU can be powered down during idling until an interrupt occurs, and similarly, the peripherals can sleep until needed. The domains are clocked by one of four clock groups: subsystem, fast peripherals, slow peripherals, and EMIF. When a clock group does not require high-speed operation, it can be set to 1/2 or 1/4 of the frequency of other groups, thus improving power efficiency [TI 08c]. Other features are also available within clock domains such as bus holders that oppose signal transitions on idle buses, thereby reducing the dynamic switching power, and power control on individual hardware elements within a domain.

While the power optimization features available on the C55x help to facilitate energy efficient operation, the designer must take advantage of these features to produce an energy efficient design. The C55x has power management techniques available using a power management (PM) module inside the DSP/BIOS real-time operating system (RTOS) to coordinate low-power operations. The programmer has access to the PM module through APIs. To manage the power management design techniques available on the C55x DSP, TI provides a Power Optimization DSP Starter Kit (DSK) to help system designers program the chip for power efficiency. The tool kit works with the on-chip RTOS PM while providing meters, scope waveforms, test code, etc. to help designers evaluate implementation methods in terms of power consumption [Patterson 06]. Utilizing these power optimization techniques and tools is critical since despite being labeled as a low-power DSP, without an efficient design implementation, the design can still require significant power. It is important to note that low-power DSPs are not inherently low-power, they provide the tools necessary for the system designer to realize a power-optimized design.

6.5 FPGA Power Consumption

FPGAs consume considerable power, creating a major limiting factor for their use in software radio systems since the most highly desired software applications are use in mobile devices. As wireless waveforms continue to evolve, the performance and density requirements increase exponentially. While the latest 65-nm FPGAs provide the computational power to

implement the latest radio waveforms, the programmability and flexibility of the devices make them much less power-efficient than custom ASIC logic designs. ASICs can implement logic functions with much less circuitry than is required by an FPGA implementation. The FPGA configuration circuitry that defines the hardware inside the device has longer wire lengths and higher inter-connect capacitances than found on an ASIC. This increased silicon area translates into high overall FPGA power consumption.

As FPGA devices evolve, their logic capacity and switching frequency tend to increase while transistor sizes tend to decrease. If the other parameters remained the same, the dynamic power would tend to increase in the newest FPGAs. However, the Virtex-5 and Virtex-6 families of FPGAs have improvements to reduce the dynamic power dissipation over previous FPGA generations. The core voltage and node capacitance play a critical factor in the dynamic power dissipation, therefore the new FPGA families has benefited from reduction in both parameters. The reduction in node capacitance comes from decreased parasitic capacitances associated with the smaller transistors. Nevertheless, these smaller transistors tend to increase static power consumption since the leakage current across the transistor increases. Table 8 shows the core voltage and node capacitance values for both 90nm Virtex-4 FPGAs and 65 nm Virtex-5 FPGAs, leading to an overall dynamic power reduction of 40%.

Table 8: Core Dynamic Power Reduction from Voltage and Node Capacitance Reduction [Curd 07]

	Virtex-4 FPGAs 90nm	Virtex-5/6 FPGAs 65 nm	Percent Change
V_{ccINT}	1.2 V	1.0 V	-16.6%
C_{Total}	1.0nF	0.85nF	-15%
Power	1.44	0.85	-40%

The third type of operational FPGA power consumption is interface power, defined as the power consumed by the I/O interfaces. The interface power follows the same basic dynamic power equation as the internal transistors but the voltage supply, the drive strengths and the terminations are determined by the I/O standards being implemented. While the static and dynamic power dissipation have changed from 90 nm FPGAs to 65 nm FPGAs, the interface power typically remains relatively constant due to the dependence on the I/O standards.

While the FPGA operational power is typically the main design focus, another power component important to system operation is the FPGA turn-on power. The turn-on power is a function of the in-rush power and the configuration power. SRAM FPGAs have a large transient in-rush power requirement during the initial turn-on and configuration. The power supply feeding

the FPGA must accommodate the required in-rush and configuration power requirement to properly power-up the FPGA, often requiring hundreds of milliseconds. Insufficient power at turn-on can prevent the system from powering up properly and cause system failure.

6.6 Designing for FPGA Power Considerations

Power considerations are a significant portion of the design process for FPGA-based software radio systems. As users desire more and more mobile applications, low-power designs must be implemented for to increase battery life. However, when using FPGAs in software radio systems, the programmability and flexibility of the devices make them less power-efficient than custom ASIC logic circuits. FPGAs require significant configuration circuitry for defining logic circuits, utilizing longer routing lengths and higher interconnect capacitances than would be found in an ASIC. Since the interconnect routing consumes additional silicon area on the chip, the static power consumption is increased. These interconnect routing resources do not provide any additional computational efficiency; therefore their reduction is a principal target for FPGA power optimization.

Based on the previously discussed dynamic power equation, it may seem apparent that the lowest power implementation would be the design that uses the lowest dynamic power. When designing for low dynamic power, the hardware blocks are utilized with the lowest possible clock speed and organized in a flat implementation where each block performs a specific function. While this design methodology will provide the lowest dynamic power consumption, it will require significant static power due to the larger number of routing resources and logic. As a design trade-off, a smaller FPGA device with fewer resources can be used where the resources are reused for different algorithms; thereby using lower static power consumption at the expense of increased dynamic power consumption. This method of hardware reuse is generally referred to as a time-division multiplexing (TDM) operation, where the same hardware is used for different operations in specified cyclic time slots. TDM operation requires a higher clock rate than a flat implementation. However, the amount of static power saved using the smaller FPGA device can often more than offset the increased dynamic power used [Pelt 05].

Chapter 7

Digital Hardware Communication

7.1 Digital Hardware Communication Issues

Digital hardware communication is an important design concern that has an impact on the performance and power consumption of a system. As software radio applications develop, mobile communications are becoming increasingly integrated with high performance applications for video, voice and data. Compared to the previous generations of wireless applications, the processing requirements for these new high performance applications are growing exponentially. New wireless standards for 3G and 4G systems can support these applications with higher bandwidths but require a powerful RF baseband processor to support the high-speed data throughput. Additionally, significant processing resources are needed for operations such as decoding video data received by the baseband processor. With the demand for these applications increasing, system designers must be able to pass data among multiple independent processing resources to facilitate application capabilities, while limiting the required transfer power. Multiple processing resources can be designed using both off-chip configurations and on-chip architectures, but with requirements for short time-to-market and low power operation; the trend toward on-chip multi-processor approaches is growing.

As both off-chip and on-chip multi-processor architectures become more common in software radio systems, the inter-processor communication (IPC) among processors becomes a critical design factor and is often a large bottleneck in the system. A common system architecture for software radio is to have a baseband DSP processor connected to a general purpose application processor, as shown in Figure 36. The baseband DSP processor provides the physical layer implementation for the wireless link while the application processor implements the multimedia content and I/O interfaces. In 2G and 2.5G systems, the wireless communications link provides only low data rates, up to 1 Mbit/s, therefore the IPC can be implemented with serial interfaces such as Universal Asynchronous Receiver/Transmitter (UART) or Serial Peripheral Interface (SPI) [Tseng 06]. However, as newer 3G and 4G systems become available, the RF link provides much higher data rates. To process demodulated data, the baseband processor must provide a sufficient data rate from the baseband processor to the application processor. Traditional serial interfaces are not sufficient at the required higher data rates needed for multimedia applications. Therefore, a new method for IPC implementation is required.

7.2 Inter-processor Communication Approaches (Buses)

Inter-processor communication (IPC) has long been an issue for system designers as various issues such as power consumption, protocol complexity, and speed provide design challenges. Dedicated serial interfaces are one method of off-chip IPC, typically implemented using serial peripheral interface (SPI) or universal asynchronous receiver/transmitter (UART) protocols and requiring general-purpose input/output (GPIO) package pins connected to dedicated PCB traces [Tseng 06]. These serial interfaces utilize software control for processor handshaking and data transfer. Since this software control is typically done in the main processor core, system performance can be reduced. Some processors do contain dedicated serial interface hardware that can be utilized; however, this may not be available on both processors and will most likely still require some software control. Serial interfaces also provide power consumption issues because the buses are often required to remain active at all times, regardless of usage. In mobile communications, power consumption becomes a considerable design constraint, so IPC power usage can present an issue. Lastly, some legacy serial protocols don't provide the data throughput necessary for high-speed IPC; however, with USB 2.0 theoretically offering 480 Mbits/s and the newest USB 3.0 theoretically offering 5 Gbits/s, serial protocols are becoming more viable options. Nevertheless, the communication overhead of the protocols reduces the raw data rate.

An alternative IPC solution is a dual-port memory approach, such as that shown in Figure 36. A dual-port memory solution utilizes a memory buffer, typically a dedicated chip, which is connected to both processors, providing a fast and efficient IPC implementation [Tseng 06]. Processor architectures include at least one external memory interface (EMIF) bus that can be used on both processors to connect to the dual-port memory. This conserves GPIO pins for other uses and since standard memory addressing can be used across the EMIF bus, no separate IPC software drivers are needed, simplifying software programming. Additionally, power consumption is reduced in a memory buffer implementation because the buses on each processor are not required to remain active at all times. After a transfer has occurred, the processor can enter a low-power sleep mode until the next transfer. Similarly, the receiving processor can remain in a sleep mode until an external interrupt indicates that it should begin reading data from the memory [Tseng 06]. Nevertheless, the dual-port IPC solution can be a limiting factor in some designs that require additional off-chip memory since it consumes one of the available EMIF buses on both processors.

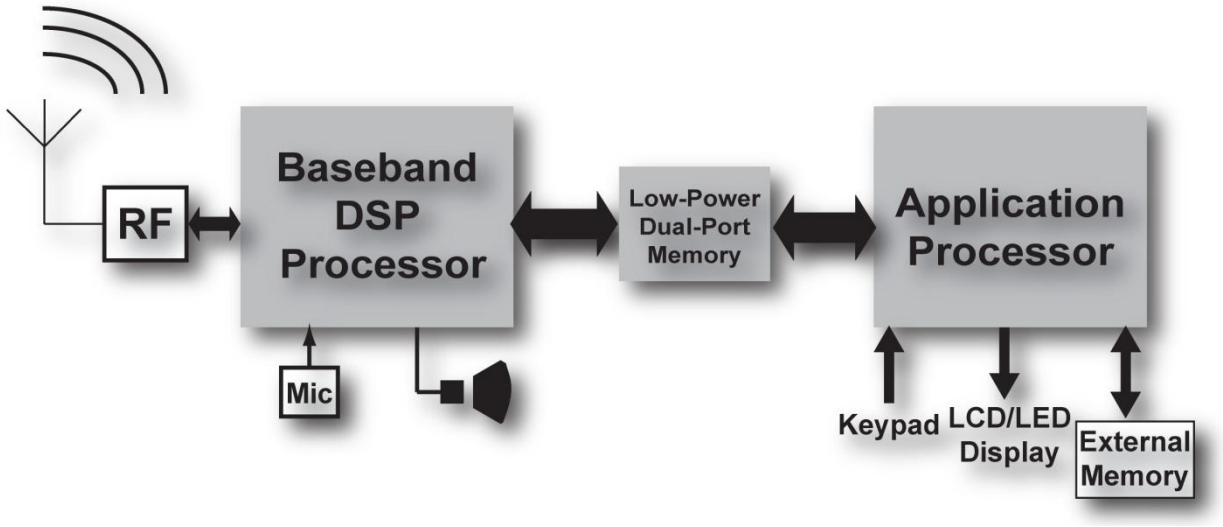

Figure 36: Dual-Port Memory Connection for Multiple Processors

While dual-memory inter-processor communication simplifies the system architecture, it also provides the high-speed communication needed to support current and emerging wireless standards. With a dual-port memory, memory access times can reach 40 ns, providing a maximum of 400 Mbits/s. While this is a maximum value and the actual data rates will also depend on the capabilities of the EMIF bus, it does provide sufficient bandwidth for 3G and 4G wireless standards [Chong 08]. Figure 37 indicates the data rates required for several wireless technology standards and the supported bandwidth of the IPC interfaces. Notice that there is a large gap between legacy interface standards and the newest wireless technology generations. This graph is concerned with the raw data rates needed between a baseband processor and an applications processor. Even higher data rates may be required for other interfaces, such as between the analog-to-digital converter and the baseband processor, since wireless standards often make use of higher sampling and chip rates. It is shown that the USB 2.0 and USB 3.0 standards can support the required data rates but these interfaces suffer from high power consumption requirements. Additionally, in practice, the data rates are often significantly less than their theoretical maximums.

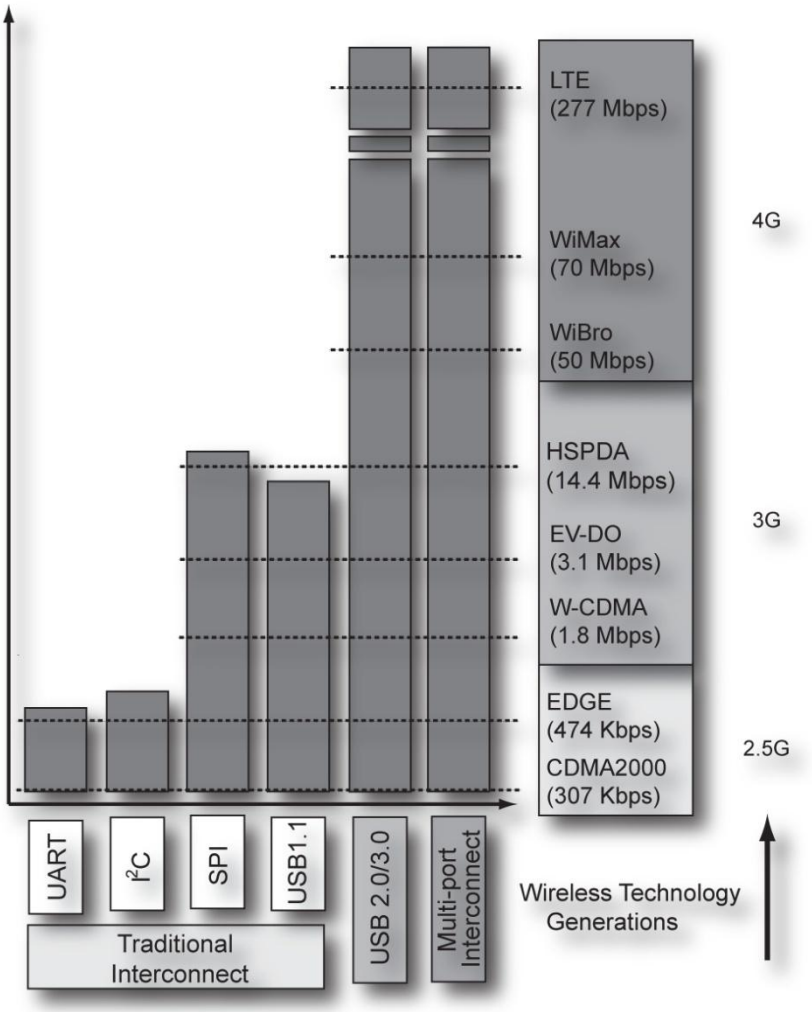

Figure 37: IPC Support Bandwidth and Corresponding Wireless Standard Data Rates[16]

When transferring data between processors using a dual-port memory, data synchronization becomes important. The processors must know when data is available from the other processor for processing. Interrupt signals are typically used for indicating to the co-processor that data is valid and ready to be processed.

7.3 On-chip Communication Networks

With new multi-core processor architectures emerging, inter-processor communication is no longer only a printed circuit board-level issue, it is also becoming a silicon-level issue. In multi-core processor architectures such as the picoArray or the Tile64, each processor must be able to

[16] Figure 37 is adapted from Figure 2, "Low-Power, dual-port inter-processor communication problem in next-generation handsets"

communicate with other on-chip processors. However, traditional bus methods and even direct-shared memory interfaces do not scale well above a dozen or so cores, causing bus contention and poor performance [Wentzlaff 07]. Therefore new IPC methods have been proposed and developed using networks-on-chip for communication among processors.

Networks-on-chips operate in much the same way as large-scale networks, such as the Internet; where individual processor nodes connect to other processor nodes through the use of switches. However, on-chip network architectures and protocols vary among vendors and can utilize either circuit-switched methods or packet-switched methods. These on-chip networks are generally designed in a two-dimensional mesh architecture, where switches are available at each wire crossing, allowing information to flow in all available directions. The switches can either be embedded into the processor itself, or be implemented as separate elements in between processors.

In general, keeping data communication on chip will be faster than off-chip communication. As discussed earlier, the picoArray utilizes on-chip processing elements linked together through an interconnection matrix known as the picoBus. The picoBus is a network on-chip architecture with 32-bit unidirectional buses connected to bus switches for communication among its hundreds of processors. The network switches operate in a time division multiplexing (TDM) method, providing dedicated time slots for transferring data on a single network with two unidirectional buses [Duller 03]. The time slots are assigned during compile time based on the required signaling rate, therefore no run-time bus arbitration is encountered. Figure 38 shows two time slot configurations during different time instants where the switches have been set to form separate signal paths between individual processors.

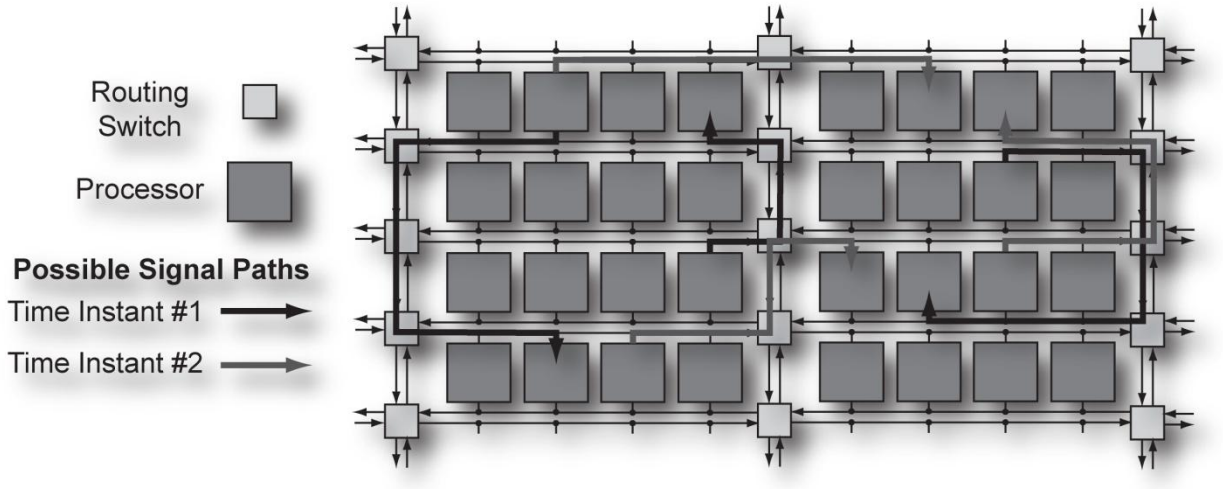

Figure 38: picoArray Interconnect Subset during a TDM Time Slot

Other on-chip network architectures have also been developed for multi-processor chips. The Tile64 processor contains a matrix of five separate 2D mesh networks, where each network has two 32-bit unidirectional links, shown in Figure 39. Rather than having switches between processors, the switches for these networks are contained in the processing units themselves; therefore data can flow in five directions, north, south, east, west, and to the processor. Four of the on-chip networks operate in a packet-switching format, where packets are sent with *x* and *y* coordinates to indicate the destination processor. Packets also contain the packet length information to indicate the size of the data being sent. These packets cannot be sent directly from source to destination unless the destination is the nearest neighbor; therefore packets must hop through processor switches along the way. Latency is generally on the order of one or two clock cycles for each hop, and data is flow-controlled to guarantee reliable delivery [Wentzlaff 07].

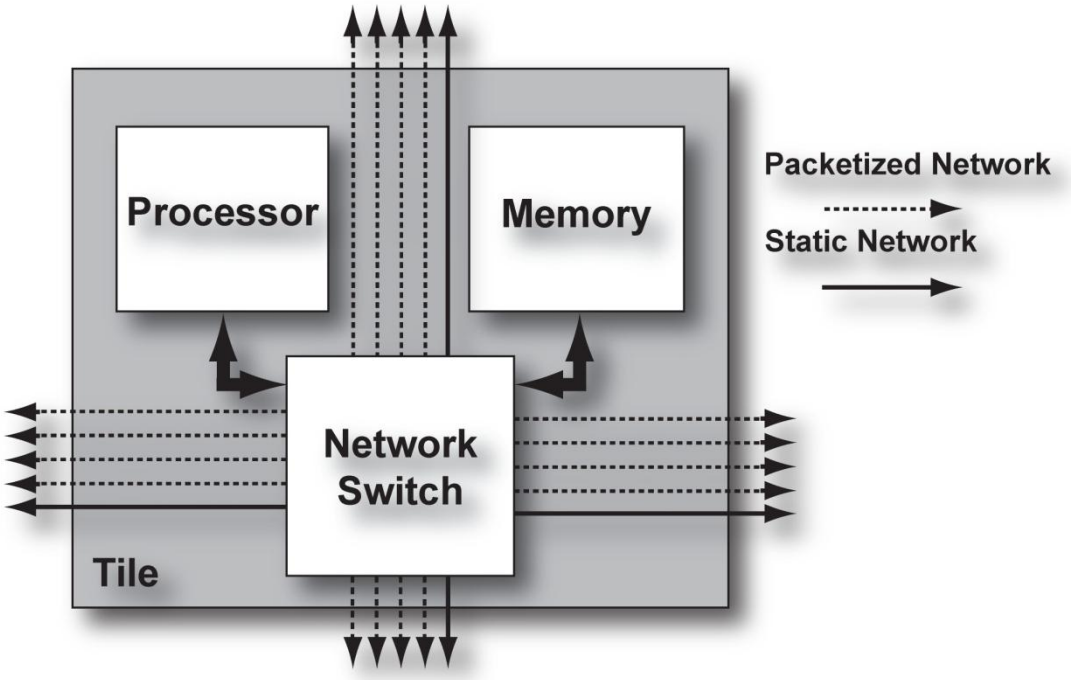

Figure 39: Tile64 On-chip IPC Packetized and Static Networks (Full Chip includes 64 Tiles)

Application requirements often dictate what communication resources are needed. Each of the on-chip buses on the Tile64 performs dedicated functions including memory accesses, I/O communications, and streaming data. This improves application performance by optimizing a network for a specific function. On-chip networks can also operate using a circuit-switched method. The Tile64's fifth bus operates in this manner and is useful for streaming data among processors. Rather than routing packets from one switch to the next, the on-chip circuit switched

network can set up dedicated routes based on the application requirements [Wentzlaff 07]. When requirements change, a new route is configured. As application requirements increase and wireless standards develop, inter-processor communication, both on-chip and off-chip, is going to be a major design concern.

Chapter 8

Testing Issues

8.1 Methods of Testing

The high density programmable digital hardware that has been discussed offers numerous benefits to software radio designs, however, it also adds complexity since software and logical errors can create numerous points of failure in a system. Therefore, testing and validation is an important phase of the development process. Testing is a process of technical investigation that increases the level confidence that the product with behave according to the specifications [Gonzalez 07]. The test and validation of any product constructed from a collection of components can be decomposed into three goals [IEEE 01]:

1) To confirm that each component performs it required function;
2) To confirm that the components are interconnected in the correct manner, and
3) To confirm that the components in the product interact correctly and the product performs its intended function.

This approach can be applied to a number of systems including integrated circuits constructed from simpler functional blocks, the code running on the digital hardware, and the overall system constructed from printed circuit boards (PCBs) and attached devices.

As hardware architectures evolve, the higher circuit density and complexity presents challenges for testing and validation. The latest PCB designs can contain thousands of electrical nodes and 15+ layers. Additionally, most devices have a ball-grid array (BGA) packaging where the electrical contacts are hidden underneath the package, making it difficult to probe individual signals on the device. In a traditional testing approach, physical probes were used to measure the response of a Device-Under-Test using test-pads designed onto the PCB. However, with higher speed signals and increased circuit density, it is not possible to provide sufficient test-pads to debug the system. To address this lack of physical access to on-chip signals, the IEEE 1149.1 boundary-scan standard was adopted [Ngo 08] as one method of facilitating functional testing.

The IEEE 1149.1 boundary-scan standard is also know as JTAG (Joint Test Action Group) and it provides a method to probe complex digital circuits with a four or five wire interface known as the Test Access Port (TAP) [Ngo 08]. JTAG uses the boundary-scan technique where signals next to component boundaries can be observed and controlled through boundary-scan register cells placed next to the component pins. The register cells are interconnected to form a shift-register chain

around the border of the design, accessible through serial input, output, clock, and control ports, hence the 4 wire interface [IEEE 01]. The optional fifth wire is an asynchronous reset signal. These boundary-scan registers allow test data to be loaded into components as well as read from the components; therefore, the processing modules on the digital hardware can best observed, tested, and debugged through external stimuli. In addition to helping with the debug procedure, self-test boundary-scans can be performed to provide many system level tests to ensure the integrity of the system during power-up and run-time, such as interconnect and memory tests.

While JTAG is available on many digital hardware architectures including most TI DSP processors and Xilinx FPGAs, other techniques are also available for functional testing. A less intrusive method requiring fewer additional hardware resources is Software-based Self-Test (SBST). SBST executes a test program loaded in internal memory that provides results on output ports or memory variables. Additionally, in FPGAs, tools such as Xilinx Chipscope Pro allow viewing of internal FPGA signals by inserting a logic analyzer block into the design. The logic analysis block records specified signals during run-time. After the code has been run, Chipscope Pro transfers the stored data to a host computer via the JTAG TAP interface, thereby allowing the developer to see internal signals.

8.2 Code Testing and Validation

The testing mechanisms described in the previous section are used to verify the board-level and IC-level interconnect circuitry as well as the configurable designs running on the digital hardware. Since the code that is written for software and HDL designs is complex, developers split the operations into independent modules that can be designed alone but assembled together to create a system [Gonzalez 07]. Each of these modules requires continuous verification and validation during the design process to ensure quality assurance and a reliable system [Gonzalez 07]. Also, government regulations limit operating parameters such as spectrum utilization, therefore verification and validation must be performed to ensure the software radio performs within legal specifications. Nevertheless, testing can be a significant challenge for programmers. Factors such as hardware flaws, software bugs, and design miscalculations cause errors and must be caught at the stages of the development process.

Validation and verification can be performed by a variety of methods including peer reviews or mathematical proofs but the most commonly used method is testing. Testing is performed on many programmable units during development including individual modules, integrated modules, and the overall system. However, as mentioned previously, the complexity of software radio

waveforms for digital hardware is so vast that is impossible to perform exhaustive tests for every situation; therefore testing cannot guarantee correct operation. Additionally, testing can often consume a large percentage of development resources, but since it is considered critical to the success of the product it should begin early in the design process to avoid expensive consequences.

Testing requires comparison between a known set of test vectors and the actual digital hardware implementation. These test vectors are created in a simulation environment, such as MATLAB. To perform the software testing, the test vectors are loaded in the development environment and a comparison oracle is used to generate test results. The comparison oracle performs the operations in a reference environment and the module under test, creating a result based on the comparison of the two outputs. One of the challenges for the comparison oracle is there is not always a strict pass/fail criterion for software radio modules. The outputs may have a range of values over which the result is considered correct. Therefore, the oracle must be able to make the comparison over thousands of tests and determine whether the module works within acceptable parameters [Gonzalez 07].

During the development process, several key factors need to be addressed when testing modular software components. Designers make assumptions about the operating context when programming a software component, thereby creating a context dependent implementation. Operating condition assumptions are also made when designing the test cases and simulations for comparison. However, these assumptions that are made during development may not be the actual operating conditions under which the component is used in practice. Therefore, it is important to test the software under as many deployable operating conditions as possible. For highly reliable systems, these tests can require extreme and unusual conditions, often difficult to generate in a development environment. For example, to test the receive path of a software radio, a highly flexible transmitter with control over the transmitted signal properties is needed for injecting errors and simulating variable environments [Gonzalez 07]. As an alternative to generating these rare signal environments, the developers can benefit from being able to send artificial signal data into the system through the JTAG TAP interface; helping reduce testing equipment needs and cost.

Since software radio modules are developed independently, they must be integrated together and interfaced with other modules to create a waveform implementation. Integration testing is a continuation of module testing and is important for assuring the quality of the system. In both software and hardware implementations, most faults occur during module interfacing, revealing issues that were not apparent in module testing. Testing the module interfaces is a challenging task because of the large number of modules that interact with each other during

operation. Faults occur for a variety of reasons including inconsistent interpretation of specifications between different programmers, incorrect assumptions made by programmers, unanticipated timing errors, and limited resources where two modules attempt to utilize the same hardware resource. Missing or misunderstood functionality and performance problems can also be a factor [Gonzalez 07]. It is a challenge to solve these issues when designing modules that can be reused across multiple software radios platforms with different digital hardware architectures.

Software radios usually constrained in terms of power, memory, size, and weight; therefore, to provide a powerful yet efficient implementation, designers have to create optimized code to achieve maximum performance [Gonzalez 07]. However, optimized code is difficult to test and validate because platform specific features are often utilized and debugging options are stripped from the code. Some software test packages offer processor pipeline examination at specific break points as well as profiling to determine amount of time spent on sections of code to help in the optimization process. The more granularity that is available in the testing environment, the more insight the programmer has into what operations are being performed; thereby yielding a more reliable product.

Chapter 9

Conclusions and Future Projections

9.1 Conclusions

The advancements in digital hardware processing capabilities are strongly linked to Moore's law where over the past several decades the number of available transistors on an integrated circuit has doubled about every 18 months. These increased processing capabilities are driving more focus on software radios for the future of wireless communications. This thesis has presented many of the processing solutions available for software radio systems, offering an overview of the architectures, design methodologies, and challenges associated with GPPs, DSP processors, FPGAs, and ASICs.

With the variety of processing options available, it is often difficult for radio developers to determine which processing architecture will provide the best solution for a given application. Since several factors are involved including performance, programmability, power consumption, and cost, developers need to balance the trade-offs to work within the system constraints while also offering sufficient capabilities. GPPs, DSP processors, FPGAs, and ASICs each have strengths in a subset of the design factors, but the most efficient solution is not often found using single architecture. Therefore, to maximize trade-offs, developers often turn to a heterogeneous processing approach using different processing architectures integrated on a printed circuit board or substrate. Advanced development tools are helping to provide designers with increased insight into the trade-offs and are helping to facilitate efficient system development.

Evaluating the capabilities of the digital hardware for performance, programmability, and power consumption is complicated and requires analysis of several features, some of which include the chip architecture, the system clock rate, the advanced power control functions, and the processor communication buses. Advancements in chip architectures have led to significant improvements in computational performance, especially with the emerging parallel processing trend. In processor-based architectures, several memory buses are included to fetch multiple operands in a single clock cycle and these operands can be fed to several parallel processing units for concurrent execution. Additional techniques are available to further improve processor performance at the expense of added development time, such as pipelining. If even higher computation performance is desired, an FPGA or ASIC can be used to facilitate high data throughput using multiple processing resources on a single chip.

The combined high clock rates and high parallel computing capabilities result in hardware architectures that can support the latest wireless standards and maintain flexibility, albeit at the expense of high power consumption. Parallel resources require more static and dynamic power dissipation, thereby increasing the power consumption of the radio design which may be necessary to support the application. Nevertheless, the parallel architectures are limited by the buses that connect them, leading vendors to develop new high-speed interconnect standards such as multi-port memory and on-chip communication networks. Other architectures such as low-power DSP processors offer less parallel computing structures but focus on power efficiency, which is especially important in the mobile wireless market. To help designers program these devices for parallel processing or power efficient implementations, the advanced development tools help partition algorithms across elements and control the power management features. While development tools can sometimes be a limiting factor, vendors are continually improving them to increase the use of their products. As the tools and the architectures develop, there will be increased use of digital hardware in wireless communication systems and system designers will continue to need to evaluate the trade-offs to determine the best solution for their application. Given that the design trade-offs involved can often be subjective, the process of digital hardware design may be considered an *art* as well as engineering.

9.2 Future Projections for Digital Hardware

Wireless communication systems are being seen in an increasing number of applications to enhance the available capabilities of a product. As discussed in this thesis, the trends in wireless systems are moving towards higher data rate applications, thereby pushing hardware vendors to include increased functionality on single chip solutions. As these technologies develop, it is anticipated that digital hardware will be included in an increasing number of designs, thus requiring categories for both low-cost and high-performance solutions. Additionally, these designs will require increased inter-operability with each other, requiring larger flexibility on the part of the digital hardware.

Since Moore's law is continuing to drive the increased computational density on integrated circuits and since on-chip communications offers drastically more data throughput than off-chip communications, several of the functions that were previously implemented on individual integrated circuits are moving into a single digital hardware integrated circuit (IC) package, using system-in-package or multi-chip module configurations. Offering as many capabilities on-chip as possible is beneficial for a number of reasons, including the increased computation efficiency and a

more compact design space. This creates advantages for developing smaller, more power efficient radio designs since many of the required off-chip integrated circuits will be included inside the same IC package and reduce power loss at the chip edge. Additionally, the added on-chip functionality also simplifies the design since added traces, package pins, and communications protocols are eliminated. The added on-chip capabilities are going to facilitate many new applications areas for software radio systems.

As the number of desired applications increases, the hardware performance and flexibility needs are going to grow. Using a software-defined concept, a larger number of standards and applications can be supported on a single hardware solution. For instance, a multi-standard cellular phone is useful in today's global economy since users often travel to areas that require different cellular standards. Utilizing flexible digital hardware, the baseband processing architecture can modify itself depending on the cellular standards required. The flexibility is particularly useful for supporting legacy systems since infrastructure upgrades will occur in stages. Nevertheless, this presents challenges as different wireless standards have different processing requirements, thereby requiring scalable solutions, especially from 3G to 4G systems where a 10-1000x data throughput increase is needed [Woh 07]. 4G systems also have shorter latency requirements where the system must respond to requests within a shorter amount of time, driving up the performance requirements. Nevertheless, while supporting multiple standards in today's environment is very useful, especially for the public safety and military systems, the commercial market is moving towards more unified standards. This may reduce the number of concurrent standards that need to be supported, but digital hardware flexibility can also be utilized for other applications.

Another emerging trend for software radios is the use of cognitive capabilities whereby the radio can change its communication parameters to effectively utilize the available spectrum and avoid interference. While the increased market use of wireless communications facilitates added application capabilities, it also introduces more interference into the environment. By utilizing a cognitive engine running on the digital hardware, the radio can avoid interference from other radios and dynamically alter it spectrum utilization to provide the best radio link. In fields such as public safety communications, cognitive engines can identify what type of communications link is required to a specific network and alter the radio profile to create a communications link.

To date, most communications networks that have been deployed are dedicated to serving fixed hardware devices. For instance, the wireless cellular network strictly services mobile handsets for user speech, data and broadband internet connections for laptops. As technology

develops, communications networks are going to be increasingly interconnected and able to support multiple hardware devices. Several products besides cellular phones and laptops will be able to connect to the network and communicate, such as personal vehicles and household appliances. Wireless handsets will also be able to connect with several of the items in the home, where handsets will inform the lights, television, oven, etc. when the home owner arrives.

These added applications create uses for both low-power, low-cost processing resources and high-performance solutions. The larger number of low-cost processing resources will be used in the consumer devices while the increased amount of data arriving at the basestation, as well as the rate at which that data arrives, will drive up the performance requirements of the basestation. Additionally, there is going to be increased use of general-purpose processors for the processing solutions used in software radios. The high backwards compatibility available on successive GPP generations yields a very useful solution for reducing the engineering costs during system upgrades. This development model is already being used in some system architectures by companies such as Vanu Incorporated. Vanu provides software based base station systems running on a general-purpose platform. Since GPPs can run standard operating systems such as Linux, development is open to a wider group of programmers. Therefore, the real-time capabilities of general-purpose operating systems are going to continue to improve to utilize this new group of programmers.

While software radio solutions are becoming more prevalent in many wireless communication products, it will be some time before the technology gains enough market share to begin replacing consumer mobile handsets. In the commercial market where devices are mass produced from a single design, custom solutions currently make more economic sense since dedicated ASICs will be cheaper than flexible digital hardware. It is anticipated that consumer products will progressively incorporate more flexible architectural features but until the cost is reduced to compete with ASIC solutions, consumer market penetration will be limited. Nevertheless, the software radio market is here to stay and promises many exciting challenges for radio developers in the years to come.

References

[Altera 07] Altera, “DSP-FPGA System Partitioning for MIMO-OFDMA Wireless Basestations,” White Papper, Altera Corporation, October 2007.

[Altera 09] Altera, “Digital Signal Processin (DSP) Blocks in Stratix Devices,” Altera Corporation, 2009, *http://www.altera.com/products/devices/stratix-fpgas/stratix/stratix/features/stx-dsp.html*

[Analog 08] Analog Devices, “Blackfin Embedded Processor: ADSP-BF512/ADSP-BF514/ADSP-BF516/ADSP-BF518,” Analog Devices, Inc., 2008

[Analog 09] “ADSP-21xx Family Selection Table,” Analog Devices, Inc., 2009 *http://www.analog.com/en/embedded-processing-dsp/adsp- 21xx/content/adsp_selection_table /fca.html.*

[Arrington 06] Arrington, Ed, et al., "Energy Efficiency in Mobile Devices," DSP DesignLine, August 31 2006, *http://www.dspdesignline.com/192501123;jsessionid=BCADBH5WH50KYQSNDLPSKHSCJUNN2JVN?pgno=1.*

[BDTI 00] BDTI, “Choosing a DSP Processor,” Berkeley, CA: Berkeley Design Technology, Inc., 2000.

[BDTI 02] BDTI, “Evaluating DSP Processor Performance,” Berkely, CA, Berkely Design Technology, Inc., 2002.

[BDTI 07] BDTI, "Massively Parallel Processors for DSP, Part 1." 18 June 2007. DSP DesignLine. 9 March 2009 *http://www.embedded.com/design/embeddeddsp/201400317?pgno=2*.

[BDTI 08] BDTI, “BDTI Releases Benchmark Results for Tilera’s TILE64 Multicore Processor,” *InsideDSP,* September 17 2008, *http://www.insidedsp.com/tabid/64/articleType/ArticleView/articleId/276/Default.aspx*

[Becker 07] Becker, Tobias, Wayne Luk and Peter Cheung., "Enhancing Relocatability of Partial Bitstreams for Run-time Reconfiguration." *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines,* pp. 35-44. 2007

[Blume 02] Blume, H., Hübert, H., Feldkämper, T., Noll, G., “Model-based Exploration of the Design Space for Heterogeneous Systems on Chip,” *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP ’02),* pp 29-40, 2002.

[Chong 08] Chong, Ming Hoong., "Optimize Inter-processor Communication in Dual Baseband Dual Mode Handsets." Mobile Handset DesignLine. Mar. 3 2008. *http://www.mobilehandsetdesignline.com/206901257?printableArticle=true*.

[Cofer 05] Cofer, RC., Harding, Ben, “Implementing DSP Functions with FPGAs,” Programmable Logic DesignLine, September 7, 2005. *http://www.embedded.com/columns/technicalinsights/170702040?pgno=1*

[Cofer 06] Cofer, RC and Ben Harding, "Fixed-Point DSP and Algorithm Implementation." DSP DesignLine. Oct. 25 2006 *http://www.dspdesignline.com/howto/193402220;jsessionid=34I2A2PSJZ25UQSNDLPSKHSCJUNN2JVN?pgno=1*.

[Cosoroaba 07] Cosoroaba, Adrian and Frédéric Rivoallon, "Achieving Higher System Performance with Virtex-5 Family of FPGAs." 2007.

[Curd 07] Curd, Derek, "Power Consumption in 65 nm FPGAs (WP246)." Xilinx Inc., Feb. 1 2007.

[Dejonge 08] Dejonghe, A., Craninckx, J., Provoost, J., Van der Perre., L., "Reconfigurable Radios, part 1: SDR architectures," *DSP DesignLine*, March 31 2008, *http://www.dspdesignline.com/howto/207000126*

[Dick 02] Dick, Chris, "A Case for Using FPGAs in SDR PHY." *EE Time,* Aug. 9 2002. *http://www.eetimes.com/story/OEG20020809S0049*.

[Donovan 06] Donovan, John, "*High-Performance DSPs for Portable Applications,"* Tulsa: PennWell Corporation, 2006.

[Duller 03] Duller, Andrew, Gajinder Panesar and Daniel Towner, "Parallel Processing - the picoChip way!" *Communicating Process Architecture - 2003* pp. 299 - 312. 2003

[Edwards 06] Edwards, Stephen A., "The Challenges of Synthesizing Hardware from C-like Languages." *Design & Test of Computers, IEEE,* pp. 375-386. 2006

[Eyre 00] Eyre, Jennifer and Jeff Bier, "The Evolution of DSP Processors," Berkeley, CA: Berkeley Design Technology, Inc., 2000.

[Francis 01] Francis, Hedley. "ARM DSP-Enhanced Extensions." ARM, Inc., 2001.

[Freescale 07] Freescale Semiconductor, "MPC7448 RISC Microprocessor Hardware Specifications," Freescale Semiconductor, Inc., March 2007.

[Gonzalez 07] Gonzalez, Carlos R., Reed, J., "Validation and Verification of Modular Software for Software-defined Radios," SDR Forum Technical Conference 2007, Vol A. pp. 151-155, 2007

[Haessig 05] Haessig, David, et al. "Case-study of a Xilinx System Generator Design Flow for Rapid Development of SDR Waveforms." *SDR 05 Technical Conference and Product Exposition,* 2005

[Hauck 98] Hauck, Scott, "The Roles of FPGA's in Reprogrammable Systems." *Proceedings of the IEEE, vol. 86,* pp. 615-638, April 1998

[Hori 07] Hori, Bjorn, Jeff Bier and Jennifer Eyre, "Use a Microprocessor, a DSP, or Both?" *TechOnline.,* April 5, 2007 *http://www.techonline.com/learning/techpaper/199202284?pop_up=http%3A//www.techonline.com/article/pdf/showPDF.jhtml%3Fid%3D1992022841&_requestid=192451.*

[Hosking 08] Hosking, Rodger, "New FPGAs Revolutionize Digital Down Converters." *SDR '08 Techincal Conference and Product Exposition*. 2008

[IEEE 90] IEEE, “IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries,” The Institute of Electrical and Electronics Engineers, New York, NY, 1990

[IEEE 01] Test Technology Standards Committee, “IEEE Standard Test Access Port and Boundary-Scan Architecture,” IEEE Std. 1149.1-2001(R2008), IEEE Computer Society, 2001.

[Kalinsky 05] Kalinsky, David, "Why use an RTOS on DSPs?" Embedded-Europe.com, June 2005, *http://www.esemagazine.com/index.php?option=com_content&task=view&id=4&Itemid=2.*

[Lapsley 97] Lapsley, Phil, Beir, J., Shoham, Amit., Lee, Edward., “DSP Processor Fundamentals: Architectures and Features,” New York: IEEE Press, 1997.

[Lerner 07] Lerner, Boris, "Fixed vs. Floating Point: A Surprisingly Hard Choice." DSP DesignLine. 6 Feb 2007. *http://www.dspdesignline.com/showArticle.jhtml?articleID=197002280.*

[Liu 98] Liu, K.J Ray, Raghupathy, A., “Algorithm-based low-power and high-performance multimedia signal processing,” *Procedings of the IEEE, vol 86, issue 6*. pp. 1155- 1202, June 1998

[Lo 08] Lo, Haw-Jing, Yoo, H., Anderson, D.V., “A reusable distributed arithmetic architecture for FIR filtering,” *Circuits and Systems 2008, MWSCAS 2008. 51st Midwest Symposium,* pp. 233 – 236, August 2008.

[Longa 06] Longa, Patrick, Miri, A., “Area-Efficient FIR Filter Design on FPGAs using Distributed Arithmetic,” *Signal Processing and Information Technology, 2006 IEEE International Symposium,* pp. 248 – 252, August 2006.

[Mathworks 08] Mathworks, “About Model-Based Design,” *http://www.mathworks.com/applications/dsp_comm/description/mbd.html.*

[Musah 08] Musah, Arthur and Andy Dykstra, "Dynamic Power Management Techniques for Multimedia Processors." Power Management DesignLine. July 27 2008, *http://www.powermanagementdesignline.com/showArticle.jhtml;jsessionid=BCADBH5WH50KYQSNDLPSKHSCJUNN2JVN?articleID=209601368&queryText=Dynamic+power+management.*

[Neel 05] Neel, James, et al., "PCET: A Tool For Rapidly Estimating Statistics of Waveform Components Implemented on Digital Signal Processors." *SDR '08 Technical Conference and Product Exposition,* 2008.

[Neel 05] Neel, James, Pablo Robert and Reed, J., "A Formal Methodology for Estimating the Feasible Processor Solution Space for a Software Radio." *SDR 05 Technical Conference and Product Exposition,* pp. A117-A122, 2005

[Nelson 95] Nelson, Victor, et al., “Digital Logic Circuit Analysis and Design,” Upper Saddle River, New Jersey: Prentice-Hall, Inc., 1995.

[Ngo 08] Ngo, Be Van, Law, P., Sparks, A. “Use of JTAG Boundary-Scan for Testing Electronic Circuit Boards and Systems,” 2008 IEEE AutoTestCon, pp. 17-22, Sept 8-11 2008.

[NVIDIA 09] NVIDIA, “What is GPU Computing?,” NVIDIA Corp, 2009, *http://www.nvidia.com/object/GPU_Computing.html*

[Oshana 07] Oshana, Robert, "Real-Time Operating Systems for DSP." Embedded.com. April 19 2007, *http://www.embedded.com/columns/technicalinsights/199101385;jsessionid=ZPLIFVP2AXZTOQSNDLPSKH0CJUNN2JVN?pgno=1.*

[Owens 08] Owens, John, Houston, M., Luebke, D., Green., Stone, J., Phillips, J., “GPU Computing: Graphics Processing Units—powerful, programmable, and highly parallel—are increasingly targeting general-purpose computing applications,” *Proceedings of the IEEE,* Vol. 96, No. 5, pp. 879-899, May 2008

[Patterson 06] Patterson, Jim and John Dixon. “Optimizing Power Consumption in DSP Designs,” Dallas, Texax Instruments, Inc., 2006.

[Pelt 05] Pelt, Rob and Martin Lee. "Low Power Software Defined Radio Design Using FPGAs." *SDR '05 Technical Conference*, pp. A107-A110. 2005.

[Pulley 08] Pulley, Doug., “Multi-core DSP for Base stations: Large and Small,” *Design Automation Conference 2008, ASPDAC 2008. Asia and South Pacific.,* pp. 389-391, March 2008

[Raje 04] Raje, Salil. "Solving Timing Closure in High-End FPGAs." 8 COTS Journal. August 2004, *http://www.cotsjournalonline.com/home/article.php?id=100162*.

[Reed 02] Reed, Jeffrey H. “Software Radio: A Modern Approach to Radio Engineering,” Prentice Hall, 2002.

[Rivoallon 07] Rivoallon, Frédéric. "Physical Synthesis Flows for FPGA Designs." *FPGA and Structured ASIC Journal.* November 11, 2007, *http://www.fpgajournal.com/articles_2007/20071120_xilinx.htm*.

[RTXC 07] “RTXC Quadros Overview,” 2007. *http://www.quadros.com/products/operating-systems/rtxc-quadros/rtxc-quadros-overview/.*

[Rudra 04] Rudra, Angsuman. "FPGA-based Applications for Software Radio," RF Design, pp. 24-35, May 2004

[Santarini 08] Santarini, Mike. "FPGAs: Primed for a Prominent Role in the 4G Wireless Network," *Xcell Journal*, pp. 9-13, 2008

[Scott 06] Scott, Troy. "Tackle team-based FPGA Design," Embedded Systems Design, March 17 2006. *http://www.embedded.com/columns/technicalinsights/180206328?_requestid=11826*.

[TI 08a] Texas Instruments. "TMS320C6424 Fixed-Point Digital Signal Processor Datasheet," Texas Instruments, June 2008.

[TI 08b] Texas Instruments, “Operating Systems (OS/RTOS),” 2008. *http://focus.ti.com/dsp/docs/dspsupportatn.tsp?sectionId=3&tabId=477&familyId=44&toolTypeId=5#biosovw*.

[TI 08c] Texas Instruments, “TMS320VC5502Fixed-PointDigitalSignalProcessor: Data Manual,” Texas Instruments, Inc., April 2001, Revised November 2008

[TI 09] Texas Instruments, “Platform Processors,” http://focus.ti.com/paramsearch/docs/parametricsearch.tsp?family=dsp&sectionId=2&tabId=25&familyId=44

[Tseng 06] Tseng, Danny and Lawrence Wong. "Low-power, Dual-port Solves Inter-processor Communication problem in Next-generation Handset," Embedded.com, May 15 2006. *http://www.embedded.com/columns/technicalinsights/187203124?_requestid=274702*.

[Verma 07] Verma, Seyi, Paul McHardy and Alexander Grbic. “Stratix III FPGAs vs. Xilinx Virtex-5 Devices: Architecture and Performance Comparision,” San Jose, CA: Altera, 2007.

[Wang 00] Wang, Alice, Rahmi Hezar and Wanda Gass. "A Low Power Implementation of a W-CDMA Receiver on an Ultra Low Power DSP," *IEEE Global Communications Conference,*pp 241-244. 2000.

[Wentzlaff 07] Wentzlaff, David, et al. "On-chip Interconnection Architecture of the Tile Processor." *IEEE Micro*, pp 17-31. Sept-Oct 2007.

[Woh 07] Woh, M., Seo, S., Lee, H., Lin, Y., Mahlke, S., Mudge, T., et al., “The Next Generation Challenge for Software Defined Radio,” In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pp. 343-354. Heidelberg: Springer Berlin, 2007.

[Xilinx 08] Xilinx, "Virtex-5 FPGA Xtreme DSP Design Considerations," Xilinx, Inc., 2008.

[Xilinx 09] Xilinx, “Virtex-6 Family Overview: DS150 (v1.0),” Xilinx, Inc., February 2, 2009.

[Ziedman 06] Ziedman, Bob., “Back to the Basics: All about FPGAs,” *Programmable Logic DesignLine,* March 21 2006.
http://www.embedded.com/columns/technicalinsights/183701900?_requestid=24347