A BASEBAND PROCESSOR FOR SOFTWARE DEFINED RADIO TERMINALS

by

Hyunseok Lee

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering and Computer Science)
in The University of Michigan
2007

Doctoral Committee:

Professor Trevor N. Mudge, Chair
Professor Chaitali Chakrabarti, Arizona State University
Associate Professor Scott Mahlke
Professor Marios C. Papaefthymiou
Professor Wayne E. Stark

ABSTRACT

A BASEBAND PROCESSOR FOR SOFTWARE DEFINED RADIO TERMINALS

by
Hyunseok Lee

Chair: Trevor N. Mudge

Software defined radio (SDR) is a technical effort to use programmable hardware in wireless communication systems so that various protocols can be easily supported by software. However, using programmable hardware for SDR terminals has been unachievable because of their tight power budget and high demand on computation capability. The main theme of this thesis is to design a power efficient programmable baseband processor for the SDR.

This thesis analyzed most contemporary wireless communication protocols both in system and algorithm levels. System level analysis is to see the interactions between algorithms and the algorithm level analysis is to investigate the computation patterns of the algorithms comprising baseband operations.

Based on the characterization results, this thesis proposes chip multiprocessor architecture, whose PEs have both parallel and scalar datapaths. Multiprocessor architecture is proposed to exploit the algorithm level parallelism. Both the parallel and scalar datapaths are used because baseband processing is a combination

of parallelizable and scalar computations. For additional enhancements, three novel schemes are applied to the SIMD style parallel datapath: macro instructions, macro pipelining, and the staggered execution of computation units.

Macro instruction is to combine several primitive instructions into one. It reduces system power by eliminating unnecessary register accesses. The macro pipelining is to form a pipeline by cascading hardware blocks for common macro operations. It enhances system throughput by concurrently executing the macro operations. The staggered execution is to shift the operation timing of computation units of the parallel datapath. It improves system throughput and power efficiency by replacing complex N×N crossbar switches with simple N×1 switches.

The power efficiency of the proposed architecture is evaluated through a Verilog model and commercial tools. The proposed architecture consumes only 150 mW while providing W-CDMA 2Mbps packet data service.

The contributions of this thesis are to analyze the characteristics of baseband operations from the perspective of architecture and to adapt the three novel schemes for system enhancement.

To min, claire, and chloe

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

CHAPTER I

INTRODUCTION

In the near future, we will use intelligent devices that include the functionalities of most hand held devices such as tablet computers, cellular phones, MP3 players, game consoles, etc [1][2]. In order to maximize the efficacy of this future device, it is crucial to provide a seamless wireless connection without any spatial or time limitation. However, in reality, it is not easy to provide such seamless wireless connection because there have been no wireless networks that cover the entire world.

A key technology that enables seamless wireless inter-connectivity is software defined radio (SDR). SDR is a wireless communication system whose function blocks are implemented by flexible software routines instead of fixed hardware, so various wireless protocols can be easily supported on the same platform [3]. An SDR terminal adaptively changes its operation mode according to the type of available wireless network. The concept of the SDR originated in the military, but now it is emerging as important commercial technology. For instance, 4G, the next generation of cellular phone networks, requires that multiple wireless protocols be supported within a single terminal [4].

Although the concept of the SDR is very attractive, there exist many obstacles on the way to its commercialization. High computation capability, tight power budget,

and high degree of programmability are quite difficult goals to achieve simultaneously. The amount of computation required for a wireless terminal to perform baseband signal processing is above several tens giga operations per second (GOPS) level. The power allowed for baseband signal processing must be lower than several hundred mW in order to be used for commercial purposes [5]. Furthermore, the programmability needed for the execution of various wireless protocols tends to undermine the power efficiency of a hardware system.

There have been many architectures for SDR. The SandBluster [6] of SandBridge is a chip multiprocessor system which supports multi-threading in the hardware level and its core has a narrow single instruction multiple data (SIMD) datapath. The PicoArray [7] of PicoChip is a multiple instruction multiple data (MIMD) style processor array. It is only suitable for base stations with loose power constraints because of its high power consumption caused by having individual control logics in each core. There are two architectures based on wide-SIMD datapath, SODA from Michigan [8] and the EVP from Phillips [9]. However, the throughput and power performance of these architectures are not sufficient to meet the requirements of emerging high speed wireless networks, such as WiMax and W-CDMA high speed downlink packet access (HSDPA).

The main topic of this thesis is to design a programmable digital hardware system that can support all the major wireless protocols while satisfying the three requirements of the SDR: high computation capability, low power consumption, and programmability. For this purpose, this thesis fulfils the design of an SDR platform with two underlying guidelines. One is to fully exploit all available parallelism existing in the baseband signal processing in order for both high computation capability and low power consumption. Another is to limit the programmability of the hard-

ware system, such that it minimally satisfies the flexibility requirement, in order to minimize power burden induced by allowing programmability. However, these design guidelines are reasonable only when the workload characteristics of the applications are properly analyzed. Thus, as a first step, this thesis analyzes the characteristics of major signal processing algorithms which comprise the baseband operation of most contemporary wireless communication networks [10][11][12][13][14][15].

Workload characterization was done at two levels, the system and individual algorithms. System level characterization helps high level design decisions such as the granularity of processing elements (PE), number of PEs, memory hierarchy, and interprocess communication mechanism. As a representative system, this thesis models the physical layer of the W-CDMA and derives many design data, such as dominant computation kernels, peak workload, the range of workload variation, and communication pattern between computation kernels. The results from the algorithm level characterization is used to determine the design of the PE. This thesis analyzed the operation of major algorithms that dominate the baseband processing workload, such as Viterbi decoder, Turbo decoder, and modulation/demodulation schemes, of time division multiple access (TDMA), code division multiple access (CDMA), and orthogonal frequency division multiple access (OFDMA) systems.

On top of these workload characterization results, this thesis proposes a coarse gain chip multiprocessor architecture whose one PE contains both a parallel and scalar datapath. All PEs are interconnected by a low speed bus. A chip multiprocessor architecture allows the utilization of the algorithm level parallelism of the baseband operation. Coarse grain PEs minimize power expensive global communication. Thus, low speed buses are sufficient to cover the communication traffic between PEs. Having both parallel and scalar datapaths within each PE allows a power ef-

ficient operation by minimizing the number of data copies for the communication between the parallel and scalar datapath.

For the parallel datapath of a PE, this thesis applies three novel schemes to improve its throughput and power efficiency: **macro instructions**, **macro pipelining**, and **staggered execution of computation units (CU)**. Macro instructions combine several instructions into one. The use of macro instructions is power efficient due to the minimizing of the number of power consuming register file accesses. Macro pipelining cascades macro hardware blocks of data load/store, data alignment, and computations. As we will show in the chapter for the algorithm level workload characterization, major parallelizable computation kernels can be modeled as the combination of data load/store, data alignment, and computations. Thus, macro pipelining allows us to exploit the macro operation level parallelism which was not utilized by previous SDR solutions based on the thread, instruction, and data level parallelism (DLP). Staggered execution shifts the operation timing of computation units in the parallel datapath. Because only one CU needs input data at a time, it is possible to use an $N \times 1$ network for feeding input data to an N wide parallel datapath instead of complex and power consuming $N \times N$ network. These three novel schemes result in good power and throughput performance compared to previous architectures. Experimental results show that the proposed architecture consumes about 150 mW while supporting W-CDMA 2Mbps packet data service.

For component level power information, this thesis builds a hardware model with Verilog and synthesizes the hardware model with Synopsys' Physical Compiler using the TSMC-13 standard cell library which is based on 0.13 micron technology. Additionally, this thesis uses Artisan's memory compiler for the generation of storage components, such as register files and data memory. Synopsys' PrimePower tool is

also used for the power evaluation in the gate level. This thesis also builds a system level power evaluation tool that uses component level power information and executable code as input for system level power evaluation.

The organization of this thesis is as follows. In Chapter II, this thesis discusses preliminary topics, which are required to explain the detail operation of wireless terminals. In Chapter III, the physical layer operations of wireless communication systems are broken down into the algorithmic level. In Chapter IV, this thesis analyzes the characteristics of physical layer operations in the system level. Based on workload analysis results, this thesis proposes an architecture for a baseband processor in Chapter V and VI. In Chapter VII, this thesis shows power and throughput evaluation results. Finally, Chapter VIII concludes this thesis. In Appendix, it is possible to see detail discussion on the characteristics of baseband processing algorithms.

CHAPTER II

PRELIMINARY

Prior to detailed discussion, several preliminary issues are explained in this chapter. The explanation will include the evolution history, types, and protocol stack hierarchies of wireless networks. In addition, a generalized model of physical layer operations is also explained. Understanding of these issues is important to validate the design decisions made on the proposed architecture for SDR application.

2.1 Types of Wireless Communication Networks

According to the required range of network coverage, it is possible to classify wireless communication networks into four types as shown in Figure 2.1 [16]: wireless personal area networks (WPAN), wireless local area networks (WLAN), wireless metro area network (WMAN), and wireless wide area networks (WWAN). The operation procedure of these networks differ greatly because of application specific optimization.

The WPAN enables users to connect to various personal devices over short distances without wires. An application might be synchronizing data between a personal data assistant and a desktop computer. Bluetooth is the most popular system at the moment [17]. Ultra wide band is one emerging technology which is applicable

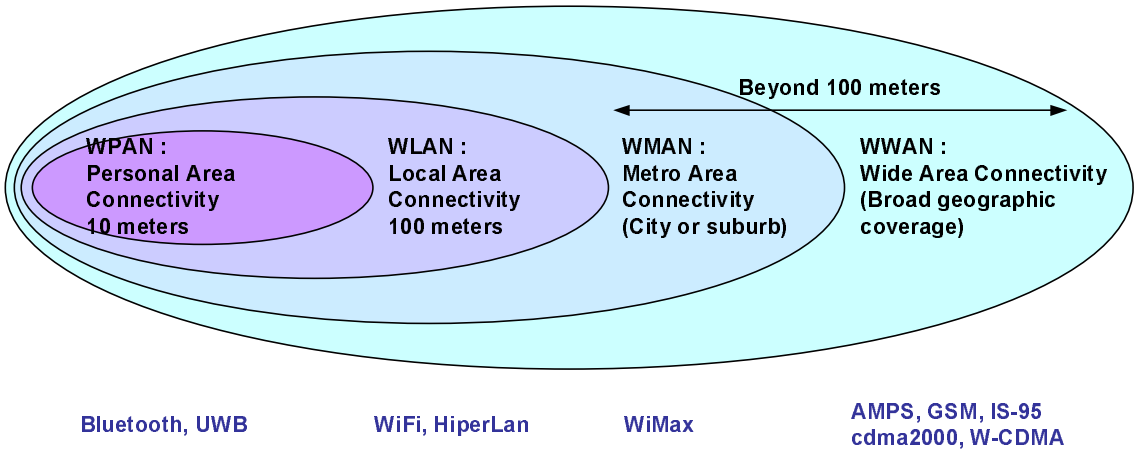


Figure 2.1: Types of wireless communication networks: WPAN, WLAN, WMAN, and WWAN

for the WPAN [18]. One unique characteristic of the WPAN is its simple operation scenario compared to other wireless networks. This characteristic is a result of an effort to minimize production cost. Because the WPAN needs to be built on any kind of consumer electronic device, low production cost is crucial. Because of such a low implementation cost, even single mode wireless terminals have WPAN interfaces as a default feature.

WLAN originated from wired local area network (LAN). It aims to replace existing wired LAN by high speed wireless channels. A typical terminal of this type of wireless networks is a laptop computer having wireless access. IEEE 802.11/a/b/g, usually called as WiFi, are the most popular WLAN systems [19]. HiperLan is the European standard for the WLAN. However, it become a minor standard supported by very limited companies. The characteristics of the WLAN are its high data rate and limited terminal mobility support. The narrow network coverage of the WLAN results in higher data rate (up to 100 Mbps) compared to other types of wireless

networks having wider coverage. Because there are no organizations who manage inter-operability of WLAN access points, the control procedure for mobility support defined in WLAN specification has no significant meaning in practice. Thus, one fundamental shortcoming of WLAN is limited coverage.

In metro areas, WMAN could be optimal. Originally this network was designed to replace roughly one mile range optical cables between end users and network router, last mile problem [20]. Installing optical link to all subscribers is economically inefficient because of the low utilization of optical links. WMAN can provide a similar level of network throughput with lower cost by replacing expensive optical cables with wireless link. IEEE 802.16 is the most common WMAN system. It has been evolved to support even mobile terminals in addition to fixed terminals [15].

WWAN evolved from telephone networks. The early generations of WWAN such as AMPS, GSM, and IS-95 provide voice service. The 3rd generation systems, called 3G, provide multimedia services such as video telephony on wireless channels. The CDMA-2000 [21] and W-CDMA [12] systems are typical examples. One characteristic of the WWAN is the support of relatively lower data rate but better terminal mobility. Usually the WWAN covers an entire country or continent. For complete mobility support with low equipment cost, a cell of this network covers wider area and consequently the allowed maximum data rate is usually lower than other networks.

Because network coverage and available maximum data rate are different, the optimal network varies according to operation environment. In building environments such as an office, the WLAN may be optimal for data service. In rural area, the WWAN may be optimal because of its wide coverage. At the street of metro area, the WMAN can be optimal. As we mentioned before, in the near future, end users will need terminals that can support above protocols to seamless span WPANs,

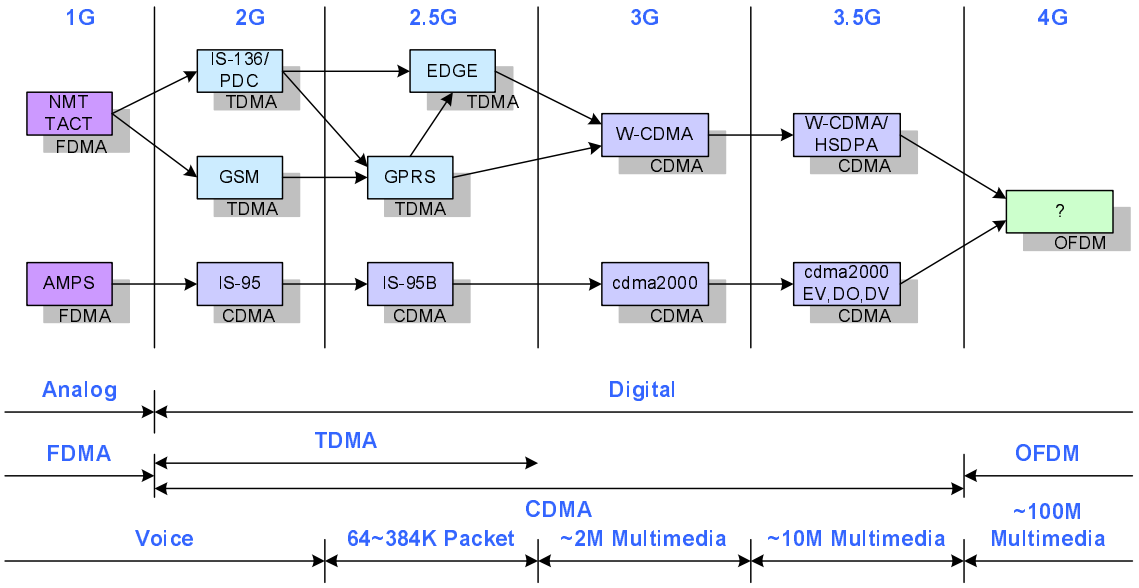


Figure 2.2: Evolution history of WWAN systems

WLANs, WMANs, WWANs, plus their future derivatives. Such a variety of network types lead us to develop a flexible hardware system that can run many wireless protocols.

2.2 Evolution of Wireless Communication Networks

In the previous subsection, we saw that there exist many different types of wireless networks. However, even for the same type of wireless network, there exist different generations because wireless networks continuously evolve in order to provide better service by adapting new technologies. As an example, Figure 2.2 depicts the evolution history of the WWAN.

The first generation of the WWAN provides voice service. The system of this generation relied on analog communication technology and consequently its network

capacity is much lower than that of successors. The second generation WWAN increases its network capacity by adapting more advanced digital communication technologies such as TDMA and CDMA. The goal of the 2.5th generation WWAN was to provide packet data service in addition to voice service. Furthermore, the 3rd generation WWAN targets 2 Mbps multimedia service including voice service, video telephony, and high speed internet access over wireless channel. From this generation, the CDMA technology is adapted for all kinds of WWANs. The 3.5 generation WWANs additionally define 10 Mbps level high speed packet channels on the existing 3G systems. The high speed channels are designed to compete with the WMAN and WLAN that can provide high speed link in building environment. Until now, there is no specification for the 4th generation WWAN. However, it is commonly predicted that OFDMA and multiple input multiple output (MIMO) technologies will be key ingredient of the 4th generation WWAN.

From the evolution history of the WWAN, we can see that, even for a same service such as voice, there exist many different network generations, and within the same network generation, there exist many specifications based on different communication technologies. Such diversity limits the coverage of the WWAN and a similar situation also exists at other type of networks. For example, most European countries use the GSM system but in USA both GSM and IS-95 systems are used for voice service. Thus, the subscriber of IS-95 system in USA can not be served in Europe with the same phone. Therefore, the evolution of wireless network also leads us to develop a flexible wireless terminal that can support all network generations and communication technologies.

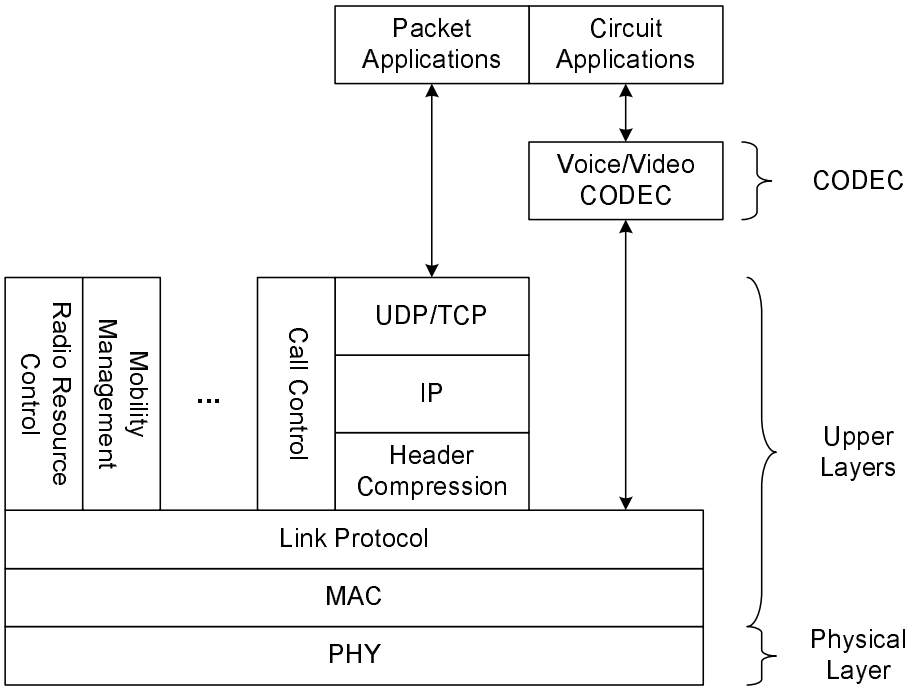


Figure 2.3: A simplified protocol stack of a wireless terminal

2.3 Wireless Protocol Stack

As shown in Figure 2.3, we can divide the protocol stack of a wireless terminal into two categories according to workload characteristics: the physical layer and the upper layers. The operation of the physical layer is related to overcoming unreliable wireless channel characteristics and maximizing the efficiency of expensive wireless spectrum. The physical layer consists of computationally intensive signal processing algorithms.

In contrast, the upper layer protocols consist of control intensive operations. Medium access control (MAC) resolves contention between terminals who share wireless channels. The link protocol performs control actions required to retransmit corrupted frames. In addition to data transmission and reception, many control

actions are performed in the upper layers. Radio resource control dynamically assigns and releases radio resource for terminals according to their operation state. Mobility management performs control actions for terminal handover procedure between basestations. Call control covers call generation and termination procedures.

According to application types, different protocol paths are used. For packet data applications such as web browsing, TCP/IP protocols are used for end to end data transmission over an IP network. In circuit applications, after the link layer, frames are directly forwarded to coder/decoder (CODEC) because no retransmission is allowed in the circuit application due to tight time budget. Between user applications and link protocol, video or voice CODEC can be placed. The function of CODEC is to minimize the amount of transmitted information by removing redundancy and to recover original information. MPEG coding/decoding is an example of the CODEC operation.

2.4 Skeleton of a Wireless Terminal

As shown in Figure 2.4, a wireless terminal can be implemented with four major blocks: analog frontend, digital baseband, protocol processor, and application processor. The physical layer of wireless protocol is mapped on both the analog frontend and digital baseband. The upper layers of wireless protocol are mapped on the protocol processor. Application processor covers the CODECs.

The operation of the analog frontend is to place a baseband signal on the carrier frequency band and vice versa. Because state-of-art digital circuit can not process a GHz level signal with reasonable power consumption, it is common to implement this block with analog ASICs. The digital baseband block performs the remaining physical layer operations. Due to the tight power budget and high computation

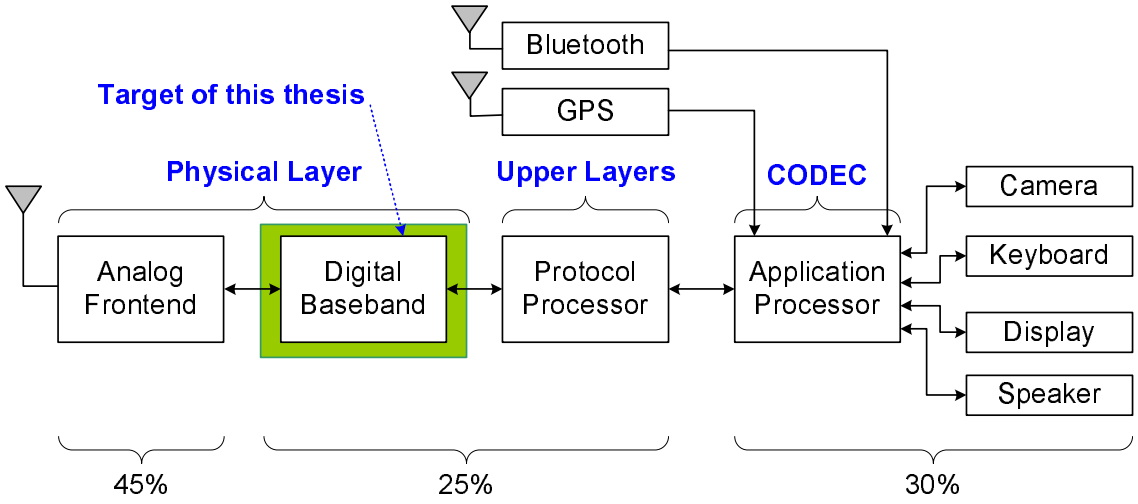


Figure 2.4: Skeleton of a wireless terminal (3G cellular phone)

requirements, it is common to implement the digital baseband in digital ASICs. Meanwhile, the operation of upper layer protocols can be represented as a finite state machine whose state transitions are initiated by external control messages or internal events. Thus, the protocol processor block is implemented in a form of general purpose processor (GPP) except for some hard realtime operations such as MAC response generation and encryption/decryption. The application processor also requires many computations. In order to achieve flexibility and high performance at the same time, a GPP with accelerators is a common implement method.

As shown in Figure 2.4, the analog frontend circuits is a primary power dissipator in wireless terminal. The analog frontend, application processor with other user interface units, and baseband processor correspondingly consume about 45, 30, and 25% of 3G cellular phone power [22].

Among the four major blocks in Figure 2.4, the ASIC based analog frontend and

digital baseband are within the scope of the SDR. However, this thesis only focuses on the architecture of digital baseband processor.

2.5 Software Defined Radio

The SDR is to implement all functional blocks of wireless communication systems with software routines and programmable hardware instead of inflexible ASICs. It enables the support of many wireless protocols without specialized hardware because a simple software routine change is enough for switching to other wireless protocols.

Because wireless communication systems consist of basestations and wireless terminals, the physical layer of both sides can be implemented in the form of the SDR. Battery powered terminals have more strict power constraint compared to AC powered basestations. So, the realization of SDR concept on wireless terminals is more difficult.

The SDR is advantageous to all bodies related to wireless communication systems. At first, it reduces the development time and cost of manufacturers. By reusing identical hardware platform for many terminals with different protocols, it is possible to reduce the time to market and development cost. Second, the SDR allows service providers to upgrade infrastructure without substantial cost. Most wireless protocols continuously upgrade to provide better services. So, if a system was not designed with SDR concept, service providers need to change the hardware of infrastructure to cope with such protocol evolution. A typical example is the addition of HSDPA channel on W-CDMA protocol. By adding this channel, the maximum data rate of W-CDMA protocol is increased from 2 Mbps to 14 Mbps. Third, the SDR provides seamless wireless connections to end users with one wireless terminal. This service is one of key features of 4G wireless communication systems.

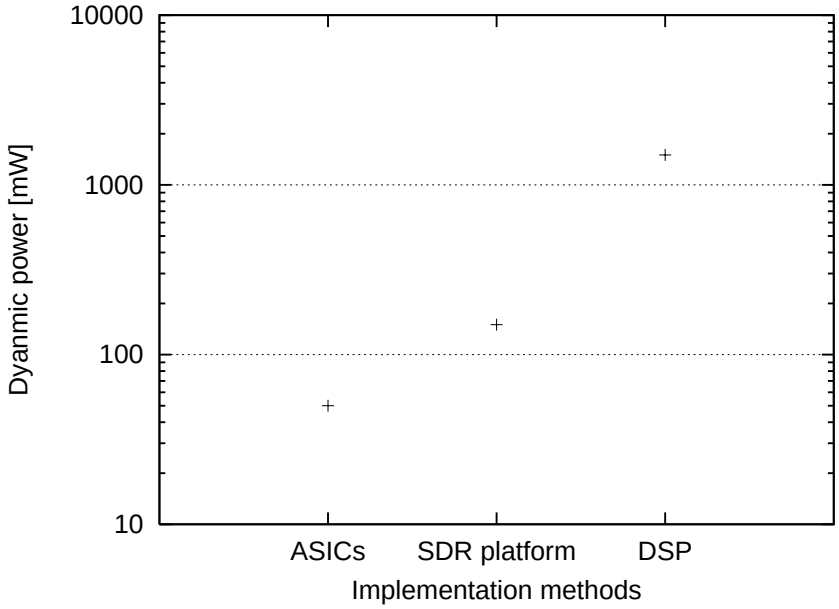


Figure 2.5: Comparison of dynamic power consumption of ASICs, SDR platform, and DSP

A fundamental challenge in a hardware system for the SDR is to overcome the inefficiencies in power and throughput caused by increasing the flexibility required for supporting multiple protocols. Thus, special purpose circuits for single protocol exhibit the best power and throughput performance, whereas fully programmable hardware such as DSP shows the worst performance. The performance of SDR platforms is placed between these two extremes. This relation is depicted in Figure 2.5. In this graph, we compare the dynamic power consumption of ASICs, SDR platforms, and DSP under identical workload[1]

[1]We assume 2Mbps turbo decoder workload where code rate R=3. the power data of ASICs is from [23], that of SDR platform is from the experiment results shown in Chapter VII, and that of DSP is from [24][25].

CHAPTER III

ALGORITHM LEVEL WORKLOAD CHARACTERIZATION

3.1 Introduction

In this chapter, this thesis analyzes the characteristics of signal processing algorithms comprising baseband processing workload. From the view point of computer architecture, this thesis identifies the computation and data movement patterns of major signal processing algorithms because these patterns have directly impacts processor architecture. Although detailed characterization was done, this chapter only contains analysis results to avoid distracting from the main topic of this thesis, the design of baseband processor architecture. Detailed analysis results can be found in Appendix.

3.2 Digital Baseband Operations

The detail operation of the digital baseband is quite different according to the type of communication technologies. However, according to their role, it is possible to model the operation of the digital baseband into the combination of five function blocks as shown in Figure 3.1: channel coding/decoding, block interleaving/deinterleaving, modulation/demodulation, channel estimation, and pulse shap-

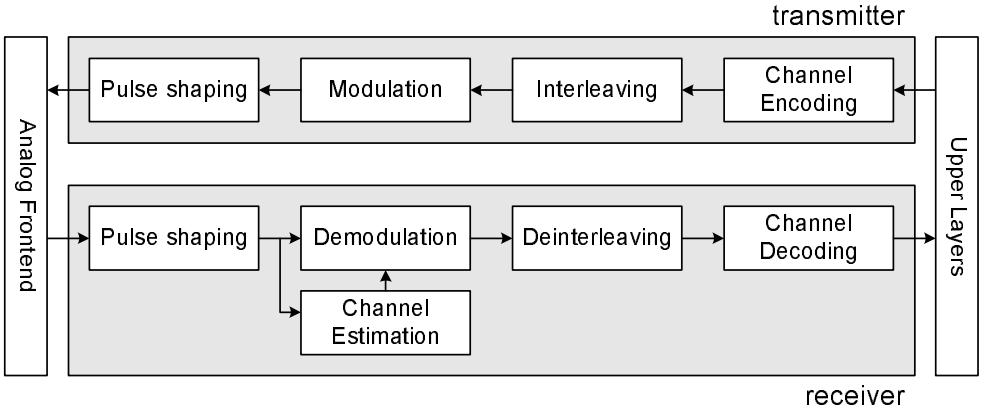


Figure 3.1: Generalized block diagram of wireless terminal's digital baseband

ing.

The function of a channel coding/decoding block is forward error correction. Because wireless channels are very unreliable, some of a received signal is corrupted while propagating through wireless channel. The channel coding/decoding makes it possible to correct the corrupted signal at the receiver without retransmission. The channel coder in the transmitter adds systematic redundancy on the transmitting information and the channel decoder in receiver exploits this redundant information to correct the corrupted signal. Among many existing channel coding schemes, Convolutional code and Turbo code are widely used at most wireless communication networks [26]. For the decoding of these codes, Viterbi algorithm is most widely used [27]. Recently low density parity check (LDPC) codes have begun to be adapted in wireless communication networks [28].

Interleaving/deinterleaving is used to overcome burst errors occurring within short time interval. In wireless channel, abrupt changes of channel quality are frequently observed and the channel decoder shows bad performance at such burst error patterns. Block interleaving/deinterleaving minimizes the effect of burst errors by

scattering error signal over longer time interval.

Modulation[1] is a procedure to map input information bit sequence on specially designed signal waveforms. Demodulation is a procedure to estimate transmitted information bit sequence from received signal waveform. Because the transmitted signal is distorted by a wireless channel, it is required to estimate wireless channel characteristic for better demodulation performance. Popular modulation/demodulation schemes are TDMA, CDMA, and OFDMA.

The channel estimator in receiver measures the characteristics of wireless channel and provides the estimation result to the demodulator. Because a wireless channel is a time varying random function, channel estimators need to measure channel characteristics periodically. In CDMA systems, multipath searcher is used for channel estimation [29]. In TDMA systems, although various estimation algorithms are used for this purpose, maximum likely sequence estimation (MLSE) algorithm is most popular among them [30]. However, in OFDMA systems, a simple channel estimator is enough in receiver because OFDM symbol is inherently robust to the signal distortion caused by the wireless channel.

Pulse shaping is required to convert binary digital information into a frequency limited signal [31]. A binary impulse train, which is used in the digital baseband, is a signal with infinite frequency bandwidth. However, the frequency band, which is allowed for practical communication networks, is limited. In transmitters, the pulse shaping suppresses signal terms out of the allowed frequency band such that the interference to other channels is minimized. However, at receiver, the pulse shaping filter suppresses nose signal power through matched filtering[2].

[1]Modulation and demodulation discussed in this section are not the operations performed in analog frontend, that place signal from baseband to carrier band and vice versa.

[2]It is to compute a convolution between the received signal and the complex conjugate of the signal waveform used for the pulse shaping at the transmitter.

3.3 Major Computation Kernels of Baseband Workload

As a first work, this thesis identifies major computation kernels of baseband workload, and the results are summarized in Table 3.1. From this table, it is possible to observe two important characteristics of the baseband workload. First, the baseband operation is the mixture of parallelizable and sequential workloads. Second, the number computation kernels is limited. Table 3.1 shows there exist only 6 parallelizable vector kernels and 8 scalar kernels. Detailed explanations on all kernels can be found Appendix.

At the case of parallelizable vector kernels, their computation pattern and data movement patterns are almost deterministic. However, some scalar kernels can be implemented with many different ways. In other word, the scalar kernels demand higher level of flexibility than the parallelizable vector kernels.

Thus, it is possible to conclude that either pure parallel architecture or pure scalar architecture is not appropriate for the baseband processing. Defining two datapaths, one for parallelizable vector kernels and the other for scalar kernels, will be useful for improving the power efficiency and throughput of system.

3.4 Characteristics of Parallelizable Computation Kernels

In this subsection, we further discuss the characteristics of parallelizable kernels in order to derive hardware design information. As we will see in next chapter, the parallelizable kernels dominate the baseband workload. So, more intensive analysis is required for efficient system design.

We found that the operation of all parallelizable kernels can be described as a combination of the following macro operations: data load, data alignment, vector computation, vector reduction to scalar, and data store.

Key Kernels	Vector/Scalar	Vector Width	System Type	Function Block
FIR filter	vector	6-320	TDMA,CDMA,OFDMA	pulse shaper, channel estimator
Pattern matching	vector	16	CDMA	synchronization
min/max finding	vector	32-10248	TDMA,CDMA,OFDMA	channel decoder
Viterbi-ACS	vector	64-256	TDMA,CDMA,OFDMA	channel decoder, channel estimator
Viterbi-BMC	vector	64-256	TDMA,CDMA,OFDMA	channel decoder, channel estimator
FFT	vector	64-2048	OFDMA	demodulation
Viterbi-TB	scalar	-	TDMA,CDMA,OFDMA	channel decoder, channel estimator
Interleaving	scalar	-	TDMA,CDMA,OFDMA	interleaver, deinterleaver, channel decoder
Symbol mapping	scalar	-	TDMA,CDMA,OFDMA	modulator, demodulator
Channel encoding	scalar	-	TDMA,CDMA,OFDMA	channel encoder
Sliding window	scalar	-	TDMA,CDMA,OFDMA	frame detection
Code generation	scalar	-	CDMA	modulator/demodulator
Interpolation	scalar	-	OFDMA	demodulator
Frequency tracking	scalar	-	OFDMA	demodulator

Table 3.1: Major computation kernels comprising the baseband processing workload

Vector Kernels	Load	Alignment	Computation	Reduction	Store
FIR Filter	scalar	vector shift	conditional complement, real multiplication	aggregation	scalar
Pattern Matching	single vector	no alignment	conditional complement	aggregation	scalar
min/max Finding	multiple vector	no alignment	compare-and-select	min/max search	scalar
Viterbi-BMC	multiple vector	Viterbi trellis	subtraction, comparison, conditional complement	-	vector
Viterbi-ACS	multiple vector	Viterbi trellis	addition, compare-and-select	-	vector
FFT	multiple vector	FFT butterfly	complex multiplication, addition, complement	-	vector

Table 3.2: Characteristics of the parallelizable vector kernels in the baseband processing workload

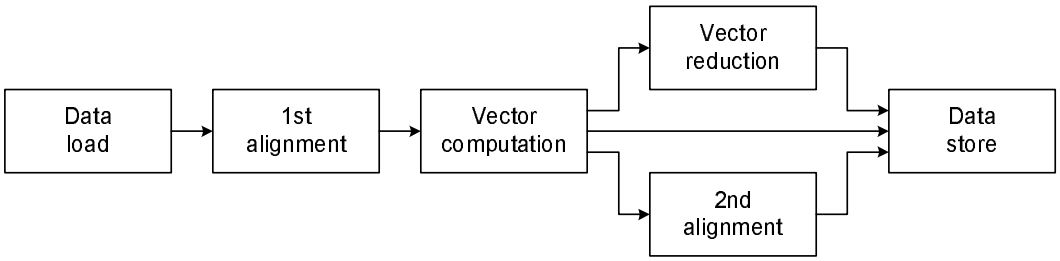


Figure 3.2: A conceptual macro pipeline model which can describe the operation of all parallelizable kernels

Data load macro operation is to read input operands from a memory. All parallelizable kernels needs to load data from a memory for their operations. Data alignment macro operation is to arrange the sequence of the loaded data. Vector computation macro operation is to currently perform arithmetic or logical operations. Vector reduction macro operation is to convert the output of vector computation into a scalar data. Data store macro operation is to save the results of the vector computation or vector reduction to a memory.

As shown in Figure 3.2, these five macro operations form a conceptual macro pipeline. This conceptual macro pipeline shows two interesting characteristics. One is there exists an unidirectional data dependency chain between macro operations. Another is that an identical operation is continued for a long period because all parallelizable kernels have sufficient input data. Thus, there is no need to reconfigure the macro operations at every cycle. From the above two characteristics, it is possible to derive a conclusion that all parallelizable computation kernels can be mapped on a hardware pipeline, which consists of macro operation blocks and has a streaming style control scheme.

Figures from 3.3 to 3.7 represent how all parallelizable vector kernels can be

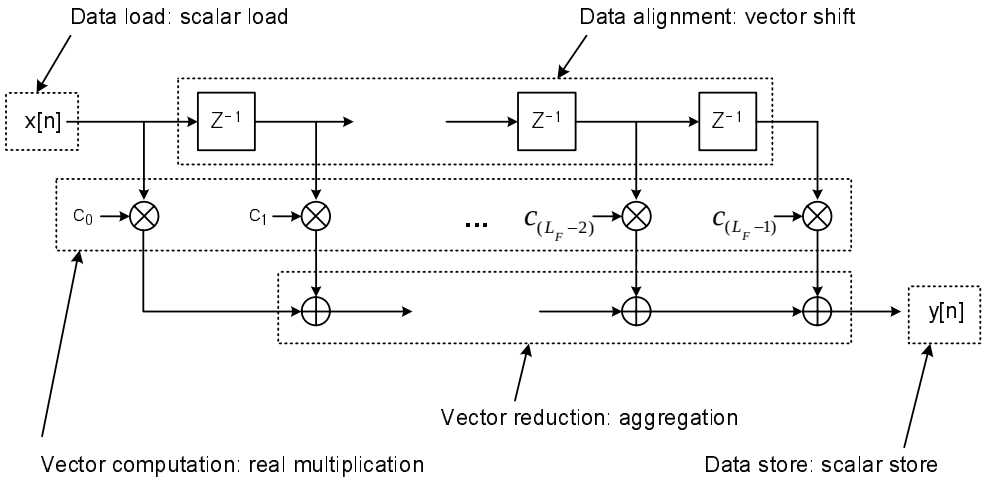


Figure 3.3: Interpretation of the FIR filter kernel into the macro pipeline

interpreted into the conceptual macro pipeline.

3.5 Types of Macro Operations

3.5.1 Data Load

In the parallelizable kernels, there exist three types of data load patterns: scalar load, single vector load, and multiple vector load. In the scalar load, in every cycle, only one data word is loaded from the data memory. The data load pattern of the finite impulse response (FIR) filter is the scalar load. In single vector load, only one vector operand is loaded from the data memory in every cycle. The data load pattern of the pattern matching kernel is the single vector load. In the multiple vector load, multiple input vector operands are loaded from the data memory in every cycle. The Viterbi branch metric computation (BMC) / add compare select (ACS) and fast fourier transform (FFT) kernels need to load multiple vectors at every cycle.

Conventionally, a datapath with vector ALU needs two input vector operands for its operation. Thus, while the datapath executes the parallelizable kernels with the

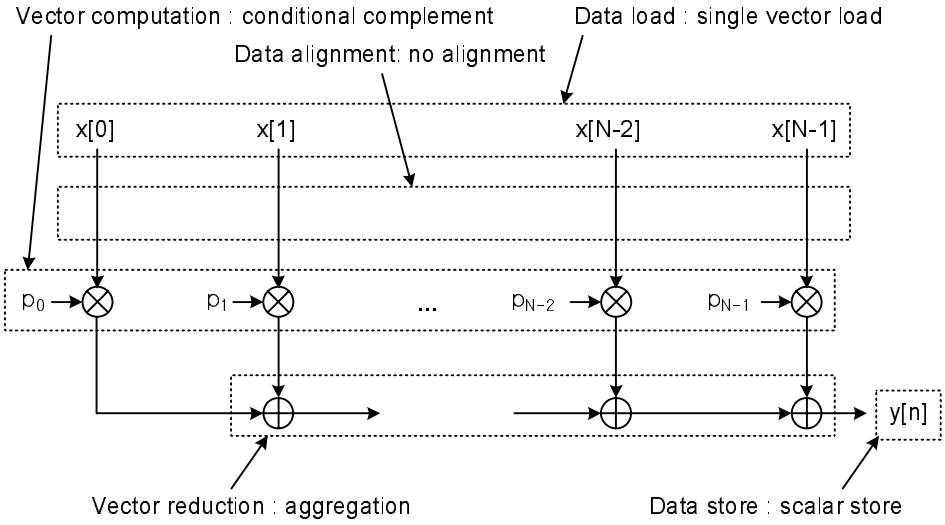


Figure 3.4: Interpretation of the pattern matching kernel into the macro pipeline

single scalar load or single vector load pattern, the previously loaded information has to be reused to build two input vector operands to the vector datapath. For this purpose, storage devices are additionally required.

3.5.2 Data Alignment

The data alignment patterns of the parallelizable kernels can be classified into five types: no alignment, vector shift, Viterbi forward trellis, Viterbi backward trellis, and FFT butterfly. No alignment means that the sequence of input data in data memory is enough for an actual vector computation. min/max finding and pattern matching kernels correspond to this case. The operations of these kernels are independent of data sequence. Vector shift is the element level data shift shown in Figure A.20. The vector shift pattern appears at the FIR filter operation. The data movement pattern of the Viterbi forward trellis is shown at Figure A.5(a). The Viterbi forward trellis pattern appears at the Viterbi decoder and max-log maximum a posterior (MAP) Turbo decoder. The data movement pattern of the Viterbi backward trellis operation

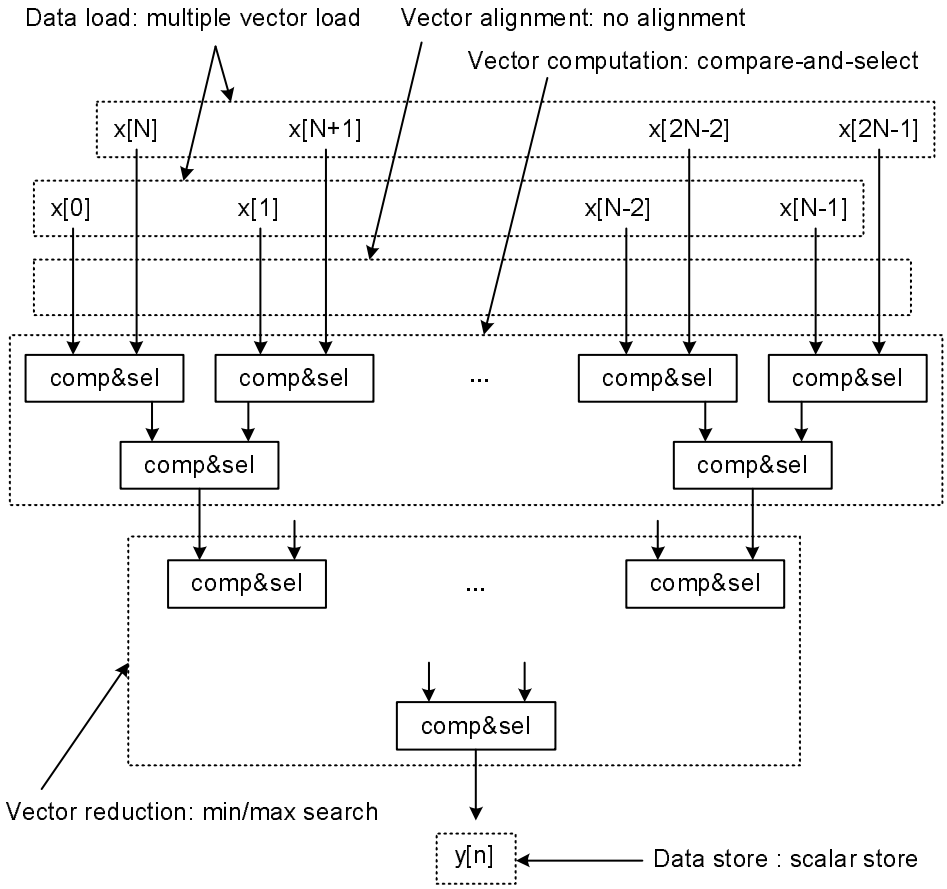


Figure 3.5: Interpretation of the min/max finding kernel into the macro pipeline

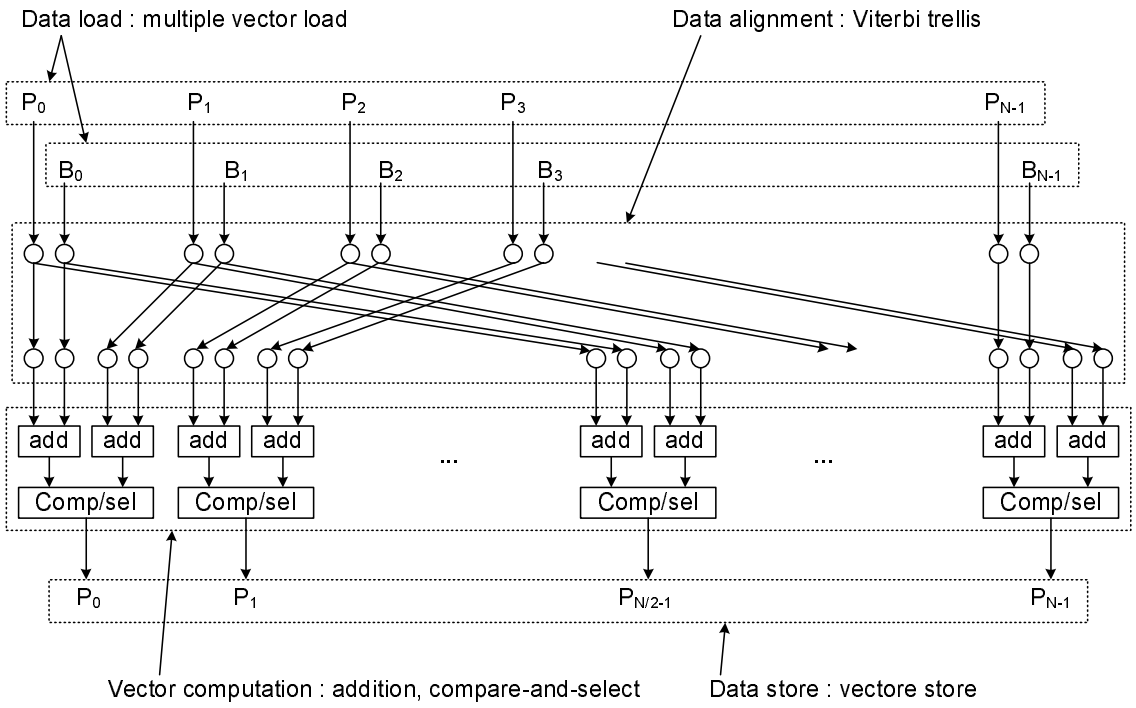


Figure 3.6: Interpretation of the Viterbi-ACS kernel into the macro pipeline

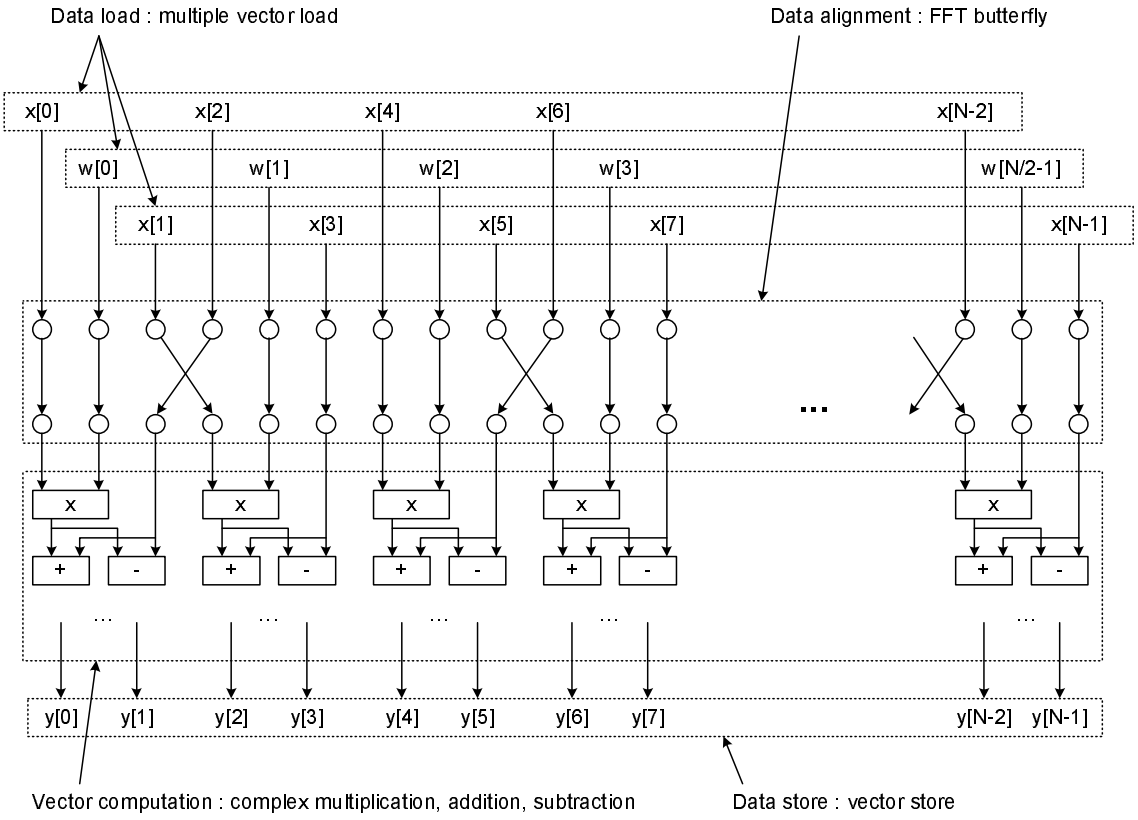


Figure 3.7: Interpretation of the FFT kernel into the macro pipeline

is shown at Figure A.9(a). It appears at the max-log MAP Turbo decoder. The FFT butterfly pattern is shown in Figure A.17(a). One common characteristic of all data movement patterns is that the relation between source node and destination node can be represented by simple linear equations.

3.5.3 Vector Computation

The computations required by the parallelizable kernels are real number multiplication, complex number multiplication, addition, subtraction, conditional complement, conditional selection, and compare-and-select. The conditional complements is to compute one's or two's complement of the input operand according to the value of a condition flag. The conditional selection is to select one data from two input operands according to the value of the condition flag. First interesting factor is that the number of computation patterns is small. Second point is that all computations are well matched with an SIMD style datapath because the number of parallelizable computations is, in most cases, greater than 32 and all parallelizable computations can be controlled by identical control signals. Third, all input operands of these computations are 8bit or 16bit fixed point numbers.

3.5.4 Vector Reduction

There exist two kinds of reduction methods: aggregation and minimum/maximum value search. Aggregation is to produce a scalar output by adding all elements of the input vector. The minimum/maximum value search is to find the minimum or maximum value from the input vector. The FIR filter, pattern matching, min/nax finding kernels need the vector reduction. No reduction is required at the Viterbi-BMC/ACS and FFT kernels.

3.5.5 Data Store

There are only two kinds of data store patterns in the parallelizable kernels: scalar store and vector store. Scalar store is to save one data element into memory, whereas vector store is to store multiple data elements into memory. With some parallelizable kernels, the vector reduction converts a vector input data into a scalar. Then, the required data store pattern is the scalar store. However, the data store pattern of the Viterbi-BMC/ACS and FFT is the vector store because these kernels produce a vector output and no reduction is performed on their vector output.

CHAPTER IV

SYSTEM LEVEL WORKLOAD CHARACTERIZATION: W-CDMA

4.1 Introduction

In this chapter, this thesis discuss the system level behavior of the physical layer of wireless terminals. This system level analysis answers many design questions that are raised at high level system design procedure. For instance, the proper number of processor cores, the proper granularity of processor core, the homogeneity of processor core, the interconnection mechanism between processor cores can be answered by this analysis.

4.2 Operation Modes of Wireless Terminal

Wireless terminals change their operation mode according to user activity [32][33][34]. When there are active user applications, wireless terminals employ all of their functionality to support high data rate communications (*active mode*). At short idle periods between traffic bursts, wireless terminals maintain a narrow control channel with the basestation for fast transition to the active mode (*control hold mode*). However, even when there exist no active user applications, wireless terminals can not turn off completely, because requests from other terminals may be received at

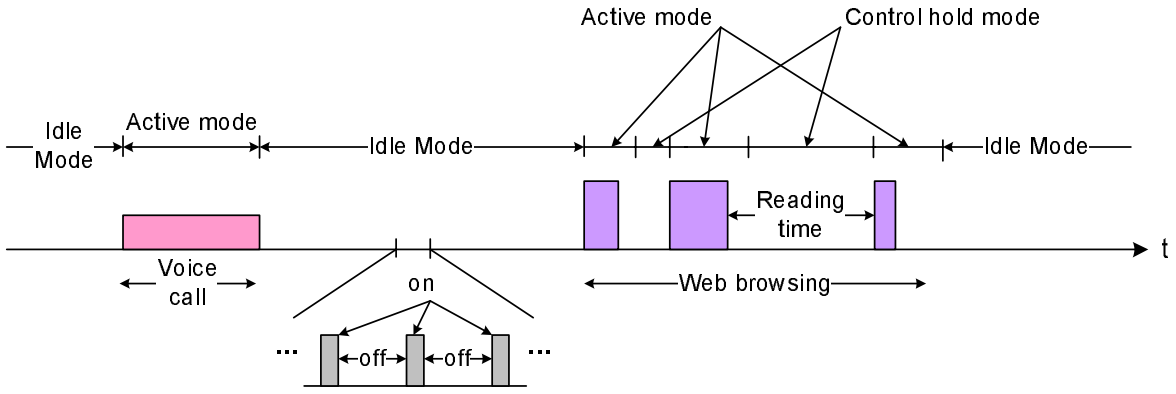


Figure 4.1: The active, control hold, and idle modes of wireless terminals

unpredictable times. So, wireless terminals activate a subset of their communication components that are sufficient to receive the requests (*idle mode*). In the idle mode, the operation is intermittent. Instead of continuously monitoring for signals, terminals completely power off their receiver and activate it only for short pre-scheduled periods. Wireless terminals spend most of their operation time in the idle mode.

For instance, if a laptop user starts a internet connection and opens a new web page, then the corresponding operation mode is the active mode. If the user spends time reading the opened web page, then a terminal enters the control hold mode. Finally, if the user terminates web browser and starts other laptop task, then a terminal enters the idle mode. This relation is depicted in Figure 4.1.

4.2.1 Characteristics of Active and Control Hold Modes Operations

In the active mode, all major blocks discussed in the previous chapter are activated. In transmission path, the channel encoder, block interleaver, modulator, and pulse shaping filters are used and, in reception path, the pulse shaping filter, demodulator, channel estimator, block deinterleaver, and channel decoder are used. Thus,

all computation kernels shown in Table 3.1 are related to active mode operation.

Among all computation kernels, as we will show in the following section for W-CDMA terminal operation, the channel decoder dominates the active mode workload. Thus, in order to minimize system power, it is important to process the Viterbi BMC/ACS kernels with minimum power cost because these two kernels dominate the channel decoder operation, and consequently the active mode workload. Because it is required to process high rate traffic, to meet peak performance with minimum power consumption is important design issue.

The operations done in the control hold mode are almost identical to that of active mode except for low data rate. Channel encoder and decoders are only used for control message decoding. So, pulse shaping filters and channel estimator are dominant workloads of the control hold mode. According to modulation schemes, dominant parallelizable kernels are different. In CDMA based systems, the FIR filter dominates the workload of the control hold mode. In TDMA based systems, the FIR filter and Viterbi BMC/ACS kernels are dominant. In OFDMA systems, the FFT is a major kernel.

In order to finish the operation on the current frame before the arrival of next frame, all computation kernels used in both active and control hold modes have tight processing time requirement.

4.2.2 Characteristics of Idle Mode Operations

Even in the idle mode, almost all signal processing algorithms, comprising the receiving path of a wireless terminal, participate in the operation of a wireless terminal. However, if we additionally consider the amount of workload, the idle mode operation is dominated by two computation kernels: the FIR filter and sliding window. The

FIR filter style computations appear in the pulse shaping, the synchronization, and the channel estimation procedures. The sliding window style computation appears in the frame detection operation in the ad hoc network. It translates as follows: the power efficiency of the idle mode processor is dominated by the power efficiency of these kernel computations. Detailed discussion on these kernels can be found in Appendix.

In addition to the limited computation patterns, the idle mode operation also has another characteristic, more relaxed processing time requirements. In active mode, a baseband processor must finish the operations on the current frame before the next frame arrives, in order to avoid buffer overflow. However, in the idle mode, the inter-frame arrival time is much longer than that of the active mode, by at least an order of magnitude. Thus, for power reduction, it is important for the idle mode processor to fully exploit the allowed processing time.

4.3 A Case Study: W-CDMA Terminal

As a case study, this section analyzes the operation of WWAN terminal among the terminals of many wireless networks. Compared to other networks' terminals, the WWAN terminal has most complex operation scenario in order to support terminal mobility. It also uses the most complex signal processing algorithms to maximize network capacity. Thus, it is possible to state that the workload of the WWAN terminal is a superset of the workload of terminals for other types of networks. Among the terminals of many WWANs, this thesis selects a W-CDMA terminal. The W-CDMA is one of the most advanced networks among WWANs. Because all WWAN terminals exhibit similar behavior, the W-CDMA terminal can be a good reference model for the system level workload analysis.

Application Type	
Service type	Packet service
Data link	2 Mbps downlink / 128 Kbps uplink
Signaling link	3.4 Kbps bidirectional
Channel Condition	
Number of basestations	3
Number of rake fingers	12
Number of average Turbo iterations	5

Table 4.1: Operation conditions of a W-CDMA terminal which are assumed for system level workload analysis

4.3.1 Terminal Operation Condition

Before the detailed workload analysis, this thesis clarifies the operation conditions of the W-CDMA terminal because the workload of the W-CDMA physical layer is affected by many reasons: 1) operation state; 2) application type; and 3) radio channel status. Among these reasons, the operation state change most significantly affects workload of physical layer. So, this thesis analyze the workload variation according to operation state changes in detail, but assume worst case conditions at the case of application type and channel condition.

Generally, it is possible to classify application services into two types: circuit service and packet service. Circuit service is a constant data rate service such as voice call. Packet service is a variable data rate service such as internet access. Because the burst packet arrival pattern of the packet service demands a more complex resource management scheme, we select the packet service as a representative service. In addition, this thesis further assumes an asymmetric packet service that consists of a 2 Mbps link on the direction from basestation to terminal, downlink, and a 128 Kbps link on the reverse direction, uplink. The asymmetric channel assumption matches the behavior of most packet services, for instance web browsing. For control

signalling, the packet service additionally has a bidirectional signaling link with a 3.4 Kbps data rate.

In detail, the workload of a W-CDMA terminal is also varied by three radio channel conditions: 1) the number of basestations that communicate with a terminal at the same time; 2) the number of strong multipath components; and 3) the quality of received signal. This thesis assumes 3 basestations, and 4 strong multipath components from the signal of a basestation with a rule of thumb. Thus, the W-CDMA terminal activates a total of 12(=3×4) rake fingers[1]. The quality of the received signal has a direct impact on the number of iterations of the Turbo decoder. Thus, this thesis assumes the average number of Turbo decoder iterations as 5 times per frame.

4.3.2 System Block Diagram

Figure 4.2 shows a detailed block diagram of the W-CDMA physical layer. It shows the variation of active algorithm blocks according to the terminal operation modes which was discussed in the previous chapter. In the idle mode, the low pass filter (LPF)-Rx, demodulator (descramble, despreader, and combiner), and multipath searcher, which are a subset of reception path, are active. It is worth noting that an complex and power consuming Viterbi decoder or Turbo decoder does not participate in the operation of the idle mode. In the control hold mode, a bidirectional 3.4 Kbps signaling link is established with basestations. Thus, a terminal activates both transmission and reception paths including the convolutional encoder/Viterbi decoder, LPF-Rx/Tx, modulator/demoulator, multipath searcher, and power control. Power control is important in the CDMA based system. It is to adaptively

[1]In CDMA based system, each multipath component is independently demodulated at the receiver. The rake finger represents the demodulator assigned to a multipath component.

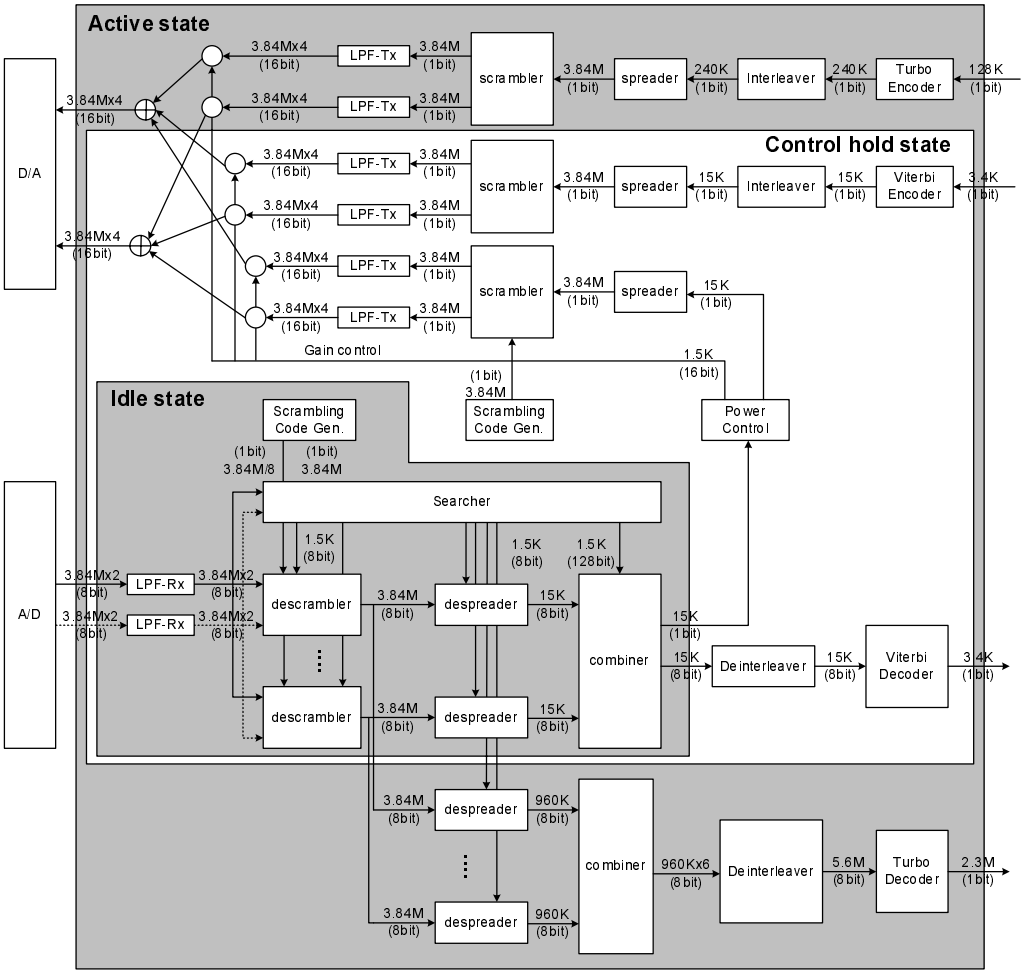


Figure 4.2: Detailed block diagram of the W-CDMA physical layer when it provides the packet data service described in Table 4.1

control signal transmission power according to the signal quality at receiver. Thus, the receiver needs to report the measured quality of the received signal to the transmitter within several hundred micro seconds. It results in a tight timing constraint in the baseband processing of the CDMA based system. In the active mode, a terminal additionally establishes a bidirectional high speed data link which is encoded by the Turbo encoder as shown in Table 4.1. Thus, the Turbo encoder/decoder participate in the active mode operation of a terminal. The activation of the Turbo decoder

	Active/Control hold modes		Idle mode
	Processing Time(ms)	**Execution Freq.(Hz)**	**Processing Time(ms)**
Searcher	Fixed(5*)	50*	Fixed(40*)
Interleaver/Deinterleaver	Fixed(10)	100	-
Convolutional encoder	Fixed(10/20/40)	Variable	-
Viterbi decoder			
Turbo encoder			
Turbo decoder	Variable(10∼50*)		
Scrambler/Descrambler	Fixed(0.67)	1500	Fixed(0.67)
Spreader/Despreader			
LPF-Rx			
LPF-Tx			-
Power control			

Table 4.2: Processing time requirements of the W-CDMA physical layer

increases terminal's workload abruptly.

Figure 4.2 also describes interface between the algorithms of W-CDMA. The number at the top of each arrow represents the number of samples per second, and that at the bottom represents the size of a sample. From the given numbers, we can derive the amount of traffic between the algorithms. The size of most input/output data in the transmission path is 1 bit, but, in the reception path, the size of input/output data is 8 or 16 bit because the channel decoders use soft number which represents binary information with higher precision for better decoding performance. From this diagram, we can see that the data rate is abruptly changed by the spreader and despreader. In the transmission path, data rate is upconverted from kilo sample per second into mega samples per seconds after the spreading operation. The reception path exhibits an opposite conversion.

4.3.3 Processing Time

Table 4.2 shows that the W-CDMA physical layer is a mixture of algorithms with various processing time requirements. The * notation in the table represents that the corresponding parameter is determined by design choice not by specification. Other parameters in the table are explicitly specified by the W-CDMA standard. The processing times shown in the second and fourth columns are the allowed completion time of each algorithm whenever it is called. The task frequency shown in the third column is the number of executions of each algorithm within a second.

We assume that the searcher is executed every 20 msec because a radio channel can be considered as invariant during this interval. The scrambler, spreader, and LPF have periodic and very strict processing time requirements because they participate in the power control action. The convolutional code is mainly used for the circuit service with a constant data rate, so the Viterbi decoder needs to complete its operation before the arrival of the next frame to avoid buffer overflow. The processing time of the Viterbi decoder can be configured with 10, 20, or 40 msec according to the service configuration on the frame arrival interval. Whereas the Turbo code aims the packet service with a burst packet arrival pattern. By buffering of packet burst, it is possible to relax processing time constraint substantially. This thesis assumes that the processing time of the Turbo decoder varies between 10$\sim$50 msec according to the amount of buffered traffic. In the idle mode, tasks have loose timing constraints, so the searcher operation can be performed in sequential with minimal hardware and the task execution frequency is not of concern.

	Active		Control Hold		Idle	
	(MOPS)	%	**(MOPS)**	%	**(MOPS)**	%
Searcher	**26538.0**	**42.1**	**26358.0**	**58.4**	**3317.3**	**37.7**
Interleaver	2.2	0.0	2.2	0.0	-	-
Deinterleaver	0.2	0.0	0.2	0.0	-	-
Conv. encoder	0.0	0.0	0.0	0.0	-	-
Viterbi Decoder	200.0	0.3	200.0	0.4	-	-
Turbo encoder	0.0	0.0	0.0	0.0	-	-
Turbo decoder	**17500.0**	**27.8**	**0.0**	**0.0**	-	-
Scrambler	245.3	0.4	245.3	0.5	-	-
Descrambler	**2621.4**	**4.2**	**2621.4**	**5.8**	**889.2**	**10.1**
Spreader	297.5	0.5	297.5	0.7	-	0.0
Despreader	**3642.5**	**5.8**	**3642.5**	**8.0**	**607.1**	**6.9**
LPF-Rx	**3993.6**	**6.3**	**3993.6**	**8.8**	**3993.6**	**45.3**
LPF-Tx	**7897.2**	**12.6**	**7897.2**	**17.4**	-	-
Power control	0.0	0.0	0.0	0.0	-	-
Total	62937.0	-	45272.9	-	8807.2	-

Table 4.3: Peak workload profile of the W-CDMA physical layer and its variation according to the operation mode change

4.3.4 Peak Workload Profile

The detailed peak workload profile of the W-CDMA physical layer is shown in Table 4.3. For this analysis, we compiled our W-CDMA benchmark [35] with an Alpha gcc compiler, and executed it on M5 architectural simulator [36]. The W-CDMA benchmark describes the algorithms of W-CDMA physical layer with C language. We measured the instruction count that is required to finish each algorithm. Peak workload of each algorithm is achieved by dividing the instruction count by the most tight processing time requirement of each algorithm that is shown in Table 4.2.

The first thing to note in Table 4.3 is that the total workload varies according to the operation mode change. The total workloads in the control hold and idle modes are about 72% and 14% of that in the active mode. Second, the types of active algorithms and the workload distribution between them also vary according

		Scalar Workload (%)	Vector Workload (%)	Vector Width	Element Width (bit)	Max Concurrent Thread
Searcher		3	97	320	1,8	5120
Interleaver		100	0	-	-	-
Deinterleaver		100	0	-	-	-
Viterbi encoder		60	40	8	1,1	1
Viterbi Decoder	BMC	1	99	256	8,8	45
	ACS	1	99	256	8,8	45
	TB	100	0	-	-	-
Turbo encoder		60	40	4	1,1	2
Turbo Decoder	BMC	1	99	16	8,8	20
	ACS	1	99	16	8,8	20
	TB	100	0	-	-	-
Scrambler		1	99	2560	1,1	1
Descrambler		1	99	2560	1,8	1
Spreader		100	0	-	-	-
Despreader		100	0	-	-	-
Combiner		100	0	-	-	-
LPF-Tx		1	99	32	1,16	6
LPF-Tx		1	99	32	8,8	2
Power Control		100	0	-	-	-

Table 4.4: Parallelism available in the algorithms of the W-CDMA physical layer

to the operation mode change. In the active and control hold modes, the multipath searcher and Turbo decoder are dominant. In the idle mode, the multipath searcher and LPF-Rx are dominant. The workload of transmission path is not substantial compared to that of receive path.

4.3.5 Parallelism

Table 4.4 shows a breakdown of the available parallelism in the W-CDMA physical layer. We define DLP as the maximum SIMD vector width and thread level parallelism (TLP) as the maximum number of SIMD threads that can be executed in parallel. The second and third columns in the table are the ratio between the run time of the scalar code and the vector code. The fourth column represents maximum

possible DLP. Because a vector operation needs two input operands, we separately represent the bit width of two vector operands in the fifth column. The last column shows the TLP information.

From Table 4.4, we can see that the searcher, LPF, scrambler, descrambler, and the BMC/ACS of the Viterbi decoder contain large amount of the DLP and TLP. For the case of the scrambler and descrambler, it is possible to convert the DLP into TLP by subdividing large vectors into smaller ones. Although it is one of dominant workloads, the Turbo decoder contains limited DLP because the allowed maximum vector length of the ACS operation of the Turbo decoder is 8. It is possible to covert this narrow DLP task into wide one by decoding multiple frames simultaneously.

There are also many unparallelizable algorithms in the W-CDMA physical layer. The interleaver, deinterleaver, spreader, despreader, and combiner operations have little DLP and TLP. Fortunately, the workload of these algorithms is not significant as show in Table 4.3. Therefore we can easily increase system throughput and power efficiency by exploiting the inherent DLP and TLP shown in Table 4.4. One factor worth to note is that there is no workloads with substantial instruction level parallelism.

4.3.6 Memory Requirement

Because memory is one of the dominant power consuming elements in most low power systems, the analysis on the characteristics of memory access pattern is important. In general, there are two types of memory in a hardware system: data memory and instruction memory. Table 4.5 presents the data and instruction memory for all algorithms in the W-CDMA.

Columns 2∼7 in Table 4.5 show the size of the required data memory. The data

	Data memory (Kbyte)						**Inst. Mem.** (Kbyte)
	I-buffer		**O-buffer**		**Scratch pad**		
	Kbyte	Mbyte/s	Kbyte	Mbyte/s	Kbyte	Mbyte/s	
Searcher	**20**	**1**	**0**	**0**	**60**	**340**	**3.0**
Interleaver	10	5	10	5	0	0	0.1
Deinterleaver	45	10	45	10	-	-	0.1
Viterbi Encoder	-	-	-	-	-	-	0.5
Viterbi Decoder	1	-	1	0	50	20	1.5
Turbo Encoder	1	1	5	5	-	-	1.5
Turbo Decoder	**45**	**10**	**25**	**5**	**1**	**3770**	**3.5**
Scrambler	1	10	1	10	-	-	0.5
Descrambler	10	100	10	100	-	-	0.5
Spreader	1	5	1	15	1	15	0.5
Despreader	5	60	1	5	1	15	0.5
Combiner	1	25	50	5	-	-	0.1
LPF-Tx	**1**	**25**	**5**	**50**	**1**	**1480**	**0.1**
LPF-Rx	**1**	**15**	**1**	**10**	**1**	**500**	**0.1**
Power control	-	-	-	-	-	-	0.1
Total	**143**	**267**	**155**	**220**	**115**	**6120**	**13.6**

Table 4.5: Memory requirements of the algorithms of the W-CDMA physical layer

memory is further divided into two categories: I/O buffer and scratch pad. The I/O buffer memory is used for data buffering between algorithms. The scratch pad memory is temporary space needed for algorithm execution. This thesis analyzes both size and access bandwidth of the data memory.

From the table, we can see that the W-CDMA algorithms require small amount data memory, generally less than 64 Kbyte. In addition, the scratch pad memory is the most frequently accessed, especially in the searcher, Turbo decoder, and LPFs. The access of I/O memory does not occupy significant portion at the total memory access. Thus, for power reduction, it is crucial to reduce the energy cost of the scratch pad memory while designing memory hierarchy.

The last column of Table 4.5 shows the instruction memory size for each algorithm. The average code size is less than 1 Kbyte and most kernels are even below

0.5 Kbyte. This result is typical in most digital signal processing algorithms due to their computation intensive characteristics. Because our W-CDMA benchmark does not include full function of the W-CDMA physical layer, we need to consider the increase of the instruction memory in a real situation.

CHAPTER V

HIGH LEVEL ARCHITECTURE OF THE BASEBAND PROCESSOR FOR SDR

In this chapter, this thesis discusses the high level architecture of the baseband processor for the SDR. As shown in Figure 5.1, the proposed baseband processor is a chip multiprocessor system which consists of many coarse grain PE. For the interconnection of PEs, a low speed bus is utilized. Each PE has both parallel datapath and scalar datapath in order to efficiently support both parallel and scalar workloads. A global memory is placed on the interconnect bus between PEs to buffer large non-realtime traffic. A GPP is appended to cover system level control actions and maintenance tasks.

5.1 Chip Multiprocessor

The physical layer algorithms demand a high performance computer system with several tens GOPS. There exist two approaches to implement a computer system that satisfies such a high performance requirement: to use a high speed single processor or many low speed processors. Although a single processor architecture is much easier to program, we need to run the single processor at high operation frequency to meet the throughput requirement. Then, the single processor will be power inefficient

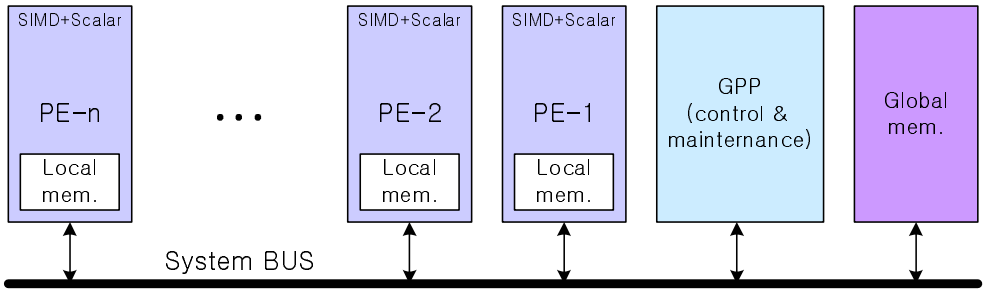


Figure 5.1: The high level architecture of the proposed baseband processor for the SDR

because of the super linear relationship between the operation frequency of circuits and their power consumption. Usually a high performance single processor system dissipates more than one hundred Watt even with state-of-art circuit technology. However, the maximum operation power allowed for the baseband processor is several hundred mW. Thus, using a multiprocessor architecture is inevitable to meet both performance and power requirements of the baseband processor. In a multiprocessor architecture, each PE can run in a lower operation frequency because the workload is shared by multiple PEs. The lower operation frequency allows us to use slower but more power efficient circuits in a PE and it eventually reduces total system power.

Fortunately, the physical layer algorithms inherently show plenty of algorithm level parallelism. As shown in Table 4.5, the channel encoder/decoder, block interleaver/deinterleaver, modulator/demodulator, pulse shaping filter, and channel estimator can be executed in parallel without intensive interactions. Thus, it is possible to map these algorithms on multiple PEs without significant communication overhead.

5.2 Coarse Grain PE

In the implementation of a multiprocessor architecture, the granularity of the PE is one of important design issues because system dynamic power is significantly affected by the size of the PE. The memory access pattern analysis results shown in Table 4.5 provide useful design information. This table shows that, in most computation intensive algorithms, the amount of memory access for internal computation is about 10∼200 times greater than that for the communication with other algorithm blocks. It means that if a PE is strong enough to cover one entire signal processing algorithm, the amount of interprocess communication can be minimized.

Minimizing the amount of interprocess communication is important in a low power system because the energy cost of inter process communication is at least two times higher than that of internal memory access. In detail, let us compared the power consumption of a coarse grain PE and a fine gain PE. With a coarse grain PE, which can covers one signal processing algorithm such as a Viterbi decoder without task partitioning, the operations required to pass the result of one step to the next step are to write on a local memory or register file and to read from it. However, with a fine grain PE, which covers only a part of a signal processing algorithm, the same data are passed through three blocks: the output buffer of the source PE, the interconnection network between two PEs, and the input buffer of the destination PE. In CMOS circuits, accessing storage elements such as register file or memory is the most expensive regarding power. Thus, the fine grain PE will dissipate more energy for the identical operations.

From the view point of system throughput, the coarse grain PE is also a better choice because the communication delay between fine grain PEs degrades system

throughput. Usually, the operation speed of the interprocess communication network is slower than the internal memory access. Furthermore, the interprocess communication time is not deterministic. So, the fine grain PEs must be scheduled under the assumption of worst case delay. These factors degrade achievable maximum system throughput. Even with the coarse grain PEs, we can satisfy both the high throughput and low power consumption by exploiting the DLP, which is abundant in baseband signal processing algorithms. As we discussed in Chapter IV, the dominant workloads of the physical layer exhibit wide DLP.

5.3 Homogeneous PE

In multiprocessor system, all PEs can have identical architecture (homogeneous) or different architectures, which are further optimized to their workloads (heterogeneous). Generally, the system with a homogeneous PE is more flexible but less efficient in power and throughput, whereas the system with a heterogeneous PE is less flexible but more efficient in power and throughput.

From the viewpoint of design cost, the system with heterogeneous PEs demands more design efforts. However, from the viewpoint of flexibility, the system with homogeneous PEs is preferred because the SDR system needs to support various workloads with different workload distributions and computation patterns. The proposed architecture of a PE is well optimized for all major kernels. So, the homogeneous PE is selected.

5.4 Low Speed BUS

Because a coarse grain PE is used in the proposed system, the communication traffic between PEs can be supported by a low speed bus. For example, in Table 4.5

the total amount of traffic between all algorithms is about one Gbps. For this level of traffic, a low speed bus with 32bit data width and 200 MHz operation frequency is sufficient. This bus can support the physical layer operation for W-CDMA 2Mbps data service with about 15% bus utilization. At the 130 nm silicon processing technology, which is used for the power evaluation of the proposed architecture, the 200 MHz operation frequency can be realized with small and power efficient circuit components.

5.5 Memory Hierarchy

5.5.1 Scratch Pad Memory

Like conventional processor architecture, there exist two kinds of memories in the proposed system for baseband processing: data memory and instruction memory. One common characteristic of these memories used in the proposed architecture is the absence of a cache. As well known, a cache is useful for the applications which need to access large memory space. A cache hides the long access delay of large memory. Another advantage of using a cache is to eliminate the memory management overhead. Despite such advantages, a cache is improper for our applications. At first, a cache is power expensive because it is a kind of content access memory which performs power consuming tag matching operations. The second disadvantage is the variation on memory access time. In all signal processing applications, the most important performance metric is to guarantee their completion time. The variation on memory access time, which can be caused by cache miss, requires a loose task scheduling, which concerns a cache miss case, and it directly results in lower system throughput. This delay variation also makes the system debugging more difficult. The processing time violation error related to the cache miss is quite difficult to

revive.

Instead of cache, this thesis proposes to use scratch pad memory for operation data and executable codes. The CELL of IBM uses a similar scheme. Although the programmer needs to be concerned with the memory management issue of the scratch pad memory, it is not a critical problem because of the following reasons. At first, the size of memories for application kernels is not huge. As shown in Table 4.5, the 32K memory is enough for data and the 5K memory is sufficient for instructions. If the scratch pad memories are bigger than the required size, there will be no frequent memory replacement operations. Second, the algorithms mapped on the proposed architecture show deterministic memory access patterns. Thus, the memory management is not a difficult task in our applications.

5.5.2 Hierarchical Memory

The size of the memory is important for power efficiency. Because smaller memory consumes less power, the size of memory needs to be minimized. As discussed before, the required size of memory is not huge in most workloads. However, there exist several exceptional cases that require huge memory, the Turbo decoder and hybrid automatic repeat request (HARQ) [37]. The Turbo code is used for the transmission of high speed packet data, which a has bursty arrival pattern. However, it takes a long time to decode Turbo coded data. Thus, a traffic buffer is required to resolve such discrepancy between the data arrival pattern and the data processing pattern. HARQ is an advanced ARQ scheme that stores error frames to reuse in the decoding of retransmitted frames. Because it takes several ten milliseconds to receive retransmitted frames, the corrupted error frames need to be stored until restarting decoding.

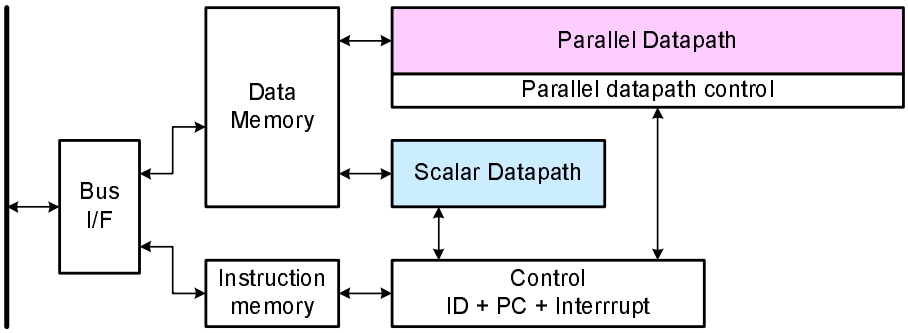


Figure 5.2: The skeleton of a PE, which has both scalar datapath and SIMD datapath

In order to satisfy these two requirements, the power efficiency and the need for large buffer space, this thesis proposes hierarchical memory structure, which consists of local memory and global memory. As shown in Figure 5.1, each PE has a local memory with minimal size to store operation data and all PEs share a large global memory to buffer Turbo decoder input data and HARQ data.

Another important factor is that the local memory is used as buffer space for the communication between PEs. The use of global memory for interprocess communication needs to be minimized because of memory copy overhead, which additionally occurs when using the global memory for the communication. In most cases, the data produced by one PE has only one consumer. Thus, the direct data transfer between PEs is more power efficient than a scheme using the shared memory.

5.6 SIMD+Scalar

One key aspect for executing algorithms mapped on a PE is to use an SIMD architecture. There are two main reasons why an SIMD architecture is beneficial. The first reason for using SIMD is that the baseband operations contain a large amount of DLP as shown in Table 4.4. For example, the multipath searcher shows 320 wide

data parallelism and the Viterbi decoder shows 256 wide data parallelism. Secondly, the computations of the tasks with high levels of DLP are mainly add/subtract or 16bit multiplications. The absence of complicated operations allows us to duplicate the SIMD functional units with minimal area and power overhead. Furthermore, we can expect significant power gain because a SIMD architecture can execute multiple data elements with one instruction memory accessing and decoding.

In addition to the SIMD support, the physical layer also requires scalar support because there are many small sequential operations in the physical layer as shown in Table 4.4. The sequential operations are the block interleaver/deinterleaver operation and the trace back (TB) operation of the Turbo and Viterbi decoders. In the case of the W-CDMA workload, the correlation peak searching operation in the multipath searcher, the scrambling and spreading code generators in the modulator, and the combiner in the rake receiver are sequential.

Therefore, a PE needs to have both an SIMD datapath and a sequential datapath. For higher throughput, it is preferred to run the SIMD datapath and scalar datapath in parallel. For this, two datapaths have independent control paths including instruction memory, instruction decoder, and control logics. However, two datapaths need to share the local memory for power reduction. It is because the scalar workload and the parallel workload are tightly coupled. The Viterbi decoder is a typical example. The parallelizable BMC and ACS operations are assigned on the SIMD datapath and the sequential TB operation is assigned on the scalar datapath. Such a configuration requires a lot of communication between two datapaths because the result of the BMC/ACS operations must be passed to the scalar datapath for the TB operation. Sharing data memory eliminates the communication cost between two datapaths. The skeleton of the proposed PE is depicted in Figure 5.2.

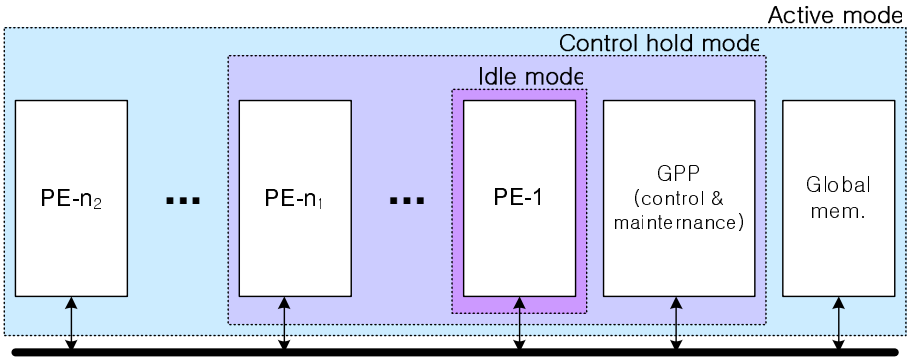


Figure 5.3: The controlling of the number of active PEs according to terminal's operation state change

5.7 Support Wide Workload Variation

As shown in Table 4.3, the workload of the baseband processor has a wide range of variance according to the terminal's operation state. A chip multiprocessor architecture is proposed in this thesis in order to efficiently cope with such wide workload variation through controlling the number of PEs, which are involved in the baseband signal processing. Figure 5.3 depicts the proposed scheme that is to control the number of active PEs according to workload variation.

In the idle mode, one PE is activated. Because the amount of computation required for the idle mode operation roughly corresponds to the 1% of that of active mode operation (refer to Table 4.3), one PE is sufficient. The GPP for control and maintenance purposes can also be turned off for idle mode power reduction. In the control hold mode, several PEs and the GPP are activated. The activated PEs cover the workload, which is required for maintaining a low speed control channel between a wireless terminal and a basestation. At the active mode, all components comprising the baseband processor are activated. The global memory is only activated at the

active mode because the Turbo decoder and HARQ, which require the global memory for the buffering of burst traffic, is only activated at this mode.

CHAPTER VI

THE ARCHITECTURE OF PROCESSING ELEMENT

In this chapter, we discuss the architecture of the PE, which comprises the proposed multiprocessor architecture for SDR. As outlined in the previous chapter (Figure 5.2), the PE consists of the parallel datapath, scalar datapath, control logics, instruction memory, and data memory. On the top of this skeleton, this chapter describes the detailed architecture of the PE and validates architectural decisions made on the PE. This thesis proposes three novel schemes on the parallel datapath for power reduction, throughput enhancement, and flexibility. Three new schemes are **macro instructions**, **macro pipelining**, and **staggered execution**.

6.1 Previous Works

Pipelining, super-scalar, and multi-threading are popular techniques for high performance GPPs [38]. However, they have significant power overhead and are not suitable for the baseband processing of the wireless terminal, which has a tight power budget. For example, Pentium-M consumes 10∼20W although it is a low power design [39]. Such low power efficiency is the result of the insufficient exploiting of the DLP, which is a plethora in the baseband processing. Briefly, for throughput en-

hancement, the pipelining and super-scalar rely on the instruction level parallelism but the multi-threading relies on the TLP. Because there exist many applications with tiny DLP, a GPP can not be optimized to exploit only the DLP.

There are numerous parallel architectures for embedded workload: CELL from IBM [40], Imagine from Stanford [41], and SCALE from MIT [42]. The CELL aims game and multimedia applications on desktop computers. The target application of Imagine is graphics. These architectures are not intended for portable applications and have much higher power budgets than we are aiming for (sub-300mW).

There have been many digital signal processing architectures for wireless communications, which employ a wide range of parallel processing styles: SIMD, MIMD, very log instruction word (VLIW), pipelining, and multi-threading.

The PicoArray of PicoChip [7] is based on the MIMD style processor array. It is the most flexible among the existing parallelization schemes because each PE has an independent control. However, its high power consumption limits its application to basestations having a relaxed power constraint. High power consumption is the result of using fine grain PEs with individual control paths. The small PE results in more power expensive communication between PEs than other architectures based on coarse grain PEs. Multiple control paths dissipate additional power when accessing instruction memory and decoding instructions. In the case of the baseband processing workload, which exhibits wide DLP, using a fine grain PE with individual control is an over specification.

The SandBluster of Sandbridge [6] is a multi-core digital signal processor whose computation engine has a narrow SIMD datapath. It supports multi-threading at the hardware level to exploit the TLP [6]. It has a strong compiler tool set that allows use of conventional ‘C’ primitives for the description of signal processing algorithms.

The ICERA solution [43] also has a narrow SIMD datapath. The power consumption is reduced by the direct cascading of CUs, which minimize the activity of a power hungry register file. However, the limitation of these architectures exists at the low utilization of the DLP. Furthermore, these systems are not scalable. Except for increasing the number of PEs, one possible way to enhance system throughput is to increase operation frequency and pipeline depth. However, it was revealed that this approach is not power efficient with nanometer technology [44].

The SODA from Michigan [8] and the EVP from Philips [9] use a wide SIMD datapath, 32 wide at the SODA and 16 wide at the EVP. For supporting the workloads with wide DLP, wide SIMD is the most power efficient among possible architectural schemes. These architectures have several SIMD units for the vector computation, data alignment, and vector reduction. In order to minimize the idle period of the SIMD datapath, which is caused by control actions such as memory address generation and loop control, the SODA and EVP have a hardware address generator and loop controller.

The EVP has a VLIW style instruction set on a wide SIMD datapath. Thus, multiple SIMD units can be executed in parallel. However, like other VLIW architecture, it requires more complex control logics as a result of using long instruction words and multiport register files for communication between SIMD units. Among the baseband workloads, the EVP only aims to cover the modulation and demodulation parts. The other algorithms for the pulse shaping and channel encoding/decoding are mapped on ASIC style special hardware. From this factor, we can infer that its power efficiency is not sufficient to cover all algorithm blocks comprising baseband operations.

Although there exist many SDR solutions as discussed in the previous section, the

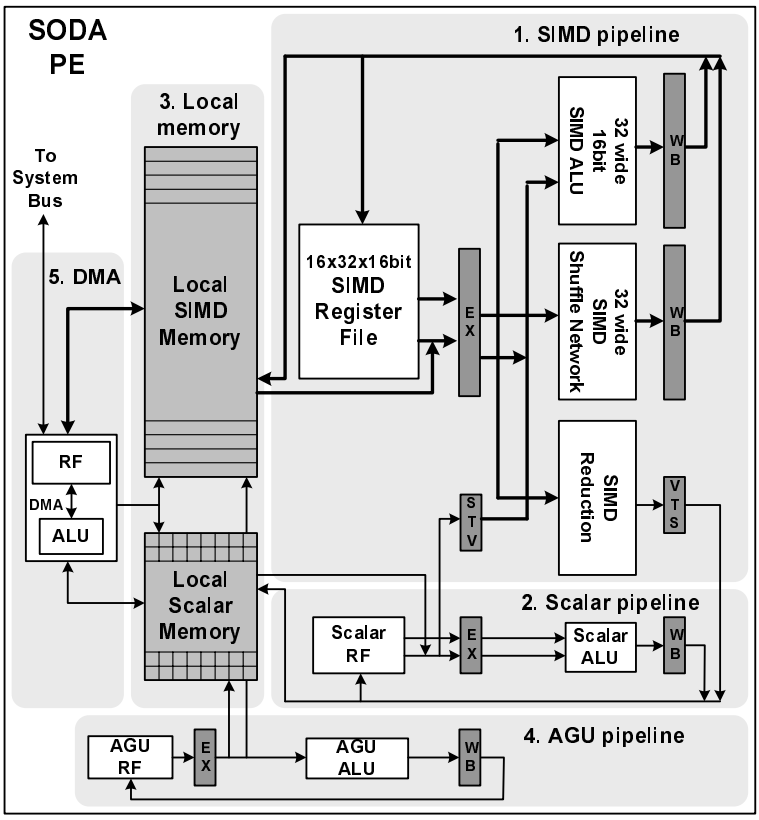


Figure 6.1: Architecture of the SODA PE

performance of these architectures is not enough to satisfy the power and throughput requirements of emerging high speed wireless networks such as WiMax and W-CDMA HSDPA. For example, Sandbridge reported that their solution implemented with 130nm technology can support 384Kbps W-CDMA packet data service with 180mW power consumption. However, the maximum data rate required by HSPDA and WiMax is several tens Mbps. As a solution, this thesis proposes a novel architecture which enables the use of programmable hardware even in the baseband signal processing of the high speed wireless terminals.

6.2 SODA Architecture

The operation of the proposed architecture is based on the three novel scheme which are motivated from the limitations of the SODA. Thus, prior to discussing the details of the proposed new schemes, this thesis explains the architecture of the SODA. It might help the understanding of the proposed novel schemes. The architecture of the SODA PE is depicted in Figure 6.1 [8].

Because the design of new architecture is based on the SODA, the SODA and new architecture have many common factors. At first, the SODA is also a chip multiprocessor architecture. It has multiple PEs and these elements are interconnected by a low speed bus. It has global memory to buffer traffic burst. Each PE has both parallel and scalar datapath. Thus, the SODA and the proposed architecture are identical in high level architecture with the reasons which we already explained in Chapter V. The SODA and the proposed new architecture differ in the detailed architecture of the parallel datapath and its control mechanism.

The parallel datapath of the SODA consists of SIMD ALU, SIMD register file, SIMD shuffle network, and SIMD reduction. The SIMD ALU performs actual computations. The SIMD register file provides input vector operands of the SIMD ALU and stores temporary computation result of the SIMD ALU as conventional register file. SIMD shuffle network performs the data alignment which is required to support data movement operation between computations. The SIMD reduction logic converts SIMD vector into scalar value. The SIMD reduction is used for the computations which require vector data as input operand and produce scalar data as output.

For the control of parallel datapath, the SODA defines reduced instruction set computer (RISC) style instruction set. As conventional RISC architecture, one in-

ADDER	MULT	Reg-RD	Reg-WR
1.44 mW	5.83 mW	0.91 mW	0.94 mW

Table 6.1: Power consumption of 16 bit datapath components which are implemented by 130 nm technology and run at 700 MHz

	d-mem.	V-reg.	V-shuffle	V-ALU	control
Viterbi	7.45%	**70.66%**	1.19%	9.77%	10.91%
FFT	17.68%	**57.62%**	4.24%	9.6%	6.3%

Table 6.2: Operation power profile of the SODA PE when it performs Viterbi decoder with K=7 and R=1/3, and 64-wide FFT operation

struction of the SODA describes primitive operations: data load/store from memory and simple arithmetic operations. Most arithmetic operations of the SODA requires two input vectors as input operands and produces one vector as output. It allows to use power efficient two read ports and one write port register file as the vector register file. In order to reduce the length of instructions, one instruction can describe the operation of one unit among the SIMD ALU, SIMD shuffle network, and SIMD reduction. As a result, only one SIMD unit can run within one cycle.

Even in the parallelizable workload, there exist some sequential actions which are related to memory address generation and loop control. To avoid idle period of the parallel datapath caused by these control actions, the SODA has hardware address generator and loop controller. Because these operations are not complex, the hardware generators and loop controller can be implemented in simple hardware.

6.3 Motivations

Based on the details of the SODA, this thesis discusses the three limitations of the SODA which lead us to propose three novel schemes. First, the SODA has high

	load/Store	alignment	computation	control
Viterbi	37.2%	4.8%	48.0%	9.9%
FFT	24.3%	25.2%	43.7%	6.8%

Table 6.3: Operation cycle profile of SIMD based architecture when it performs Viterbi decoder with K=7 and 64-wide FFT operation

power consumption in its SIMD register file. Although the power cost of the register file is smaller than other datapath components as shown in Table 6.1, its high access frequency results in high power consumption. Note that, at most operation cycles, two input vector operands are read from the SIMD register file and one output vector is written back to the SIMD register file. It is a side effect of having a RISC style instruction set in the SODA. Table 6.2 shows that, in the SODA, the power consumption of the SIMD register file occupies about 60∼70% of total datapath power. Thus, minimizing the SIMD register file access frequency is crucial for power reduction.

Second, the non computational operations, such as data load/store and vector alignment, occupy a significant portion of the total operation cycle count. Table 6.3 shows that about 50% of total operation cycles are assigned for non-computational operations in the SODA. For higher system throughput, we need to minimize the operation cycles for these non-computational operations. Furthermore, because of the super-linear relation between the power consumption of a circuit and its processing time, reserving more processing time for the computations, such as multiplications instead of the data load/store operations, will substantially reduce the system power.

Third, we observe that, in the baseband signal processing, the previously discussed macro operations, such as the data load/store, vector alignment, vector computation, and vector reduction, can be executed in parallel and the data dependency

between these macro operations is unidirectional. Therefore, it is possible to build a macro pipeline by cascading these macro operations without power consuming data forwarding logics. However, this type of parallelism was not exploited by the SODA. As we mentioned before, the SDR processor needs further improvement on its throughput and power. To meet such requirements, it is important to fully exploit all available parallelism.

Fourth, for higher throughput with low power consumption, the most straight forward approach is to extend the width of the SIMD datapath. However, at the SODA, the extension of the SIMD width is limited by two factors: the existence of workloads with low DLP and the complexity of the shuffle network. For example, the Turbo decoder of the W-CDMA system shows only 8 wide DLP. There exists a way to stack multiple operations having low DLP on a wide SIMD datapath. However, it breaks the regularity of the data movement pattern, so the shuffle network becomes more complex. The hardware complexity of the shuffle network is proportional to the number of input ports. Thus, for the extension of SIMD width without power overhead, it is important to find a way to minimize the complexity of the shuffle network with many input ports.

6.4 Novel Schemes for Power Reduction and Higher Throughput

In order to overcome the limitations of the SODA, this thesis proposes three novel schemes: macro instructions, macro pipelining, and staggered execution.

6.4.1 Macro Instruction

Macro instruction is to concatenate several primitive RISC instructions into one. Obviously it reduces the number of instructions, so the macro instructions minimize

the power overhead caused by instruction memory access, decoding, and controlling. However, it is not the major reason this thesis insists to use macro instructions. This thesis assumes a wide SIMD architecture and so the power reduction in its control path has no significant impact on the reduction of total system power. The major advantage of using macro instructions comes from reducing the number of power hungry register file accesses. One macro instruction describes a sequence of primitive operations which have mutual data dependency. By concatenating arithmetic units, it is possible to directly forward the operation results of an arithmetic unit to the next arithmetic unit without writing back and reading from the register file.

Before discussing the details of macro instructions, let us validate why the register file dominates the power consumption of a datapath. In a low power system, the difference between the power consumption of CUs and a register file is not large. Usually, low power systems operate at a low operation frequency, where the energy efficiency of a system is maximized. It allows sufficient operation time on arithmetic units, and finally it minimizes the power consumption of arithmetic units. However, the power consumption of a register file is not well scaled down by reducing operation frequency because the power consumption of a register file is dominated by the large capacitance of read/write and clock tree networks, not by operation speed. To validate this factor, this thesis performed a simple power experiment on major datapath components such as an adder, multiplier, and register file as shown in Table 6.1. From this result, we can infer how a register file dominates the power consumption of a datapath. As an example, let's analyze the power dissipation pattern of the 'add' instruction. In total, the register file consumes 2.76 mW ($= 1.91 \times 2 + 0.94$) and the adder consumes 1.44 mW for the 'add' instruction. The power consumption of the register file is almost double of the adder.

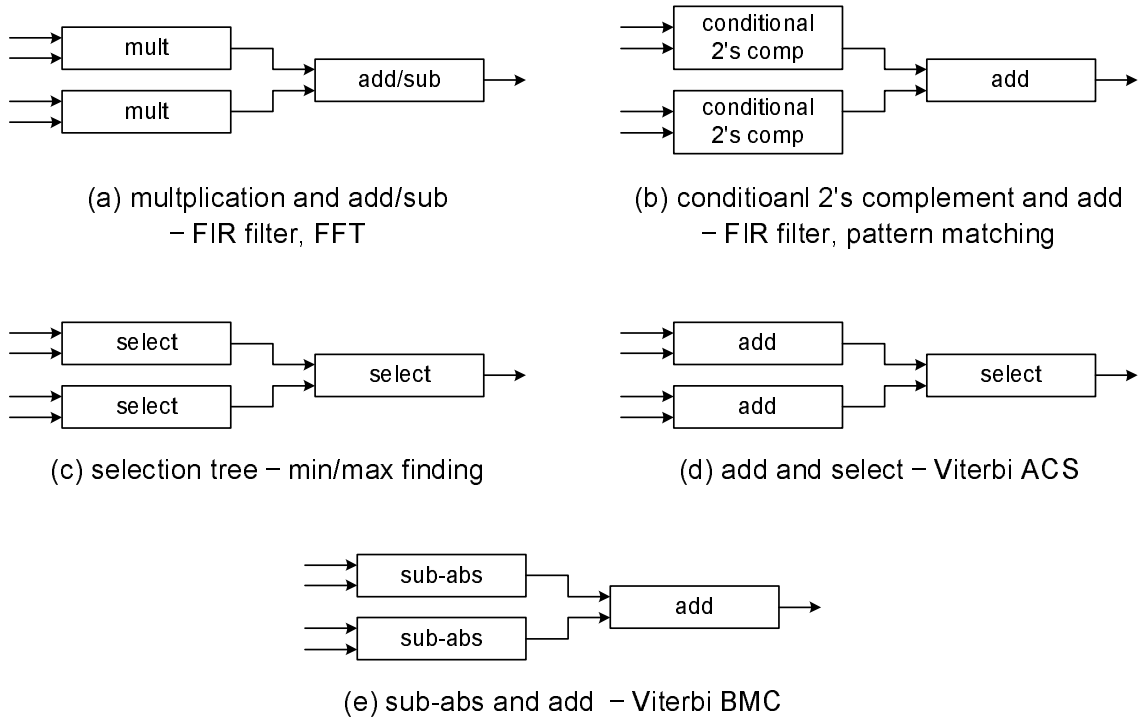


Figure 6.2: Operation of major parallelizable kernels which can be represented by macro instructions through concatenating arithmetic units

In the major kernels of the baseband workload, which were identified in Chapter III, there exist many operations which can be represented by the macro instructions. In the FIR filter, which consists of multiplications and additions (Figure 6.2(a)), the outputs of multiplications are immediately consumed by the next adder. From all major kernels, this thesis finds out the computation patterns which can be represented by macro instruction and they are depicted in Figure 6.2. Commonly, all identified operations have four input operands and one output operand. This characteristic is exploited at the design of CUs.

Arithmetic units such as adders and multipliers are concatenated through multiplexers placed between the first and second stages. For pipelining, the multiplexer

	without macro-inst.	with macro-inst.	Throughput Enhancement (%)
FIR filter	4	1	400%
min/max finding	6	1	600%
pattern matching	3	1	300%
Viterbi BMC/ACS	40	11	360%
FFT	10	7	140%

Table 6.4: Operation cycle count comparison between the datapath with macro instructions and without macro instructions

can latch its selected output data. However, as we will discuss in the experiment chapter, the power overhead of the multiplexer with latch is not substantial.

By concatenating arithmetic units, we can eliminate about $40 \sim 50\%$ of register file accesses. For example, at the multiplications and add/sub of the FIR filter (Figure 6.2(a)), 9 register file accesses (=6 reads + 3 writes) are reduced into 5 (=4 reads + 1 write). Because the power consumption of the multiplexer for the concatenation is about half of the register file access power, we can expect about a $30 \sim 40\%$ register file power reduction in the FIR filter kernel case. For other kernels, this amount of power reduction also can be expected.

In addition, the concatenation of arithmetic units results in less cycle count, in other words higher throughput. Such cycle count reduction is achieved through two factors, the simultaneous execution of two computations at the first stage and the pipelining between the first and second stages. For instance, at the 'add and select' of the Viterbi-ACS kernel (Figure 6.2(d)), two additions are performed in parallel at the first stage. At the same time, the additions in the first stage and the selection in the second stage also performed in parallel. Thus, we can expect a number about three times higher throughput, if the pipeline is fully utilized. However, the exact amount of cycle count reduction varies according to workload types because each

workload shows different pipeline utilization factors.

Table 6.4 shows the amount of cycle count reduction when the macro instructions are used. This table shows that the macro instructions result in about $140 \sim 600\%$ throughput enhancement. With cycle count evaluation, these factors are assumed: no idle periods occur by insufficient input data, and the datapath without macro instruction only has conventional RISC style instructions.

6.4.2 Macro Pipelining

The purpose of macro pipelining is to execute macro operations in parallel. Through the macro pipelining, we can expect further throughput enhancement in addition to the throughput enhancement achieved by the macro instructions. In Chapter III, this thesis identified that there exist five macro operations and they can form a conceptual macro pipeline. In this subsection, we discuss how we realize the conceptual macro pipeline in the form of actual hardware.

A straight forward way to build such a macro pipeline is to directly concatenate hardware blocks for the data load/store, vector alignment, vector computation, and vector reduction, as shown in Figure 6.3. Data memory and its read/write ports correspond to the data load/store macro operations. Vector alignment blocks are used to realize the first and second vector alignments operations. Vector computation is covered by the vector computation block.

In some low power systems, pipelining is not desirable if there exists a complex data dependency between pipeline stages, due to high data forwarding overhead. Fortunately, the major computation kernels of the baseband workload have unidirectional data dependency between macro operations with the sequence from data load, vector 1st alignment, vector computation, vector 2nd alignment (or vector

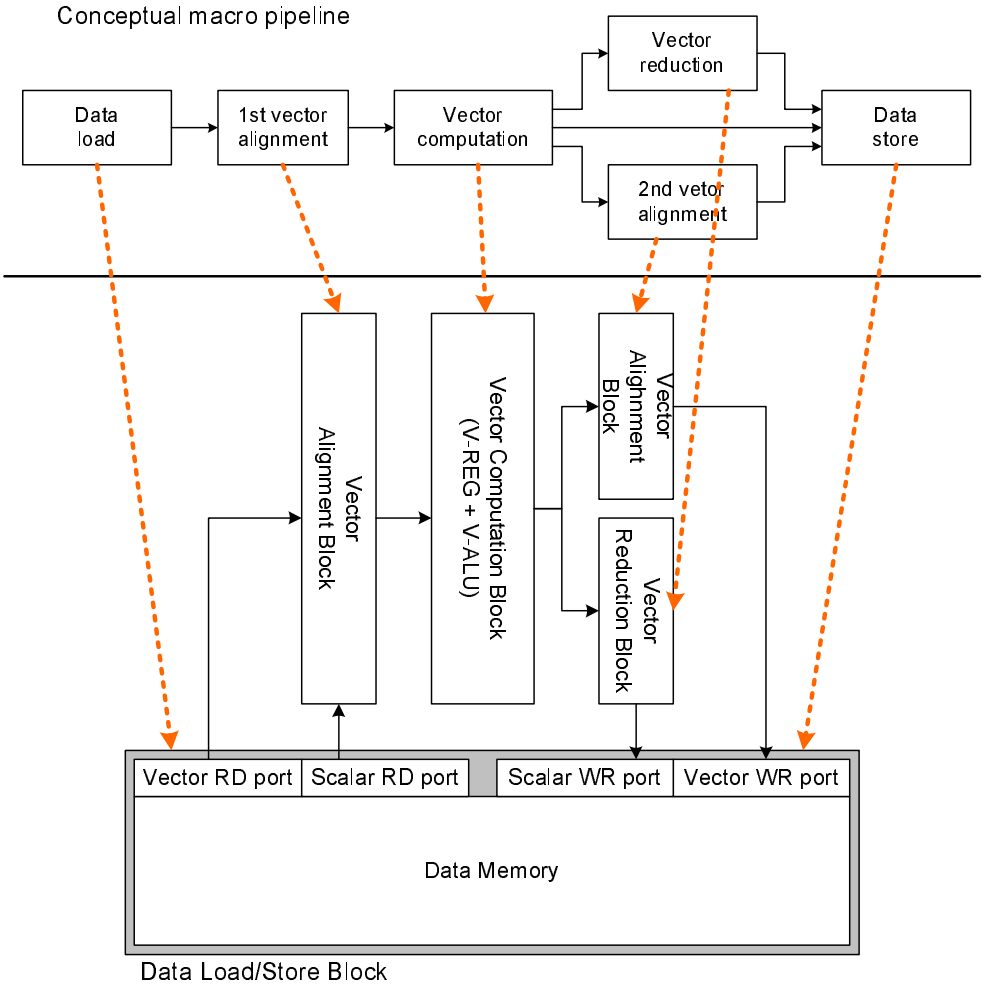


Figure 6.3: Conceptual macro pipeline and its mapping on real hardware, which consists of macro hardware blocks

reduction), and data store.

Like other pipeline schemes, the macro pipeline also needs sufficient workload to fully utilize its pipeline. According to our workload analysis results, all major parallelizable kernels can make the macro pipeline busy for long durations by continuing identical operations. For example, while performing the FIR filter kernel for the multipath searcher, the macro pipeline continues its operation for more than a thousand cycles without any idle cycles and configuration changes.

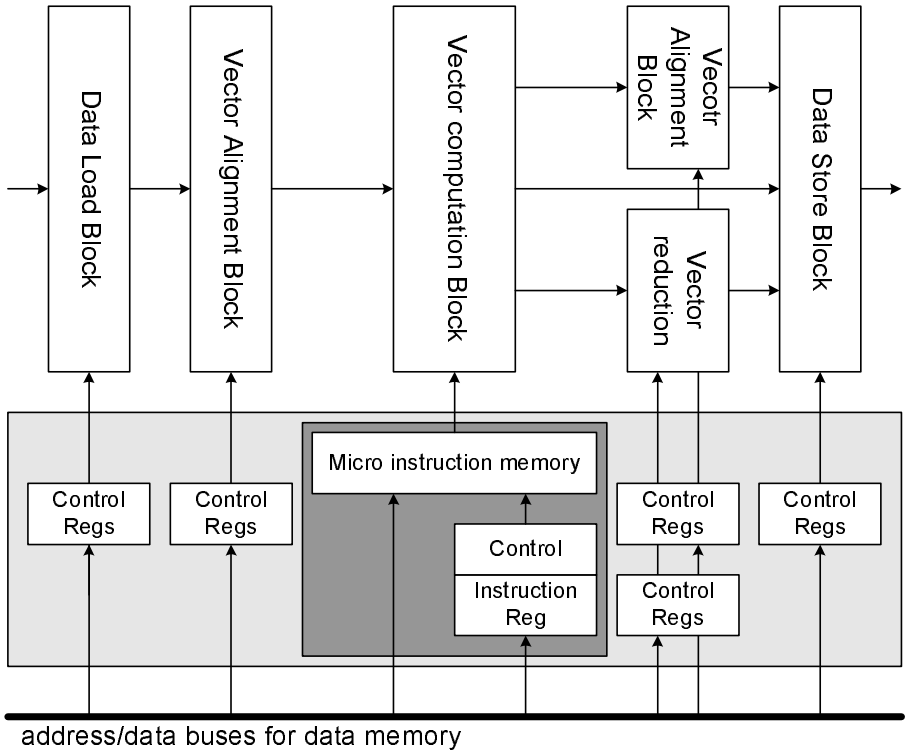


Figure 6.4: The control scheme of the macro pipeline, which uses both control registers and CISC instructions

The amount of throughput enhancement, which can be achieved by the macro pipelining, varies according to the types of algorithms because each algorithm has different workload distribution. For instance, one algorithm can require more cycles for the vector computations and another algorithm can have more complex data movement patterns. Thus, the maximum throughput of the macro pipeline is determined by the most heavily loaded block. In most cases, the vector computation block is the bottleneck of the macro pipeline. Therefore, the macro instructions, which were discussed in the previous subsection, directly improve the system throughput. It is because the macro instructions reduce the operation cycle count of the vector computation block, which is the most heavily loaded.

Another issue for the realization of the macro pipeline is the generation of its control signals. The most flexible approach is to use a VLIW style control scheme. An instruction decoder loads a long instruction word from the instruction memory, and generates control signals according to the control information, which describe the operation of all macro blocks. The EVP system from Phillips adapted this approach. However, the VLIW style control scheme induces a lot of power overhead. We simplify the control logics of the macro pipeline by exploiting a fact that the macro pipeline can be controlled like a stream line. In other words, after control information is configured, all macro blocks can continue the same operation for a sufficiently long time without reconfiguring. Thus, there is no need to load instructions at every cycle for each macro block, like the VLIW scheme.

Based on this observation, this thesis has decided to use control registers for the control of some macro blocks with a simple operation scenario. The data load/store, vector alignment, and vector reduction blocks have a simple operation scenario. For instance, the possible configurable parameters of the vector reduction block are only a computation type, input data width, and vector width. The computation type represents whether the vector reduction operation is performed through compare-and-selects or additions. As shown in Figure 6.4, the blocks for the data load/store, vector alignment, and vector reduction have control registers, which are mapped on the data memory space. By writing control information on these control registers, it is possible to configure the operation of these macro blocks.

However, the operation of the vector computation block is not simple like other blocks in the macro pipeline. Thus, this thesis applies a complex instruction set computer (CISC) style control scheme for the control of this block as shown in Figure 6.4. The reason to use CISC style instructions instead of RISC style instructions

is to minimize configuration overhead at run time. There is a control register which stores a CISC instruction. It contains the start address of micro instructions, which describe the operation of CUs. The memory for the micro instructions is also mapped on the data memory space. This mapping allows end users to define the CISC instructions. This approach simplifies the configuration procedure at run time and allows flexibility at initialization time.

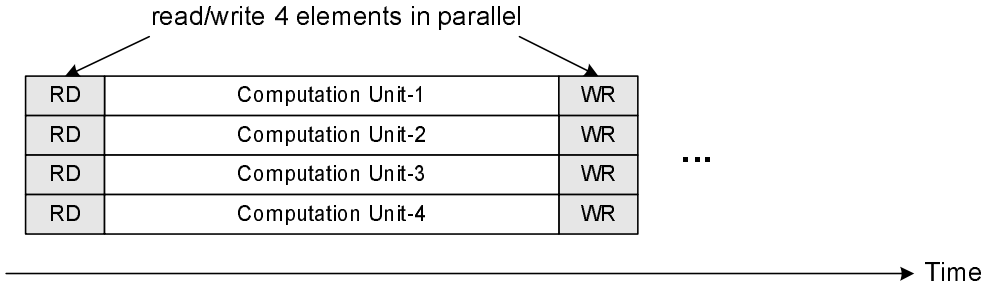
The operation of each kernel corresponds to a CISC instruction. Thus, this thesis defines five CISC instructions for describing the operation of kernels: FIR, PATTERN, MIN_MAX, VITERBI, and FFT. These instructions can be redefined by end users by rewriting the micro instruction memory.

6.4.3 Staggered Execution of Computation Units

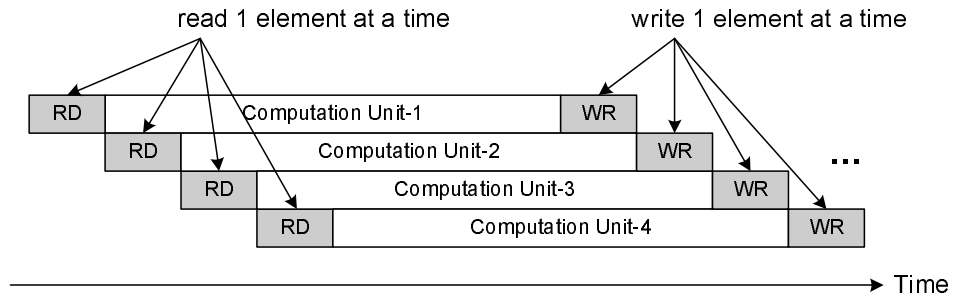
The staggered execution of CUs is to shift the operation timing of the CUs of the vector computation block as shown in Figure 6.5(b). In a conventional SIMD architecture, all CUs need their input data at the same time. However, if we stagger the execution timing of CUs, only one CU requires input data at a time. It reduces the complexity of the vector alignment block. Instead of an $N \times N$ switch, it is possible to use an $N \times 1$ switch sequentially with a time division multiplexing scheme.

It is note worthy that there is no performance degradation even after we stagger the execution timing of the CUs. It is because the number of CUs, which run in parallel, is not changed by the staggering. In the examples shown in Figure 6.5, four CUs run in parallel in both cases, the synchronous execution and staggered execution.

The staggered execution of CUs is a key scheme that allows us to exploit both the macro operation level parallelism and the DLP at the same time. As we discussed



(a) Synchronous execution of CUs

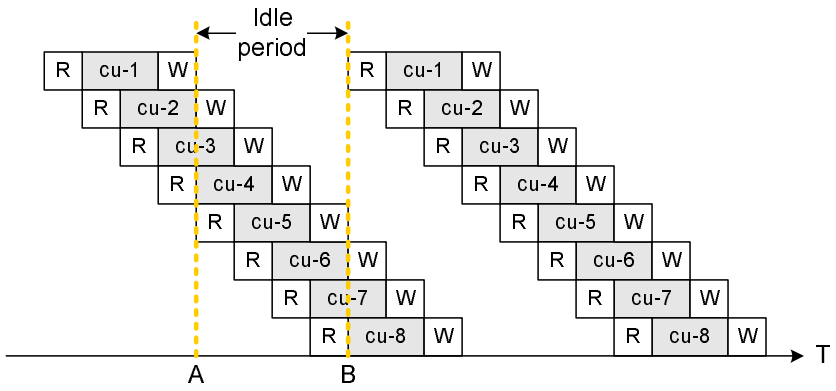


(b) Staggered execution of CUs

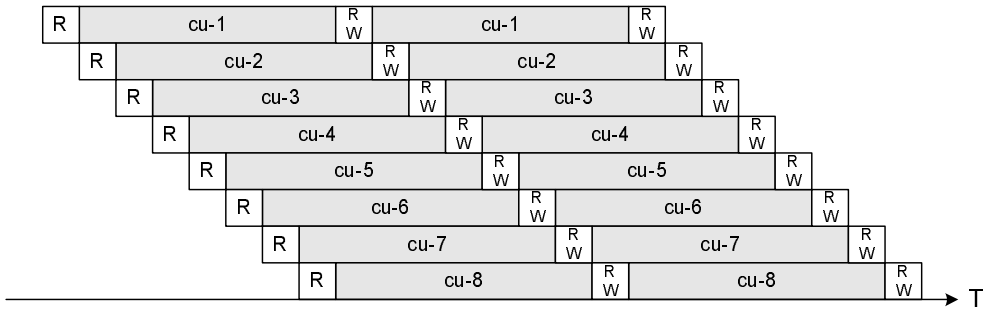
Figure 6.5: The difference on the execution timing of CUs in (a) the synchronous operation mode and (b) the staggered operation mode

before, these two parallelisms are a plethora in the baseband processing. The macro pipelining, which is proposed in this thesis as one of three novel schemes, is a way to exploit the macro operation level parallelism. On top of this, one straight forward way of exploiting the DLP is to implement a wide SIMD datapath on the macro pipeline. However, the vector alignment block will be a system bottleneck if a wide SIMD datapath is implemented on the macro pipeline without special considerations. This is because the hardware complexity of an $N \times N$ switch is super-linearly proportional to the number of input/output ports N. After a certain level, the throughput and power consumption of the switch will dominate these of the entire parallel datapath.

Due to this reason, the previous architecture only exploits a part of the available



(a) Staggering of short computations (2 cycles workload).



(b) Staggering of long computations (7 cycles workload).

Figure 6.6: Comparison of the effect on the staggering of workload having (a) short computations (b) long computations

parallelism. For example, the DXP from ICERA and the EVP from Philips implemented a narrower SIMD datapath on top of the pipelined datapath. The SIMD width of the EVP is 16 and that of the DXP is only 4. The SODA, our previous system, has a 32 wide datapath, but it did not exploit the macro operation level parallelism. However, the proposed new system uniquely implements a 32 wide SIMD datapath on top of the macro pipeline through the staggered execution of CUs.

However, the staggered execution is not always useful. For the workloads with short computation time, the staggering degrades system throughput. The FIR filter, min/max finding, and pattern matchings are major kernels with short computation

time. Figure 6.6(a) shows what happen if we stagger the operation timing of the workload with a short computation time. Although the same number of CUs are used, an idle period occurs between the active periods of a CU. With regard to the example shown in Figure 6.6(a), although the CU 1 finishes its operation at time A, it needs to be in idle state until input data is available at time B. Such idle state is the result of the resource conflict on the switches for data read and write. At the idle period (the interval between time A and B), the switches for data read and write are busy for providing input data to CUs 5∼8 and storing their computation results.

For the workload with a short computation time, the synchronous execution of CUs is better for throughput, which is shown in Figure 6.5(a). In the synchronous execution mode, we need to provide the input operands of all CUs in parallel. However, it conflicts with the design concept of the staggered execution, which is to simplify the complexity of the switches through sequential data memory access. Fortunately, all parallelizable kernels with short computation time have simple data movement patterns. The data movement pattern of the FIR filter is the vector shift and that of the pattern matching and min/max finding kernels are the vector load without alignment.

The two data movement patterns of the computation kernels with short computation time can be implemented with simple hardware by using a vector register file and banked memory in addition to the $N \times 1$ switch. At first, The vector shift data movement pattern (Figure A.20) can be efficiently implemented by a vector register. The vector register is a collection of 16bit registers. The number of 16bit registers is equivalent to the SIMD width. It supports element level shift. If the vector register is used, only an $N \times 1$ crossbar switch is sufficient for providing input operands of all CUs, which run in parallel. It is done by combining one new data element, which

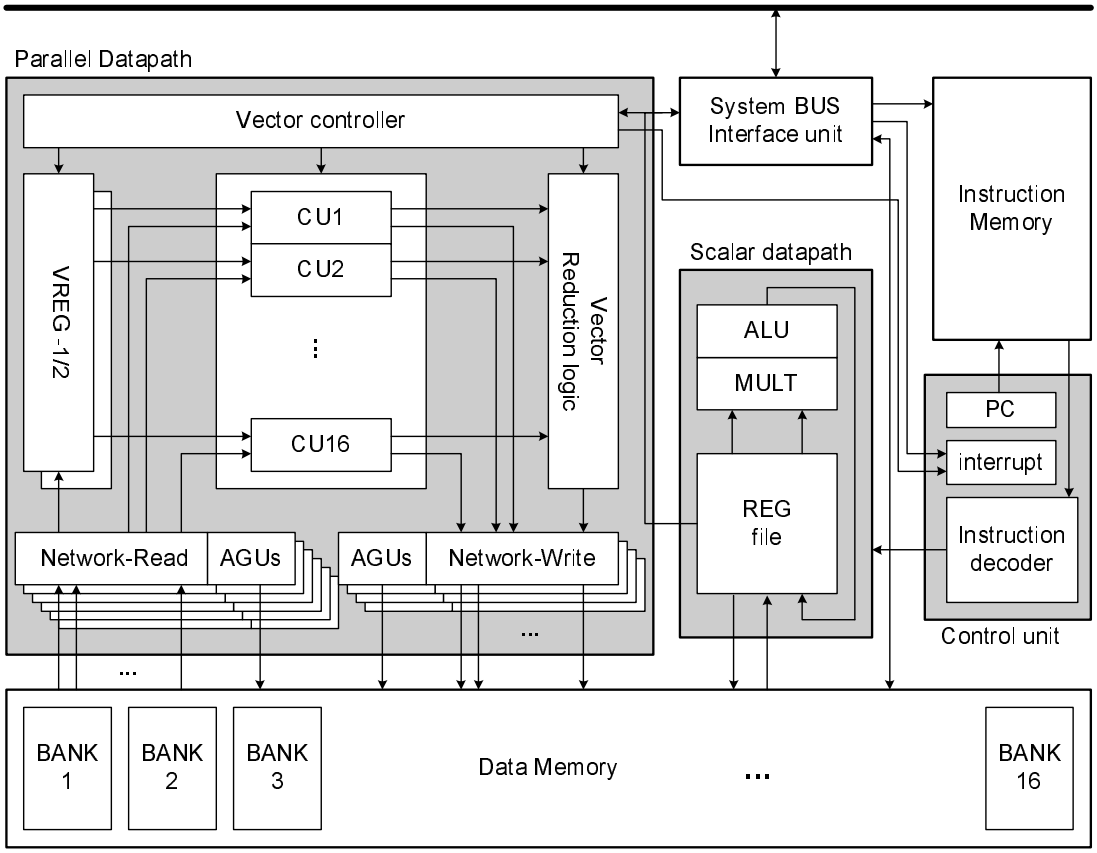


Figure 6.7: High level architecture of PE

is from the data memory, with the $N-1$ data elements, which are stored in the vector register after shift operation. Second, the vector load (Figure A.7) can be implemented by the one to one mapping between the banks of data memory and CUs.

6.5 Processing Element Design

6.5.1 High Level Architecture of Processing Element

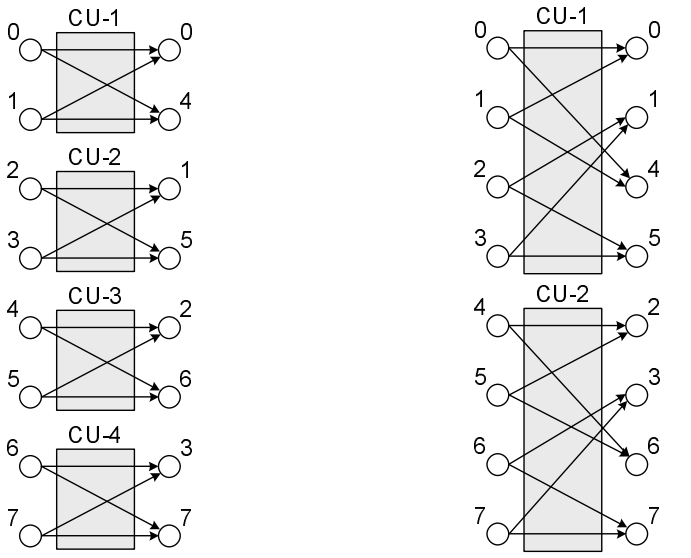
As shown in Figure 6.7, a PE consists of six units: parallel datapath, scalar datapath, data memory, instruction memory, control unit, and system bus interface unit. The parallel datapath executes the parallelizable part of the baseband algo-

rithms whereas the scalar unit performs the sequential algorithms. The control unit consists of an instruction decoder, interrupt handler, and program counter. It directly controls the scalar datapath and indirectly controls the parallel datapath. The system bus interface unit is used for the communication between PEs through the system bus.

The scalar datapath is equivalent to the datapath of a conventional low power GPP. It consists of a 32bit ALU, multiplier, and 32 entry register file. It is controlled by the control unit. The scalar datapath, control unit, and instruction memory form a four stage pipeline, which consists of instruction read, instruction decoding, register read, and execution stages. At multiplications, the execution stage is further divided into three stages. A RISC instruction set is used for the control of the scalar datapath. Thus, the data memory, scalar datapath, control unit, and instruction memory can be seen as a RISC GPP.

The parallel datapath is a kind of vector accelerator attached to the RISC GPP, which is formed by other units. In the control unit, no special instructions are defined for the control of the parallel datapath. All control actions on the parallel datapath are fulfilled through writing control information on the memory mapped control registers and instruction register. The task completion signal of the parallel datapath is passed to the control unit through the interrupt handling logic.

The parallel datapath consists of six types of components: CUs, address generators, switches, vector registers, vector reduction logic, and vector controller. All components of the parallel datapath are the realization of the macro pipeline shown in Figure 6.3. The read ports for the data memory, 6 address generators, 6 switches for read, and 2 vector registers are used for the realization of the data load and the vector alignment macro blocks. The write ports of the data memory, 4 address



(a) Assigning 2×2 ACS operation (b) Assigning 4×4 ACS operation

Figure 6.8: An example which shows the relation between the amount workload and the amount of input/output data

generators, and 4 switches for write are the realization of the second vector alignment and data write macro blocks. All components comprising the parallel datapath are controlled by the vector controller. As discussed before, the parallel datapath of a PE has two operation modes, the staggered mode and the synchronous mode. Detailed operation procedures of these operation modes are shown in the following subsection.

In the proposed architecture, the parallel datapath consists of 16 CUs, 10 address generators and 10 switches (6 for read and 4 for write). These numbers are the results of the following design procedure. The first step of the system design is to find the optimal workload of the CU. It is because other system parameters are the function of the amount of workload assigned on a CU. For instance, if we double the workload of a CU, then the increased workload demands more input data and produces more output data. Figure 6.8 is an example of such a relation. By assigning 4×4 ACS

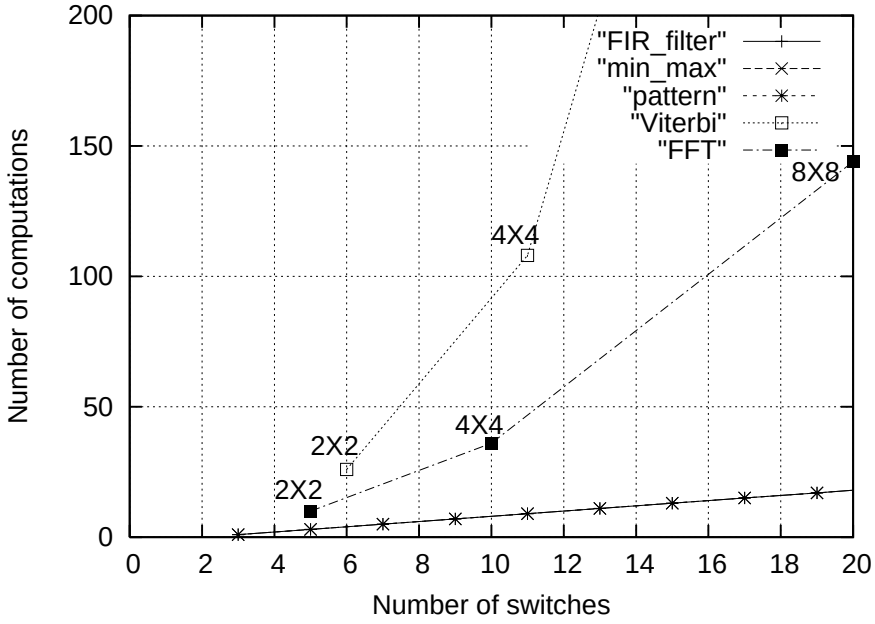


Figure 6.9: The relation between the number of required switches for parallel data load/store and the number of computations done in a CU

operation, instead of 2×2, on a CU, the number of input/output nodes is changed from 2 to 4.

Because all data load/store operations with respect to one CU are performed within one cycle in the staggered execution mode, assigning more workload on a CU results in the use of more switches and address generators for the parallel data load/store. Figure 6.9 shows the relation between the number of computations, which are done in a CU, and the number of switches, which are required to process input/output data of the assigned workload. The X-axis of this graph represents the total number of switches. The Y-axis shows the total number of computations which are induced by the assigned workload. For counting computations, primitive computations such as addition, subtraction, and multiplications are assumed. One point of this graph represents one workload mapping among the all possible workload

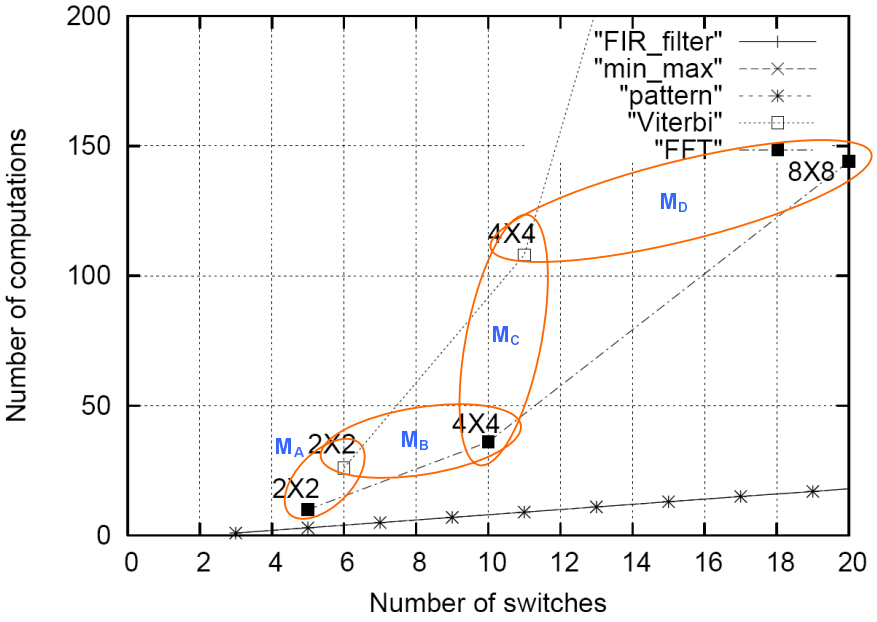


Figure 6.10: Four possible workload mapping schemes

mapping scenarios.

From this graph, we can observe several interesting factors. First, with a given number of switches, the maximum number of computations is dominated by the Viterbi-BMC/ACS and FFT kernels. So, the FIR filter, min/max finding, and pattern matching kernels have no impact on determining the computation capability of the CU.

Second, the Viterbi-BMC/ACS kernel and the FFT kernels demand different level of data load/store and computation capability. For instance, let us assume that 4×4 Viterbi-BMC/ACS and 4×4 FFT operations are assigned on a CU (mapping M_C in Figure 6.10). Then, both kernels require similar number of switches (11 in the Viterbi and 10 in the FFT). However, the required computations differ about 200% (about 120 in the Viterbi and 40 in the FFT). There exist other workload mapping scenarios which show opposite characteristics. If we assume that 4×4 Viterbi-BMC/ACS and

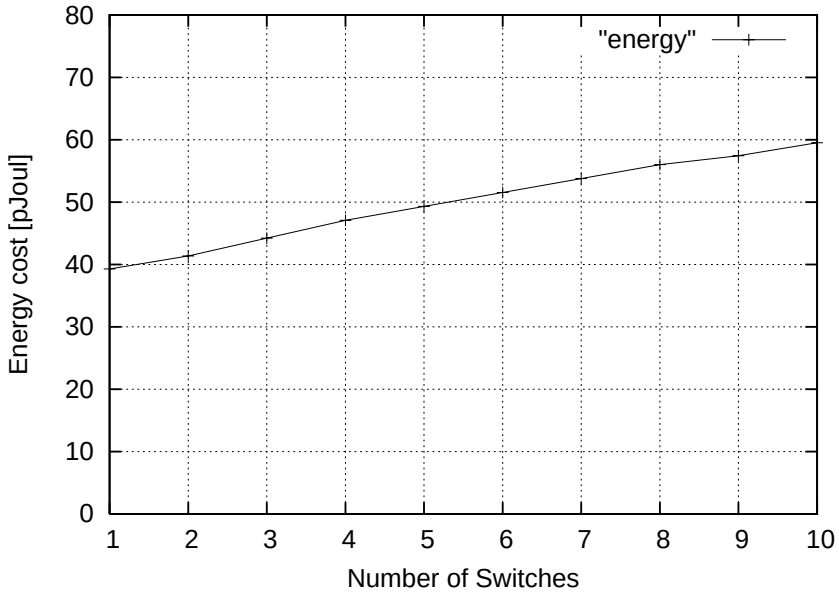


Figure 6.11: The relation between the number of switches for parallel data load/store and the energy cost of switches

8×8 FFT operations are mapped on a CU (mapping M_D in Figure 6.10), then the required computations only show about 30% difference (about 110 in the Viterbi and 145 in the FFT) but the required number of switches show about 80% difference (11 in the Viterbi and 20 in the FFT).

Based on above observation, this thesis denotes four possible workload mapping schemes into M_A, M_B, M_C, and M_D as shown in Figure 6.10. The mapping schemes M_A and M_C utilize the switches more efficiently whereas the mappings M_B and M_D maximize the utilization of the CU. From these mapping schemes, we find that it is impossible to simultaneously maximize the utilization of both switches and CUs. Thus, one remaining design issue is whether to maximize the utilization of switches or CUs.

Before making a decision, it is required to analyze the energy cost of switches and CUs. If energy cost exponentially increases, it is impossible to use the workload

mappings that require power expensive components.

We can predict that the energy cost of a CU increases substantially by assigning more workload on it. For instance, as shown in Figure 6.10, the number of required computations of the 4×4 Viterbi operation is 3 times higher than that of the 2×2 Viterbi. Thus, the CU needs to process 3 times more input traffic. For such throughput enhancement, the CU needs to use more complex and power consuming hardware components.

However, the energy cost of switch is not sensitive to its total number as shown in Figure 6.11. The X-axis of this graph shows the number of switches which can access the data memory banks in parallel. The Y-axis shows the energy cost while reading 32bit data from the data memory through an switch. According to this graph, the energy cost of adding one more switch is about 3%. Such insensitivity is due to the memory banking. Although multiple switches access data memory at the same time, they are pre-scheduled such that only one switch accesses only one memory bank. Thus, power consumption of memory bank is independent of the total number of switch which access the data memory. However, the maximum number of switches is limited by wiring. According to our analysis, about 15 networks are maximum at the silicon technology we use now[1].

Based on above discussions, we can conclude that maximizing the utilization of the CU is more power efficient. Thus, mappings M_A and M_C can be excluded from the candidate set because they inefficiently utilize CUs. From the remaining mappings M_B and M_D, we can also exclude M_D because it requires too many switches, 20. Therefore, the M_B is the most power efficient workload mapping. For the realization of the workload mapping M_B, we need to implement 10 switches and address

[1] 0.13 micron.

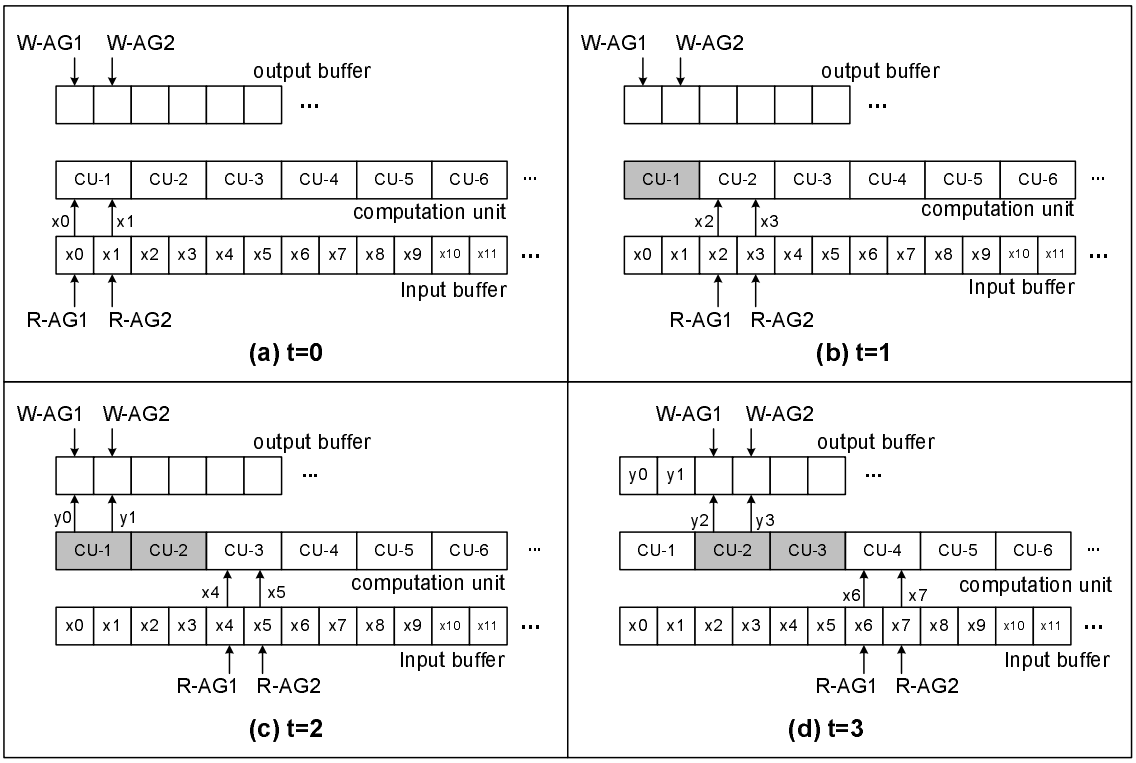


Figure 6.12: The operation of PE in the staggered operation mode when it performs the first stage of the radix-2 FFT operation

generator. More switches and address generators are assigned for the data read because most computations need multiple input data to generate one output result. Furthermore, we find that a CU can finish the M_B workload within 16 cycles by applying the macro instructions. Thus, the optimal number of CUs is determined as 16.

6.5.2 Execution Examples in Two Operation Modes

6.5.2.1 At Staggered Operation Mode

To illustrate the staggered operation mode, we step through the operation of PE when it performs the first stage of a radix 2 FFT operation. In this example, we

assume the following factors:

- A 2×2 FFT operation is mapped on a CU.
- Each CU has proper phase factor W_N values in its register.
- All input data are previously stored in the data memory and the address generators are initialized such that they point to the starting addresses of the input and output buffers.
- In order to simplify the explanation, we assume that the processing time of a CU is 2 cycles (in the actual system, it is 16 cycles).

At $t = 0$ (Figure 6.12(a)), two address generators for data read, R-AG1 and R-AG2, generate memory addresses to load two input data, x_0 and x_1, from the input buffer. At the end of this cycle, CU-1 (CU-1) latches these two input data and R-AG1 and R-AG2 increase their value to indicate the next input data, x_2 and x_3.

At $t = 1$ (Figure 6.12(b)), while R-AG1 and R-AG2 read input data x_2 and x_3 for the CU-2, the CU-1 performs its operation with the previous input data x_0 and x_1. However, there is no change on the value of address generators for data write, W-AG1 and W-AG2, until the CU-1 produces output. At the end of this cycle, CU-2 latches input data and R-AG1 and R-AG2 increase their value.

At $t = 2$ (Figure 6.12(c)), both CU-1 and CU-2 are active and the R-AG1 and R-AG2 are also busy for loading input data x_4 and x_5. At the end of this cycle, the CU-1 produces output data y_0 and y_1 and these output are stored at the memory locations where the W-AG1 and W-AG2 indicate. At the same time, the CU-3 latches input data and all address generators R-AG1,2 and W-AG1,2 increase their value.

At the steady state (after $t > 2$ in this example), the data load, data alignment, computation, and data store macro operations are performed in parallel at every cycle. If we assume N CUs, then all CUs will be active after N initial transient cycles.

6.5.2.2 At Synchronous Operation Mode

As a second example, we show the operation procedure of PE in the synchronous operation mode when it performs the 32-tap FIR filter operation. Different from the previous example, the operation timing of all CUs are synchronized. In this example, we assume the following factors:

- Filter coefficients $(c_0, \cdots, c_{31})$ and input data $(x_0, \cdots, x_{31})$ were previously loaded into vector register 1 (VREG-1) and vector register 2 (VREG-2) before $t = \tau$.

- The address generators R-AG and W-AG were initialized before $t = \tau$ such that they indicate the start addresses of input and output buffers.

- The computation times of both vector reduction block and vector computation block are assumed as 1 cycle for simplicity (in real system, the operation time of the reduction block is 3 cycles and that of the vector computation block is 2 cycles.).

At $t = \tau$ (Figure 6.13(a)), R-AG reads input data x_{32} from the input buffer and W-AG is ready to write the computation result $y[\tau]$, which is generated by the vector reduction block. At the same time, the vector computation block produces partial output $p_1[\tau], \cdots, p_{16}[\tau]$. At the end of this cycle the vector register 2 (VREG-2)

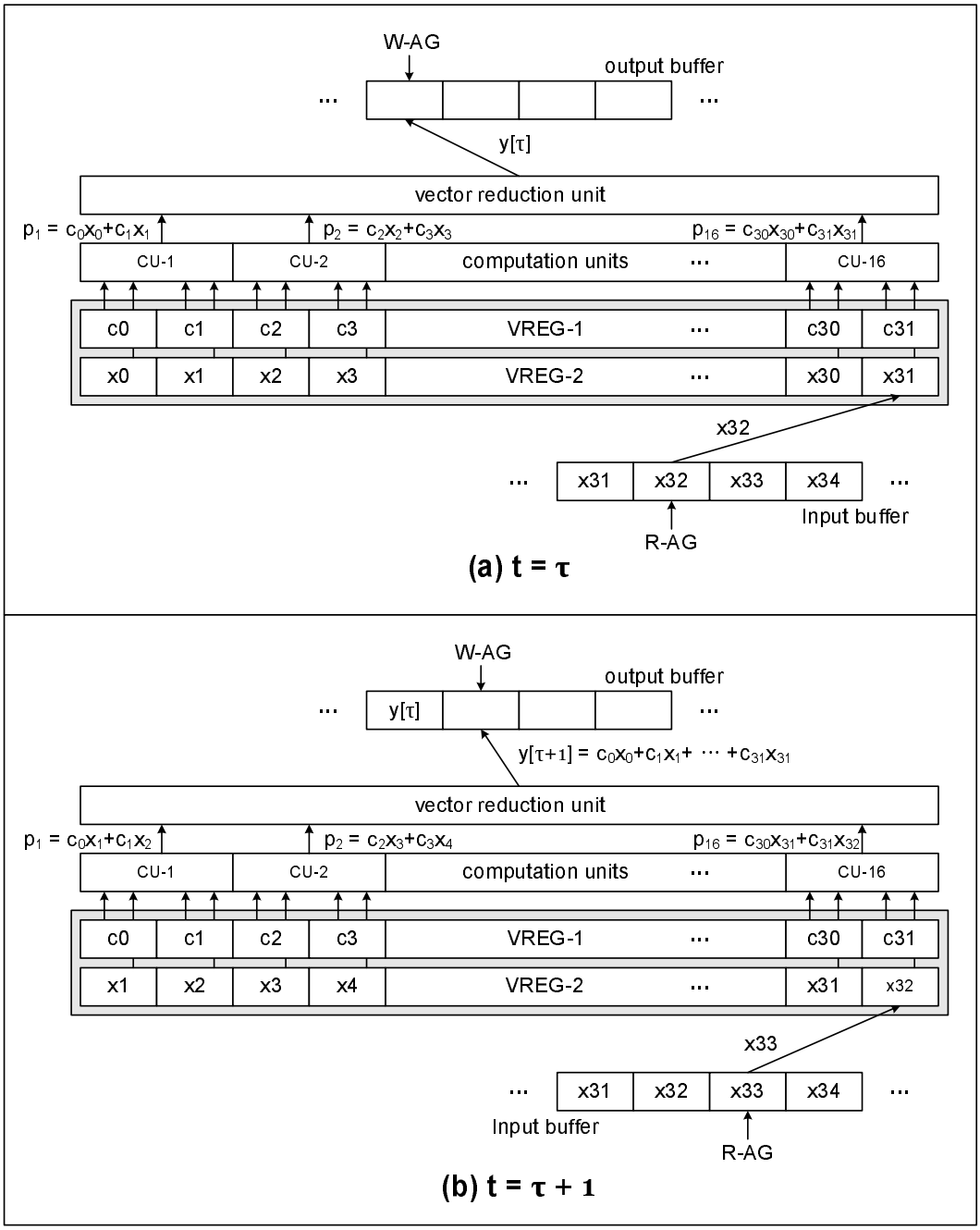


Figure 6.13: An example of the synchronous execution of PE when it performs 32-tap FIR filter operation

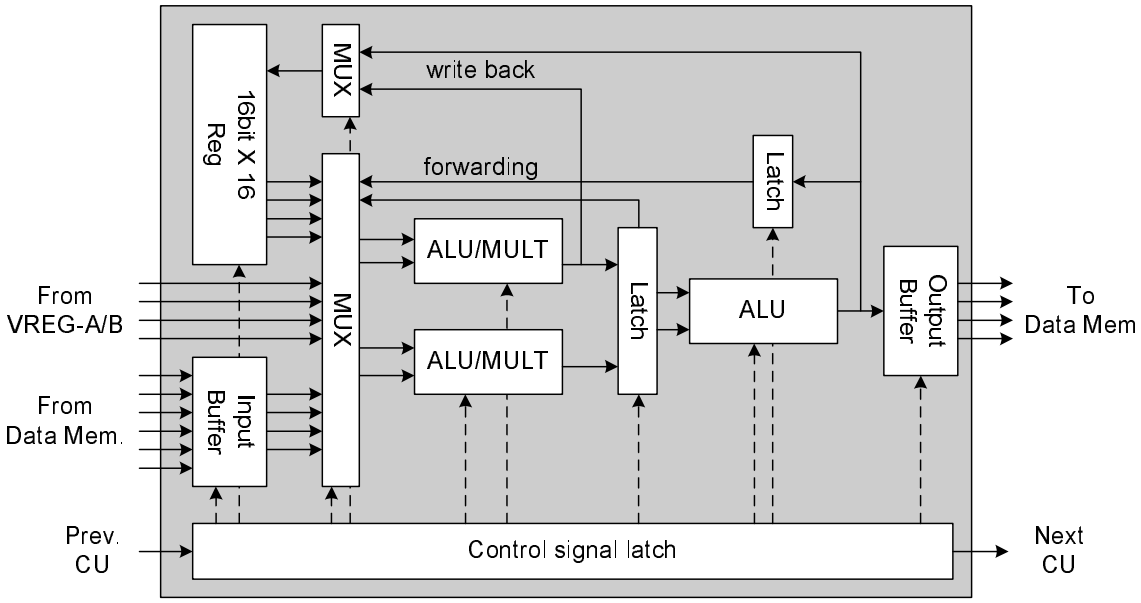


Figure 6.14: The detailed architecture of the CU

latches x_{32} after shifting its elements to the left direction and the output buffer also latches $y[\tau]$. R-AG and W-AG are increased to indicate the next address.

At $t = \tau + 1$ (Figure 6.13(b)), R-AG addresses the x_{33} and W-AG addresses the $y[\tau + 1]$. The $y[\tau + 1]$ is the aggregation result of $p_1[\tau], \cdots, p_{16}[\tau]$, which were computed in the previous cycle at the vector computation block. At the end of this cycle, the output buffer latches $y[\tau + 1]$ and the VREG-2 latches x_{33}. The values of two address generators are increased.

6.5.3 Computation Units

The detailed architecture of the CU is shown in Figure 6.14. It consists of input buffer, output buffer, register file, mux, latch, and three arithmetic units. The topology of the arithmetic units is based on the computation patterns of the major kernels, which were identified in the previous chapter. As shown in Figure 6.2, all

computation patterns of major kernels show the interconnection of arithmetic units with identical topology. For additional pipelining effect, the result of the first stage is latched. In order to avoid unnecessary register write back, the output of both stages can be forwarded via the data forwarding path. These outputs also can be written on the register file. The multiplexer reads input operands from three different sources, input buffer, vector registers, and register file. At the synchronous operation mode, the input operands come from the vector registers. At the staggered mode, they are from the input buffer.

To minimize the overhead of generating control signals at the staggered operation mode, the CU latches control signal and forwards the control signals of the current cycle to the next CU. The energy cost of the control signal latching is cheaper than that of generating control signals at each CU.

The operation of the CU is described by the instructions shown in Table 6.5. The CU instructions have both primitive instructions and macro instructions. Although only 11 instructions are listed in the table, it is possible to define additional instructions on the CU by updating the micro instruction memory.

6.5.4 Vector Reduction Unit

The operation of the vector reduction unit is to produce a scalar output data from the input vector. As discussed before, there exist two kinds of reduction operations, summation and minimum/maximum value searching. The summation can be represented by $y = \sum_{i=0}^{N-1} x_i$ where the input vector $X = (x_0, \cdots, x_{(N-1)})$ and scalar output is y. The minimum value search operation can be represented by the following equation, where the input vector is $X = (x_0, \cdots, x_{(N-1)})$ and the output

Macro Instructions	
Instruction	**Description**
DCCAD RD,R1,R2,R3,R4	Double conditional complements and addition T_1 = R1 ? R2:-R2; T_2 = R3 ? R4:-R4; RD = T_1+T_2
DMLAD RD,R1,R2,R3,R4	Double multiplications and addition T_1 = $R1 \times R2$; T_2 = $R3 \times R4$; RD = T_1+T_2
DMLSB RD,R1,R2,R3,R4	Double multiplications and subtraction T_1 = $R1 \times R2$; T_2 = $R3 \times R4$; RD = T_1-T_2
TMIN RD,R1,R2,R3,R4	Triple minimum value search T_1 = (R2>R1) ? R1:R2; T_2 = (R4>R3) ? R3:R4; RD = (T_2>T_1) ? T_1:T_2
TMAN RD,R1,R2,R3,R4	Triple maximum value search T_1 = (R1>R2) ? R1:R2; T_2 = (R3>R4) ? R3:R4; RD = (T_1>T_2) ? T_1:T_2
DBMCA RD,R1,R2,R3,R4	Double branch metric computation and addition T_1 = abs(R1-R2); T_2 = abs(R3-R4); RD = T_1+T_2
BMCA RD,R1,R2,R3	Branch metric computation and addition T_1 = abs(R1-R2); RD = T_1+R3
BMC RD,R1,R2	Branch metric computation, RD = abs(R1-R2)
ACS RD,R1,R2,R3,R4	Add, compare, and select T_1=R1+R2; T_2=R3+R4; RD=(T_1>T_2) ? T_1:T_2
Primitive Instructions	
Instruction	**Description**
ADD RD,R1,R2	Addition, RD = R1+R2
SUB RD,R1,R2	Subtraction, RD = R1-R2

Table 6.5: Macro and primitive instructions of the CU

is y.

$$y = \min_{i=0}^{N-1} x_i \tag{6.1}$$

Both min(·) and max(·) functions are based on subtraction and selection. Thus, the summation and min/max value searching operations can easily be mapped onto the same hardware. In order to increase reduction speed, it is possible to build a reduction tree, which consists of $\log_2 N$ operation steps for an input vector width N.

6.5.5 Address Generators

Address generators are used to automatically generate the next address of the data memory. Because of the deterministic address generation patterns of the signal processing kernels, automatic address generation of the data memory is not a complex task. There are two address generation modes: linear mode and page mode. In the linear mode, the next data memory address is given by the following equation:

$$y[n] = \alpha + \beta \cdot n \tag{6.2}$$

where α is an initial address and β is the difference between two adjacent data points. The linear operation mode is used for FIR filter and Viterbi-BMC/ACS. In the FIR filter operation, all address generators operate in the linear mode. However, at the Viterbi-BMC/ACS kernels, the address generators for data read only operate in the linear mode.

The address generation pattern in the page mode can be represented by the following equation:

$$y[n] = \begin{cases} y[n-1] + \alpha, & \text{if } (n \ mod \ M) \neq M-1 \\ y[n-1] + \beta, & \text{otherwise} \end{cases} \tag{6.3}$$

where M is the number of data elements in a page, α is the displacement within a page, and β is the displacement when entering a new page. The page mode is used for data write in the Viterbi-BMC/ACS and FFT kernels.

In the proposed system, 6 data are read and 4 data are written at the same time. This is implemented by 10 address generators (6 for read address and 4 for write).

6.5.6 Interconnection Networks

The role of the switch is to route a data from the data memory to the CUs and vice versa. In the parallel datapath, ten switches are used, six for data read and

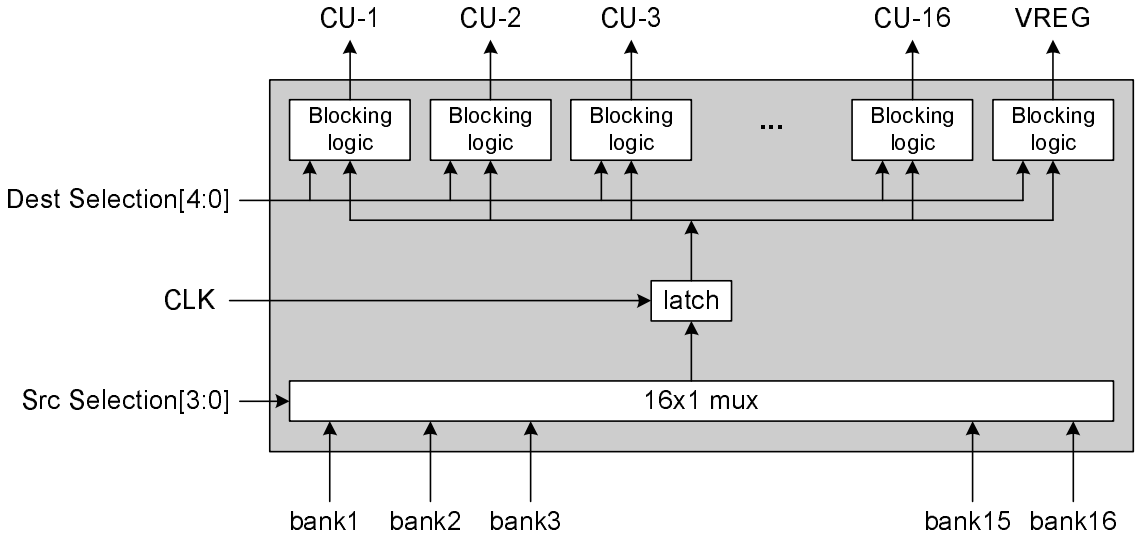


Figure 6.15: Detailed architecture of the interconnection network for data load

four for data write. One switch for data read has 16 input ports and 17 output ports. The 16 input ports are connected to the banks of the data memory with a one to one relation. The 17 output ports are connected to the CUs (16 ports) and vector registers (1 port). The switch for data write shows inverse relationship, 17 input ports from the CUs and vector reduction, and 16 output ports to data memory banks.

As shown in Figure 6.15, the switch for data read consists of a 16×1 multiplexer, latch, and blocking logics. It has two types of control inputs, source selection, and destination selection. The 16×1 multiplex is used to select the data of one bank from the outputs of 16 banks. The selected signal is latched by the latch placed between the multiplexer and blocking logics. The latch forms a pipeline on the switch. Because the switch drives long metal wires, it is inevitable to use a pipelining scheme on the switch in order to meet the timing requirement with low power. From the view

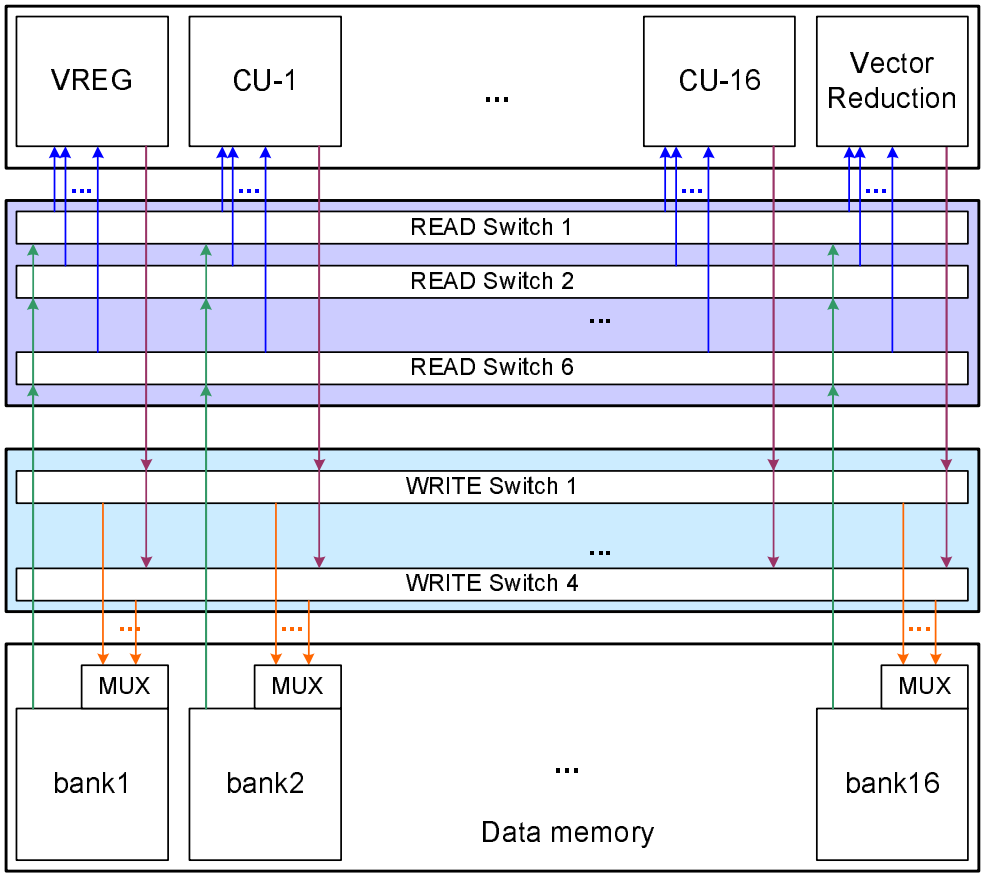


Figure 6.16: Relation between the data memory, switches, and CUs

point of system throughput, the impact of applying pipeline scheme at the switch is negligible because the parallel datapath operates in a stream fashion. The blocking logics are used for blocking glitch signal propagation. Because of the long distance, the metal wire must run between the input port of a CU and the latch of the switch. Thus, the amount of energy dissipated by the glitch signal is quite substantial. Logic AND gates are used as blocking logics instead of latches in order to avoid clocking power overhead. The structure of the switch for data store is identical to this except for the number of input/output ports.

Figure 6.16 shows how ten switches operate together. With the data read path,

the output of one memory bank forms a bus. One input port of all read networks are connected to this bus. Although there exists a loading effect, the amount of capacitive loading induced by input ports are not substantial compared to that of the long metal wire. At the data write path, there is a multiplex for the data memory, which selects corresponding input data among the output data of all write networks.

6.6 Programming Model

The programming model of the proposed architecture is based on using library routines. All detailed control actions, which are required for configuring a PE to perform major kernels, are described in library routines. The function arguments, which are passed from an application program to a library routine, determine the detailed configuration of the kernel. Thus, it is possible to hide complex hardware information from an application programmer.

The number of library routines dominates the efficiency of this programming model. If there exist too many library routines, the proper use of the library routine is not an easy task. In the case of baseband processing, there exist a limited number of parallelizable kernels. So, the number of library routines that describe the operation of major kernels is also limited.

Figure 6.17 shows an example of an application program. This program is executed in the scalar processor. The routine `vector_reg_A_load()` is a function call, which loads filter coefficients from the data memory into the VREG-A. The parameters passed to the library routine are the start address of filter coefficient, the data width of filter coefficient (8bit or 16bit), and the number of filter coefficients. The routine `FIR_filter()` actually performs the filter operation. The parameters passed to this library routine are the start address of input data buffer, the start address of

```
#define  VECTOR_WIDTH         32
#define  BYTE2                 2
#define  FIR_DOWN_SAMPLE_RATE  2
#define  INPUT_SIZE           320

short firCoef[VECTOR_WIDTH]; // filter coefficient
short inBuf[INPUT_SIZE];     // buffer for input data
int outBuf[INPUT_SIZE];      // buffer for filtered output data

int main() {

  // load filter coefficient
  vector_reg_A_load((int)firCoef, BYTE2, VECTOR_WIDTH);

  // perform filtering for input signal
  FIR_filter((int)inBuf, (int)outBuf, FIR_DOWN_SAMPLE_RATE, INPUT_SIZE);

}
```

Figure 6.17: An example of an application program for FIR filtering

output data buffer, the down sampling rate of filter, and the number of input data.

Figure 6.18 shows an example of the library routine. It contains detailed operation of the library routine for the FIR filter operation. The operation of this routine is to configure the control registers of the address generators, loop controller, and vector CUs. At last it triggers the operation of the parallel datapath. When the parallel datapath ends its operation, it asserts an interrupt signal to the scalar processor to inform the completion. Because the control registers of the parallel datapath are mapped on the memory space of the scalar processor, the write operation on a control register is performed by writing data on the data memory space. Thus, no special instructions are defined in the scalar processor for configuring the parallel datapath.

```
void FIR_filter(int idata_addr, int odata_addr, int step, int length) {

  volatile int *ptr;

  // configure address generator 1 for input data read
  ptr = (int *)AGU1_ADDR_REG_1;
  *ptr = idata_addr;
  ptr = (int *)AGU1_CTRL_REG_1;
  *ptr = AGU_4BYTE | AGU_READ | AGU_LINEAR_MODE | AGU_INC_4B;

  // configure address generator 2 for output data write
  ptr = (int *)AGU2_ADDR_REG_1;
  *ptr = odata_addr;
  ptr = (int *)AGU2_CTRL_REG_1;
  *ptr = AGU_4BYTE | AGU_WRITE | AGU_LINEAR_MODE | AGU_INC_4B;

  // configure loop controller for iteration number control
  ptr = (int *)LCU_COUNTER_REG;
  *ptr = length;

  // configure kernel type
  ptr = (int *)SIMD_FUNC_REG;
  *ptr = SIMD_FIR;

  // ignite parallel datapath operation
  asm volatile ("wait");

}
```

Figure 6.18: An example of a library routine for the FIR filtering

CHAPTER VII

POWER AND THROUGHPUT ANALYSIS

7.1 Experiment Environment and Methodology

This thesis adapts a three step approach on the evaluation of the dynamic power and throughput of the proposed architecture. The evaluation is performed at component level, kernel level, and system level. This approach reduces the amount of required computations with a minor accuracy penalty.

7.1.1 Component Level Evaluation Environment

For the component level power data, a hardware model is built by Verilog and converted into logic gates by commercial synthesis tools: Synopsys' Design compiler and Physical compiler. This thesis uses the TSMC-13 standard cell library, which relies on 0.13 micron technology. In order for the further dynamic power reduction, a low voltage and low dielectric constant (lvlk) library is used. With this library, cells run at 1 V, whereas other conventional libraries operate at 1.2 V. Low dielectric constant (low-K) material and copper are used at interconnection wires instead of SiO_2 and aluminum. Lower operation voltage reduces dynamic power about 30% by the following relation between the dynamic power and operation voltage: $P \propto \alpha \cdot CV^2$. Low-K material and copper reduce the resistance and capacitance of interconnection

wires and also reduce system dynamic power (note the C term in the above equation). Wire capacitance is also considered at the evaluation by generating a layout with Cadence's Silicon ensemble. Additionally, this thesis uses Artisan's memory compilers for the generation of storage components such as the register file and data memory. The gate level dynamic power is evaluated by Synopsys' PrimePower.

7.1.2 Kernel Level Evaluation Environment

For the kernel level evaluation, this thesis develops a power evaluation program. The power evaluation program uses the assembly routine and component level power information as input data. The assembly routine was coded by hand. Because the number of the parallelizable kernels is limited, hand coding is not difficult work. The component level power data was computed in the previous step.

The evaluation program decodes an instruction and identifies the system components, which participate in the execution of this instruction. The evaluation tool computes the power consumption of the decoded instruction by aggregating the power consumptions of all related system components. Because there are no dynamic components such as cache, the instruction level decoding is sufficient to identify the system components related to an instruction. Same decoding procedures are iteratively performed on all instructions describing the operation of a kernel. Whole steps are performed against all major kernels. As a result, this thesis obtains the power consumption data and cycle counts of major kernels such as the FIR filter, min/max finding, pattern matching, Viterbi BMC/ACS, and FFT.

7.1.3 System Level Evaluation Environment

Finally, this thesis analyzes the operations of wireless protocols and splits them into a collection of major kernels. Because the power consumption and the cycle

counts of the major kernels were already identified in the previous step, it is possible to compute the system level throughput and power data by aggregating the results of the kernels.

7.2 Kernel Level Analysis

In this section, this thesis compares the throughput and power consumption of the proposed architecture with the our previous architecture SODA [8]. For evaluation, this thesis also builds the kernel level evaluation program for the SODA. Two evaluation programs for the SODA and the proposed architecture share the same circuit level components. This thesis assumes the following factors:

- **A1:** One PE is concerned while analyzing the throughput and power consumption.
- **A2:** The PE operates at 1V and 700MHz[1].
- **A3:** While measuring the power, the PE is assumed as fully loaded. It means that the PE does not enter an idle period due to insufficient workloads.
- **A4:** While measuring the throughput, 10% idle period[2] is assumed in order to consider the configuration time and transient period.

7.2.1 FIR filter

This thesis assumes FIR filter with 32 taps, down sampling rate of 2, 16bit filter coefficient, and 16bit input data. This is the configuration of the W-CDMA pulse shaping filter. The throughput is measured by counting the number of output data per second. The energy is measure by accumulating the energy dissipated for generating one output data. While measuring the energy, the SODA and the

[1]This operation frequency will be validated at the subsection on the system level analysis.
[2]This assumption has no effect on the power evaluation.

FIR filter					
Energy (pJoul/output)			**Throughput** (Output/Sec)		
Proposed	SODA	Ratio	Proposed	SODA	Ratio
660	930	71%	6.3E+8	1.6E+8	400%

Table 7.1: Comparison of the throughput and energy consumption of the proposed architecture with the SODA, when they perform the FIR filter kernel

proposed architecture perform the identical task. However, the completion times of two architectures are not same because of their throughput difference.

Table 7.1 shows that the proposed architecture consumes about 30% less energy than the SODA. At the same time, the proposed architecture produces 4 times more output than the SODA. Most of the energy savings come from two factors. The first is the use of macro instructions, which minimizes the number of the power consuming register file accesses. The second is the low control overhead because there is no decoding overhead in the proposed architecture after one time configuration. About 400% throughput enhancement is the result of adapting the three novel schemes: the macro instructions, macro pipeline, and staggered execution. The proposed architecture can produce the filtered output data at every cycle. However, the SODA requires four cycles for the generation of one output because it sequentially performs the vector load, vector multiplication, and vector reduction operations.

7.2.2 Pattern Matching

This thesis assumes 256 binary patterns with the length of each pattern being 16. One element of the pattern is 8bit data. This configuration is used for the initial synchronization of the W-CDMA system. The throughput is measured by the number of correlation results per second. The correlation is computed between one unknown pattern and the 256 reference patterns. The searching procedure, which is

Pattern Matching					
Energy (pJoul/output)			**Throughput** (Output/Sec)		
Proposed	SODA	Ratio	Proposed	SODA	Ratio
250	535	47%	6.3E+8	1.6E+8	395%

Table 7.2: Comparison of the throughput and energy consumption of the proposed architecture with the SODA, when they perform the pattern matching kernel

to find a pattern with the highest correlation value, is not evaluated because it can be covered by the min/max finding kernel.

The proposed architecture consumes about 50% less energy and processes 4 times more input data than the SODA. The major reason of such energy savings and throughput enhancement are identical to the FIR filter case. One difference is short pattern length. In order to process multiple patterns simultaneously, the SODA uses shuffle networks for the data alignment and it results in an additional throughput and power degradation.

7.2.3 Minimum/Maximum Finding

min/max finding					
Energy (pJoul/output)			**Throughput** (Output/Sec)		
Proposed	SODA	Ratio	Proposed	SODA	Ratio
747	1121	67%	1.6E8	12.5E8	125%

Table 7.3: Comparison of the throughput and energy consumption of the proposed architecture with the SODA, when they perform the minimum/maximum finding kernel

This thesis assumes an input data as 64 elements 8bit data. This computation pattern appears when the multipath searcher finds a correlation peak among the correlation results or the Viterbi decoder finds a minimum path matric among the path matrices of all survival paths. The configuration used for the evaluation is

from the Viterbi decoder of the convolutional codes with K=7, which is used for the IEEE802.11a traffic channel. The throughput is measured by the number of min/max finding results per second.

In this kernel, the proposed architecture dissipates 35% less energy and processes 25% more data. The throughput enhancement is low compared to two previous kernels because the macro pipeline is not fully utilized at this kernel. The parallel datapath has 16 CUs and one CU can process 4 input data in every cycle. For full utilization of the parallel datapath, it is required to load 64 data (=16×4) elements in every cycle. However, in the proposed architecture, the maximum width of the vector load is only 16. Thus, the proposed architecture produces one min/max comparison result for every 4 cycles.

7.2.4 Viterbi-BMC/ACS

Viterbi-BMC/ACS					
Energy (pJoul/ouput)			**Throughput** (Output/Sec)		
Proposed	SODA	Ratio	Proposed	SODA	Ratio
48160	65970	73%	4.92E6	2.5E6	195%

Table 7.4: Comparison of the throughput and energy consumption of the proposed architecture with the SODA, when they perform the Viterbi-BMC/ACS kernel

As a workload, this thesis assumes the Viterbi decoder of the convolution code with constraint length K=9 and code rate R=1/3. This is the configuration of W-CDMA system voice channel. The throughput is measured by the number of decoded bits per second.

The proposed architecture consumes about 25% less energy and shows about 2 times higher throughput than the SODA. Most of the energy savings come from less register file access as with the other kernels. Because the Viterbi-BMC/ACS kernel

is the most dominant kernel, its power savings have significant impact on the system power.

7.2.5 FFT

FFT					
Energy (pJoul/output)			**Throughput** (Output/Sec)		
Proposed	SODA	Ratio	Proposed	SODA	Ratio
1363	1722	79%	2.52E8	9.83E7	255%

Table 7.5: Comparison of the throughput and energy consumption of the proposed architecture with the SODA, when they perform the FFT kernel

As a workload, this thesis assumes 1024 points complex FFT with a 32bit complex number (two 16bits for real and imaginary parts) as input data. This is one of the configurations of the mobile WiMax system. Throughput is measured by the number of transformed complex number output data per second.

The proposed architecture consumes about 20% less energy and processes about 250% more input data compared to the SODA. In the FFT kernel, the power saving is achieved by reducing data memory access, not register file. By mapping radix-4 FFT operation on a CU, we reduce the number of the data memory access and it results in low power.

7.3 System Level Analysis: W-CDMA 2Mbps Workload

7.3.1 Optimal Active Mode Operation Frequency

In this subsection, this thesis explains why the system runs at 700 MHz. Figure 7.1 shows the relation between the operation frequency of the proposed system and the system energy per cycle. The system energy per cycle is computed by dividing the total energy for a given workload with total cycle count. The workload used

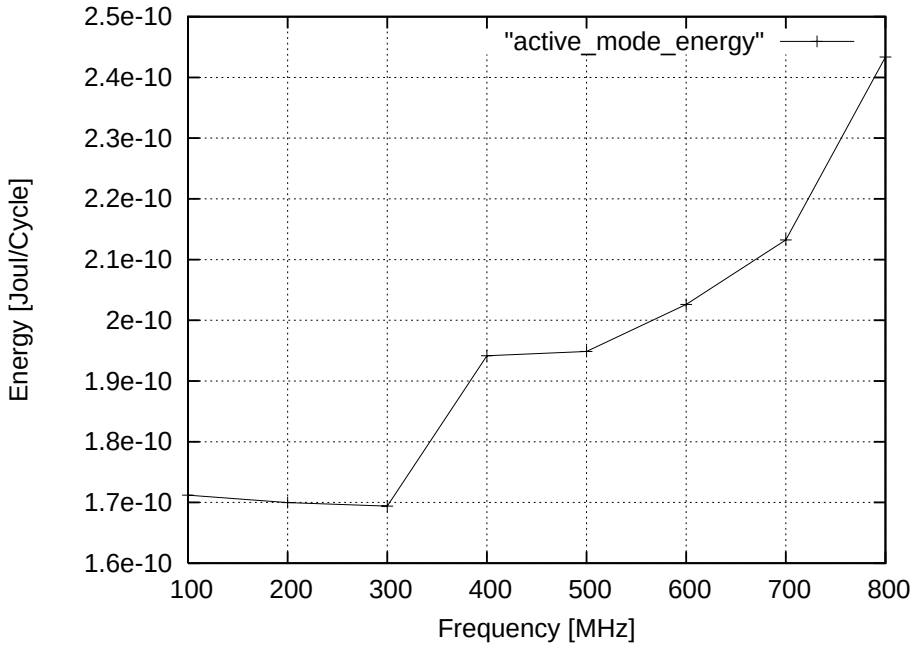


Figure 7.1: The relation between the operation frequency and the system energy per cycle, when the system provides the W-CDMA 2Mbps packet data service

for the energy measurement is the W-CDMA 2Mbps packet data.

Naturally, the system energy per cycle value increases according to the speed up of the system operation frequency. With the frequency range below 300 MHz, the system energy cost is insensitive to the operation frequency change because same library cells, which have identical energy cost, are used within this range. When synthesizing gate logics with hardware description language input, a logic synthesis tool adaptively selects library cells with difference size according to the timing requirements for the power saving. Small cells are used for slow operations because they consume less power than large cells due to their low capacitive load. The reason of such constant system energy cost is that, from 300 MHz, the synthesis tool begins to use library cells with minimum size. Although more processing time

is allowed by decreasing operation frequency below 300 MHz, the synthesis tool has no other option to exploit additional processing time.

From 400 MHz, the system energy cost increases because the synthesis tool starts to use larger library cells from this point to satisfy the timing requirements. From 700 MHz, the system energy cost starts to increase rapidly.

In this graph, although 300 MHz is the most energy efficient, this thesis selects 700 MHz as the optimal operation frequency because it is possible to increase system throughput about 230% (=700/300) with about 25% ($=\frac{(2.13-1.7)}{1.7}$) additional energy overhead. While processing W-CDMA workload, simple components, such as adders and subtractors, are frequently used. Because the energy cost of these components increases gradually, the system energy cost only increases 25% within 300 MHz and 700 MHz range. If such throughput enhancement is not required, then the 300 MHz can be a possible operation frequency.

7.3.2 Idle Mode Support

In this subsection, this thesis discusses whether designing additional special hardware is required for only the idle mode operation. As this thesis previously discussed, the power consumption in the idle mode is important for the extension of terminal standby time. Because of this reason, we proposed the architecture of an idle mode processor [45]. However, this thesis finds that the proposed PE can also be used for the idle mode with about 27% ($=\frac{(2.26-1.78)}{1.78}$)energy overhead. If we further consider that other components, such as the RF frontend circuits, A/D converter, and LCD display, also participate in the idle mode operation, the impact of 27% energy cost on the terminal standby time will not be substantial.

As we previously discussed in Chapter III, the workload of the idle mode operation

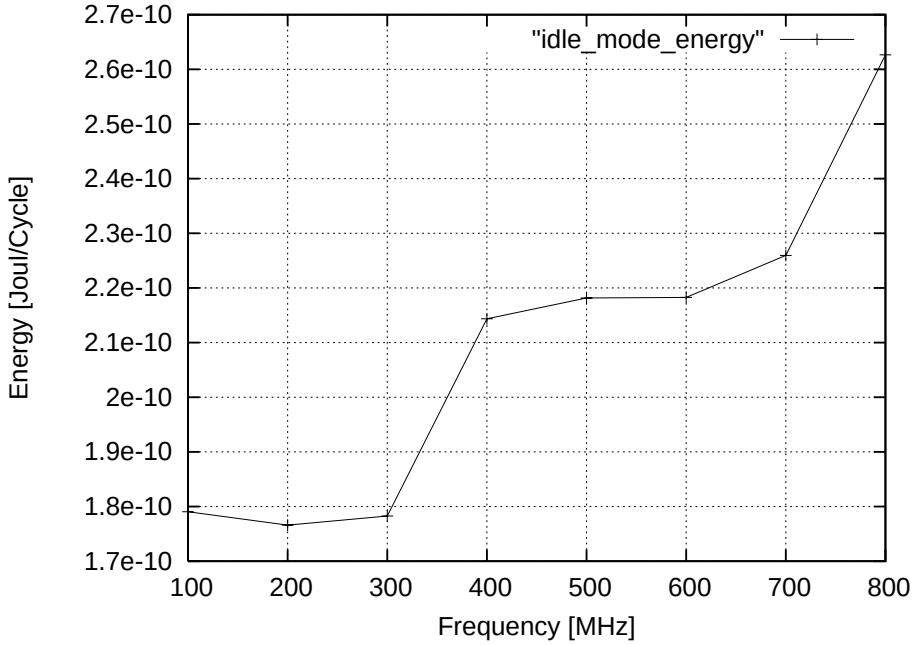


Figure 7.2: The relation between the operation frequency and the system energy per cycle, when the system is in the W-CDMA idle mode

is dominated by the FIR filter kernel. The proposed architecture was designed to optimally support the five major kernels, which include the FIR filter as one of them. Thus, from the architectural view point, the proposed architecture is already optimized to the idle mode workload.

One remaining problem is to validate the energy cost of running at the non optimal operation frequency, 700 MHz. The PE will be synthesized at 700 MHz in order to support the active mode operation with maximum throughput. However, according to our previous workload analysis result, about 50MHz operation frequency is sufficient to meet the timing requirement of the idle mode workload [45].

Figure 7.2 shows the relation between the system operation frequency and the system energy per cycle in the idle mode of the W-CDMA terminal. Because there is no energy cost change between 100∼300 MHz, it is possible to increase operation

		Peak Power(mW)	**Average Power**(mW)	**Leakage Power**(mW)	**Area** (mm^2)
Memory	Data memory	60.8	17.2	1.41	3.7
	R/W Network	85.92	24.3	0.18	0.1
	Address gen.	27	9.3	0.10	0.1
Vector CU	REGs	88.6	8.7	0.25	0.4
	MULTs	185.6	6.9	0.42	0.5
	ALUs	44.8	10.8	0.43	0.4
	MUX	28.8	2.7	0.10	0.0
Vector Registers		10	0.5	0.05	0.0
Vector Reduction		14	0.5	0.04	0.2
Vector Control		12.8	8.3	0.01	0.2
BUS I/F		4.5	0.5	0.10	0.1
SCALAR	REG	4.8	3.0	0.06	0.1
	MULT	21.8	3.0	0.04	0.1
	ALU	4.5	2.2	0.00	0.1
	Control	2.0	1.4	0.01	0.1
	I-MEM	21.3	15.0	0.70	1.8
Miscellaneous		-	34.3	-	2.4
Total(rounded)		-	150	4	11

Table 7.6: Dynamic power consumption of the proposed architecture when it provides W-CDMA 2Mbps packet data service

frequency up to 300 MHz without additional overhead. Based on this result, let us compute the energy cost of running at 700 MHz. In the graph, the energy cost of running at 700MHz is about 27% higher than the energy cost at 300 MHz. As we discussed, this amount of energy overhead has no significant impact on total system energy.

7.3.3 Component Level Breakdown

Table 7.6 shows the component level power breakdown result of the proposed architecture when it performs W-CDMA 2Mbps packet data service. The peak power shown in the second column is the power dissipated when the corresponding blocks are activated at every cycle with maximum throughput. It is also assumed that the

half of input signals toggles at every cycle. The average power shown in the third column is the result of considering the activity ratio of the components. Thus, if the average power is half of the peak power, then it means that the corresponding component is activated at every two cycles on average. In order to consider the effect of the global clocking and interconnection overheads, this thesis assumes 30% power [46] and area overheads and it appears in the table with the name 'Miscellaneous'. It is because the hardware model was built at the component level, which does not include the overhead of clock and global interconnection networks. After considering all factors, we compute that the proposed architecture consumes only 150 mW while supporting W-CDMA 2Mbps packet data service

In this table, we can find two interesting results. At first, in the vector CUs, the power consumption of the register file is successfully reduced by the use of macro instructions. Its power consumption is only 80% of the ALU. Although we replace the register file with the multiplexer and latches, their power overhead is negligible (less than 2%). Second, the power overhead of the multiplier is not substantial. The algorithm, which intensively uses the multiplier, is the FIR filter. However, with the W-CDMA workload, the FIR filter only occupies less than 5% of total operation time. However, if we consider the OFDMA workload, the power consumption of the multiplier can be substantial.

7.3.4 Comparison with SODA

We compare the power consumption of the proposed architecture with the SODA when they process W-CDMA 2Mbps packet data workload. For fair comparison, we apply the evaluation methodology used on the proposed architecture on the SODA. Thus, the power consumption and throughput of the SODA were evaluated through

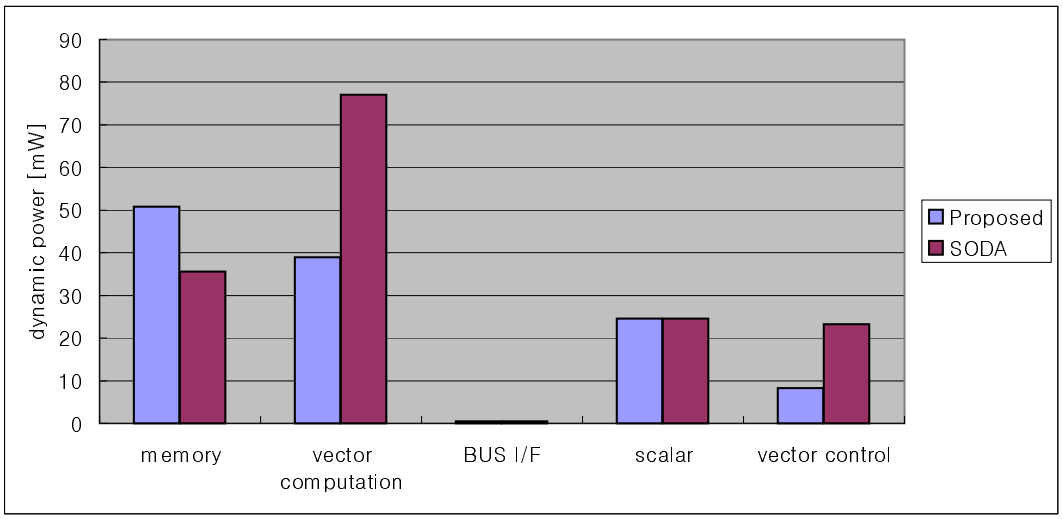


Figure 7.3: Power comparison between the SODA and the proposed new architecture when they support W-CDMA 2Mbps packet data service

the three level approach and the TSMC-13 lvlk library cells were used at the logic synthesis.

As a result, we find that the SODA dissipates 200 mW at the W-CDMA 2Mbps packet service workload whereas the proposed architecture consumes 150 mW. At the same time, the proposed architecture shows about 2 times higher throughput compared to the SODA. About 25% of system level power saving and two times higher throughput are mostly from the efficient processing of the Viterbi-BMC/ACS kernel which mainly comprises the turbo decoder workload. As discussed in Chapter IV, the turbo decoder dominates the baseband processing workload.

Figure 7.3 shows the power consumption profile of the proposed architecture and SODA. In X-axis, the "memory" represents the power dissipated for data load/store from/to memory. The "vector computation" is the power for actual vector computations. This term includes the power consumption of arithmetic units and register files. The "scalar" represents the power consumption for the scalar workload. The

"vector control" represents the power dissipated for vector control signal generation.

Two architectures show identical power consumption for system bus interface and scalar workloads. It is because two architectures have the same amount of input/output data and scalar workload. The proposed architecture dissipates less power for the vector computations and the vector control signal generation. The less vector computation power is the result of applying the macro instructions. Because, in the proposed architecture, the parallel datapath runs in stream fashion, control instructions are not decoded at every cycle. So, the generation of vector control signals results in less power overhead. However, while accessing memory, the proposed architecture dissipates than the SODA. It is because data load/store operations are performed through switches, which cover data alignment operations, in the proposed architecture. However, the power gains of the vector computation and vector control generation operations surpass this overhead.

7.4 Future SDR Solution

In this section, this thesis predicts the future of SDR solution. Wireless communication systems will continue their evolution to support higher data rate. The amount of computation demanded by future wireless protocols keeps super linear relation with maximum data rate. It is because wireless channels are quite limited and expensive resources. For higher spectral efficiency, more complex and computation intensive algorithms will be adapted in future wireless communication protocols. To cope with such evolution, the throughput and power efficiency of SDR system also must be enhanced.

Before discussing the future SDR system in detail, let us recall an equation that

explains the relation between system dynamic power and circuit parameters.

$$P_{dynamic} \propto C_e V^2 \tag{7.1}$$

This equation explicitly shows that the system dynamic power can be saved by reducing the effective capacitance of system circuits C_e and system operation voltage V.

Generally, there exist three schemes that can reduce system dynamic power. First is to shrink device size. Due to the progress on silicon technologies, the gate length and operation voltage of CMOS circuits have been reduced. This change directly causes the reduction of system dynamic power. For instance, the migration from 180 nm silicon technology to 130 nm technology results in about 70% dynamic power reduction. Second is to exploit available parallelism. Parallel executions allow longer processing time to each arithmetic unit. Because slower circuits operate at lower voltage and induce less capacitive load, the parallel execution can result in dynamic power saving. The macro pipelining and staggered execution of CUs, which were proposed by this thesis, reduced system dynamic power by maximally exploiting DLP. Third is to minimize effective capacitance of system by using less capacitive components even for identical operation. For instance, the macro instructions proposed by this thesis reduced system power by minimizing register file access, which induces high capacitive load. The dynamic power of most digital hardware system has been minimized successfully by applying these three schemes.

However, in near future, the shrinking down device size will not be an effective way to reduce system dynamic power because the operation voltage of devices will not be scaled down along with device size reduction in order to avoid high leakage power. Due to this phenomenon, reducing the effective system capacitance C_e in

Equation (7.1) becomes more important in the system dynamic power reduction.

One way to reduce system capacitance is to use customized logics which are optimal to applications. The customized logics are also required in order to fully exploit the DLP for power reduction. Under a constant operation voltage, performing more complex operations within one cycle is only possible way to exploit the additional operation time achieved by concurrent executions after using minimum size logic gates. Thus, for dynamic power reduction, customized logics which represents complex operations are required. The extreme case of customized logics is ASICs. However, the use of customized logic can limit the flexibility of SDR solution. Another way for reducing power is to use more energy efficient circuits such as dynamic logics or adiabatic logics. However, for using these energy efficient logics, special design techniques are required.

In summary, the number of computations rapidly increases by much higher maximum data rate of future wireless protocols. The future SDR solution might use more complex customized logics and energy efficient circuits on top of parallel architecture in order to compensate the limitation on operation voltage scaling down.

CHAPTER VIII

CONCLUSION

This thesis proposes a low power, high performance, and programmable architecture that targets baseband processing workloads of software defined radio terminals. Due to a tight power budget and high demand on computation capability, using programmable hardware for the baseband processing of software defined radio terminals has been quite a challenging goal. The baseband processor of this thesis achieves these goals by maximally utilizing the characteristics of the baseband workloads. It is designed to fully exploit all available parallelism for higher throughput and to support algorithm diversity with the minimally flexible hardware for power savings.

For the proper understanding of workloads, in Chapters II, IV, and III, this thesis analyzes the characteristics of major computation kernels of most contemporary wireless networks, based on time division multiple access (GSM, GPRS, and EDGE), code division multiple access (IS-95, IS-2000, W-CDMA, and IEEE802.11b), and orthogonal frequency division multiple access (IEEE 802.11a/g, and WiMax) technologies. From this analysis, this thesis identifies the following factors:

- The baseband processing consists of five algorithm blocks: 1) channel encoding/decoding, 2) interleaving/deinterleaving, 3) modulation/demodulation, 4) channel estimation, and 5) pulse shaping. These algorithm blocks can run in

parallel without frequent interactions between them.

- Each algorithm block is the combination of two heterogeneous operations, 1) parallelizable operations and 2) sequential operations. These two operations are tightly coupled and require frequent communications between them.
- The parallelizable operations 1) dominate the entire baseband processing workload, 2) show a high degree of DLP, and 3) consist of a limited number of computation kernels.
- The major computation kernels comprising the parallelizable operations of the active mode are 1) the FIR filter, 2) pattern matching, 3) min/max finding, 4) Viterbi BMC/ACS, and 5) FFT.
- The major computation kernels of the idle mode are only 1) FIR filter and 2) pattern matching.
- The operations of these computation kernels can be mapped onto a conceptual pipeline, which consists of the following stages: 1) data load, 2) vector alignment, 3) vector computation, 4) a second vector alignment or vector reduction, and 5) data store.
- The sequential operations 1) occupy a minor portion in the entire baseband processing workloads, 2) show limited DLP, and 3) consist of numerous computation kernels with many variations.

Based on the above analysis results, in Chapters V and VI, this thesis proposes a coarse grain chip multiprocessor architecture, which consists of four PEs. The proposed architecture has the following high level features:

- A chip multiprocessor architecture is used to exploit the algorithm level parallelism.

- A coarse grain PE is used to minimize the power consuming interprocess communications.
- A low speed bus is used to support the communication between PEs.
- Hierarchical memory is used to minimize the power and delay cost of memory accesses.

On top of the high level architecture, this thesis proposes the detail architecture of the PE in Chapter VI. The PE, which consists of the parallel datapath, scalar datapath, control unit, instruction memory, and data memory, has the following features:

- Both the parallel and scalar datapaths were implemented within a single PE in order to process both parallelizable and sequential operations efficiently.
- Both the parallel and scalar datapaths share the same control logics and data memory in order to efficiently support the frequent interactions between the parallelizable and sequential operations with minimum power and delay cost.
- In order to exploit the abundant DLP, the parallel datapath has 16 CUs which concurrently perform arithmetic operations.
- The scalar datapath, control unit, instruction memory, and data memory form a 32bit single issue in-order processor to cover sequential operations with low power.

For further enhancement on system throughput and power efficiency, this thesis adapted the three novel schemes on the SIMD style parallel datapath: the macro instructions, macro pipelining, and staggered execution of CUs. These schemes were motivated from the limitations of the previous architecture, SODA. In the parallel datapath of the SODA, the substantial amount of system power was dissipated by

the vector register file due to unnecessary accesses. Furthermore, the SODA did not exploit the macro operation level parallelism.

- Macro instructions are adapted on the CU of the parallel datapath. A macro instruction is equivalent to several primitive instructions. Macro instruction enhanced the system throughput by executing several primitive instructions in parallel and system power efficiency by minimizing the power consuming register file access.
- The parallel datapath has a macro pipeline which consists of the switches, address generators, vector registers, CUs, and vector reduction logics. These hardware entities are the realization of the five macro operations identified in the workload characterization procedure, data load/store/alignment and vector reduction/computation. The macro pipelining increases system throughput by the parallel execution of these macro operations.
- In the workloads with long computation time, such as the Viterbi BMC/ACS and FFT, the operation timing of the CUs is staggered. The staggering allows to replace complex N×N crossbar switches with simple N×1 crossbar switches because only one CU accesses the data memory within one cycle. Consequently, it improve system throughput and power efficiency by minimizing the operation cost of data load/store/alignment operations.
- In the workloads with short computation time, such as the FIR filter, min/max finding, and pattern matching, the operation timing of CUs is synchronized. In order to provide input operands to all CUs with simple N×1 crossbar switches, the vector registers are used. The synchronous execution of CUs allows to support the idle mode workload with minimum power overhead.

In Chapter VII, the efficiency of the proposed architecture is evaluated through a Verilog model and commercial design tools such as Synopsys' Physical Compiler, Design Compiler, and PrimePower. The power estimation result shows that the proposed architecture consumed only 150 mW while providing W-CDMA 2Mbps packet data service.

In summary, this thesis proposed a programmable processor which targets baseband processing workload. Among all features proposed by this thesis, the followings are novel contributions:

- This thesis identified the characteristics of baseband processing workload in both system level and algorithm level from the view point of architecture design.
- This thesis improved the throughput and power efficiency of the proposed baseband processor by applying three novel schemes: the macro instruction, the macro pipelining, and the staggered execution of CUs.
- This thesis minimized the power consumption of baseband processor both in the idle and active modes.

Furthermore, the application area of the proposed architecture can be extended by the following future works.

- This thesis did not cover emerging wireless communication technologies such as MIMO and LDPC code. These algorithms show heterogeneous operation characteristics compared to the algorithms which were analyzed in this thesis. Optimizing the proposed architecture to these algorithms is additionally required to support future wireless protocols.
- The application area of the proposed architecture can be extended to source coding workloads, for instance video and voice CODECs. Because the compu-

tation patterns of CODECs also show high level of DLP, the proposed architecture could cover the source coding workloads without substantial architectural changes. Most wireless terminals also perform the source coding operations in addition to the baseband processing operations. Thus, to design a programmable processor, which supports both workloads, has significant meaning in practice.

APPENDICES

APPENDIX A

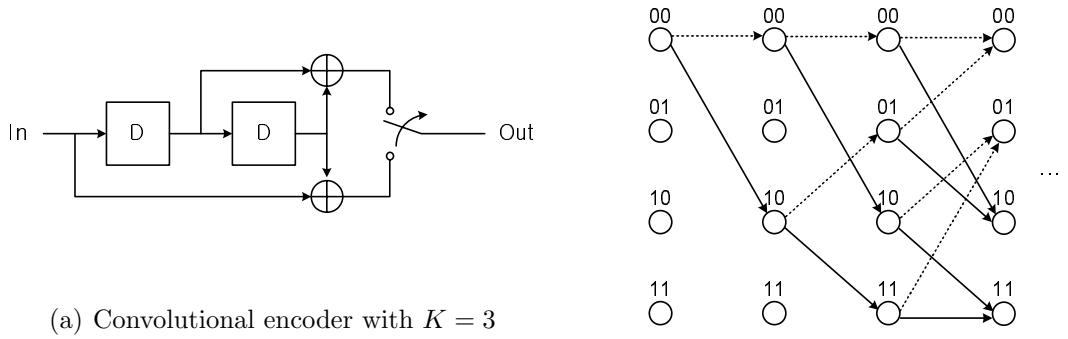
DETAIL ALGORITHM LEVEL WORKLOAD CHARACTERIZATION

A.1 Introduction

In this appendix, this thesis analyzes in detail the signal processing algorithms used in various wireless terminals. The analysis will include the computation pattern, data movement pattern, and inherent parallelism of five major blocks which are shown in Figure 3.1: channel encoder/decoder, block interleaver/deinterleaver, modulation/demodulator, pulse shaping filter, and channel estimator. As a representation of channel decoder, the Viterbi decoder and Turbo decoder are examined. Among modulation schemes, this appendix examines the computation pattern of TDMA, CDMA, and OFDMA. To understand the characteristics of channel estimators, this thesis analyzes the multipath searcher of CDMA based systems and the equalizer of TDMA based systems.

A.2 Channel Encoding and Decoding

Most of existing wireless communication networks use two kinds of channel coding schemes: Convolutional codes [27] and Turbo codes [47]. The encoders of Convolutional codes and Turbo codes are simple enough to be implemented with several



(a) Convolutional encoder with $K = 3$

(b) Trellis diagram

Figure A.1: An example of the (a) a convolutional encoder with $K = 3$ and (b) corresponding trellis diagram

flip-flops and exclusive OR gates. However, the decoders of these codes are very complex because their operation is to find maximum likely sequences from the received signals with noise. The amount of computations required for the channel encoding is negligible compared to that of other blocks, so this thesis only focuses on the characteristics of the channel decoders.

A.2.1 Viterbi Decoder

Viterbi decoder is a channel decoder whose operation is based on the famous Viterbi algorithm. In most wireless communication systems, it is a de facto standard to use Viterbi decoder for the decoding of the convolutional code.

Figure 1.1(a) shows an example of the convolutional encoder. Because the output of the convolutional code is determined by both one new input data and previous input data, which are stored in flip-flops, the constraint length of a convolutional encoder, K, is defined as the sum of the number of flip-flops and the number of input ports. Thus, the constraint length of the convolutional encoder shown in Figure 1.1(a) is 3.

The operation of the Viterbi decoder is based on a trellis diagram, which consists

of nodes and arrows as shown in Figure 1.1(b). The nodes in a trellis diagram represent the state of a convolutional encoder in the transmitter and the arrows represent possible state transitions of the convolutional encoder while time proceeds. The number of nodes in a column of a trellis diagram is 2^{K-1} because the convolutional encoder can have one of these many states at a given time. In an ideal situation, the number of columns of a trellis diagram must be equal to the length of the received frame. However, such an ideal Viterbi decoder requires huge memory space, more than several mega bytes, to store intermediate decoding results. Fortunately, there exists a previous research result showing that about $5K$ columns in a trellis diagram is enough for the Viterbi decoding, with tolerable decoding error probability [48]. The decoding method using partial trellis diagram is the sliding window method [26].

The operation of the Viterbi decoder can be divided into three steps: BMC, ACS, and TB. The BMC is a procedure that computes the cost of a state transition between two nodes in a trellis diagram. The ACS is to compute the cost of a path by accumulating the branch metrics of the path and then to select the minimum path among input transitions with different path costs. The TB is a procedure to recursively trace back the minimum cost path in order to find the input sequence which caused the selected minimum cost path.

A.2.1.1 BMC of Viterbi Decoder

From a perspective of computation, the BMC operation is equivalent to computing euclidian distance between two vectors as shown in the following equation:

$$\nu_n^{(i,j)} = |\overline{r_n} - \overline{x_n}^{(i,j)}|^2 = \sum_{k=0}^{l-1} |r_{n,k} - x_{n,k}^{(i,j)}|^2 \tag{A.1}$$

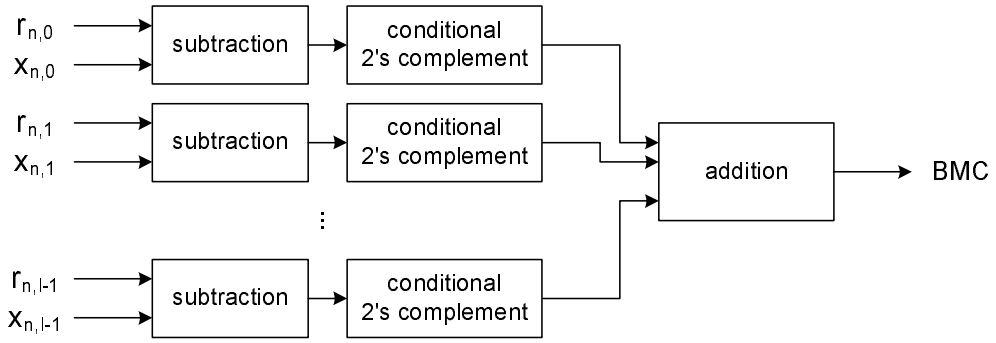


Figure A.2: A representation of BMC operation as a combination of elementary arithmetic operations

where $\nu_n^{(i,j)}$ is the branch metric of the edge between the source node i of the n-th column and the destination node j of the $(n+1)$-th column of a trellis diagram; $\overline{r_n}$ is the received signal vector with length l, which corresponds to the n-th column of a trellis diagram; and $\overline{x_n}^{(i,j)}$ is the error free reference output vector of a channel encoder with length l, which corresponds to the state transition from the node i of the n-th column to the node j of the $(n+1)$-th column. By assuming average white gaussian noise channel and binary phase shift keying (BPSK) or quadrature phase shift keying (QPSK) modulation scheme, Equation (A.1) can be further simplified in the following equation by eliminating square term[1]:

$$\nu_n^{(i,j)} \simeq \sum_{k=0}^{l-1} |r_{n,k} - x_{n,k}^{(i,j)}| \tag{A.2}$$

where $|\cdot|$ is an absolute value function.

Based on Equation(A.2), it is possible to represent the BMC operation into the combination of primitive arithmetic operations as shown in Figure A.2. The conditional complement operations shown in Figure A.2 is the real implementation of

[1]Detail explanation on this simplification procedure can be found in [48]

absolute value function based on the following relation:

$$abs(x) = \begin{cases} x, & x \geq 0 \\ -x, & x < 0 \end{cases} \tag{A.3}$$

According to [48], a 3bit number can be used for representing $r_{n,k}$ and $x_{n,k}^{(i,j)}$ in an ideal situation. In practical systems, a 4∼5bit number is used for representing these values.

The BMC operation on all nodes in a trellis diagram can be done in parallel because the input data of a BMC operation is not produced by other BMC operations. The error free reference encoder output vector, $\overline{x_n}^{(i,j)}$, can be pre-computed by using channel encoder specifications. The received signal vector, $\overline{r_n}$, must be prepared prior to the BMC operation.

The BMC operation exhibits a lot of DLP. The maximum number of BMC operation, which can be done in parallel with identical control, is equivalent to the number of nodes in a trellis diagram. For instance, the maximum number of parallelizable BMC operations is 10240 if we assume the Viterbi decoder of W-CDMA receiver, (K=9 and 5K columns in the trellis diagram).

A.2.1.2 ACS of Viterbi Decoder

Let us assume that there exist two input transitions at the node k of column $(n+1)$ from nodes i and j of column n as shown in Figure A.3. Then the output of the ACS operation on a node k of column $(n+1)$, μ_{n+1}^{k}, can be represented by the following equation:

$$\mu_{n+1}^{k} \;=\; min(\mu_n^i + \nu_n^{(i,k)},\;\; \mu_n^j + \nu_n^{(j,k)}) \tag{A.4}$$

where $min(\cdot)$ is the minimum value searching function and $\nu_n^{(i,k)}$ is the branch metric from node i to node k, which is computed in the BMC operation.

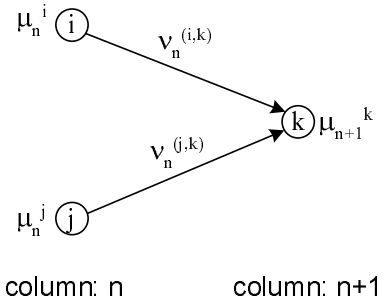


Figure A.3: ACS operation in a trellis diagram

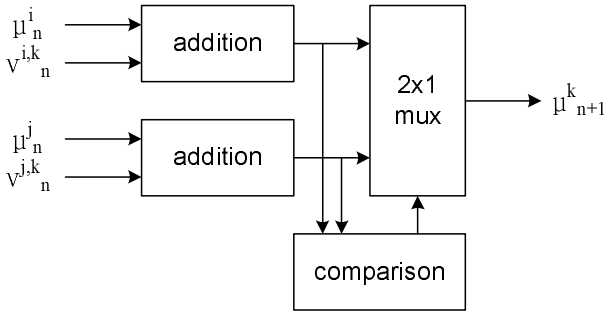
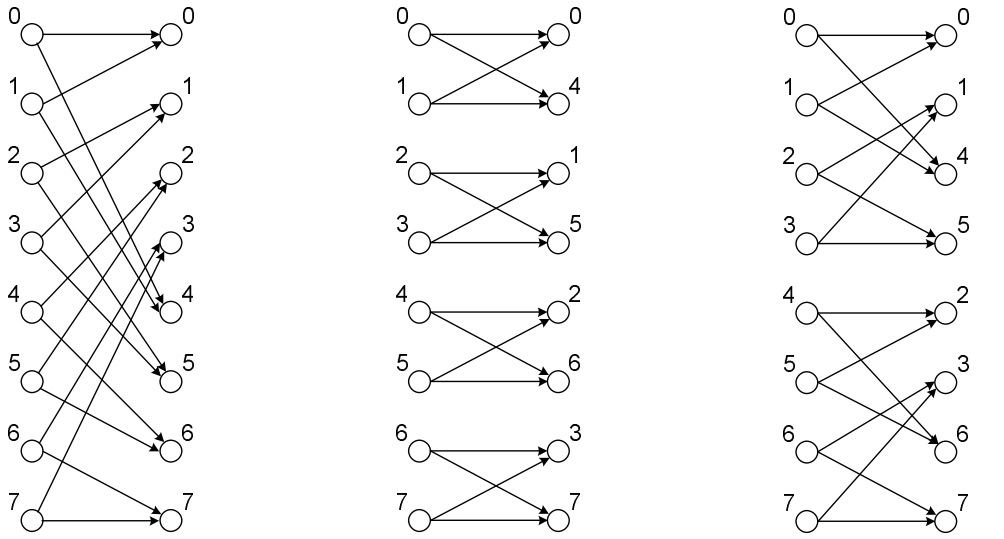


Figure A.4: An implementation of the ACS operation with primitive arithmetic operations

From the above equation, we can see that the required computations for an ACS operation is two additions and one minimum value searching function. The minimum value searching function can be converted into one comparison and one conditional selection according to the comparison result. Figure A.4 depicts an implementation example of the ACS operation with primitive arithmetic operations.

From Equation (A.4), we can see the data dependency between the ACS operations. μ_{n+1}^k depends on μ_n^i and μ_n^j which are the outputs of the ACS operations of the previous column. Thus, the maximum number of parallelizable ACS operations is 2^{K-1} where K is the constraints length of a convolutional code. The maximum



(a) The data movement pattern (b) 2×2 sub-grouping (c) 4×4 sub-grouping

Figure A.5: The data movement pattern of ACS operation when K=4 and sub-grouping of ACS operation into smaller groups

number of parallelizable ACS operations is 64 at the Viterbi decoder of the IEEE 802.11a system and 256 at that of the W-CDMA system.

The ACS operation exhibits a complex data movement pattern. It is because the results of the ACS and BMC operations of a column become the input operands of the ACS operations of the next column. A node i of a column is mapped onto two nodes of the next column.

$$i \longrightarrow \lfloor i/2 \rfloor \tag{A.5}$$

$$i \longrightarrow \lfloor i/2 \rfloor + 2^{K-1} \tag{A.6}$$

Based on the above relation, the operation results of column n are forwarded to the nodes of column $n+1$. The data movement pattern of the ACS operation is depicted in Figure A.5(a).

Another interesting factor in the data movement pattern of the ACS operation is the possibility of splitting the ACS operation on a column into smaller independent

ACS operation sub-groups. Figure 1.5(b) and 1.5(c) show an example that the ACS operation with K=4 is divided into four 2×2 sub-groups or two 2×2 sub-groups. It means that the ACS operation can be efficiently mapped on several independent PEs.

The operation results of the ACS need to be normalized in order to minimize hardware complexity. As shown in Equation (A.4), the ACS operation is a cumulative operation. In the trace back step, which will be discussed in the next subsection, the selection of a minimum cost path is based on relative difference between the path costs, not by their absolute value. Thus, the path cost normalization has no effect on the minimum path selection results. It allows less storage use, and consequently saves power. The normalization procedure consists of two steps. The first is to find a minimum value among all path costs of a column, and the second is to subtract the minimum value from all path costs of a column as shown in the following equations:

$$\mu_{min} \;=\; min(\mu^0, \mu^1, \cdots, \mu^{2^{K-1}-1}) \tag{A.7}$$

$$\mu^i_{norm} \;=\; \mu^i - \mu_{min}, \quad \text{where} \;\; 0 \leq i < 2^{K-1} \tag{A.8}$$

where μ^i_{norm} represents the normalized path cost of node i.

Parallelization of the minimum value searching function shown in Equation (A.7) can be done by using the compare-selector tree which is depicted in Figure A.6. The number of stages of a compare-selector tree is $\log_2 N - 1$, where N is number of nodes in a column. The data movement pattern appears in the minimum value searching function after parallelization is illustrated in Figure A.7. In order to maximize system throughput, it is required to load in parallel an input vector with width N from the memory which stores the ACS results. Due to the linearity of the minimum value searching function, no additional data alignment is required.

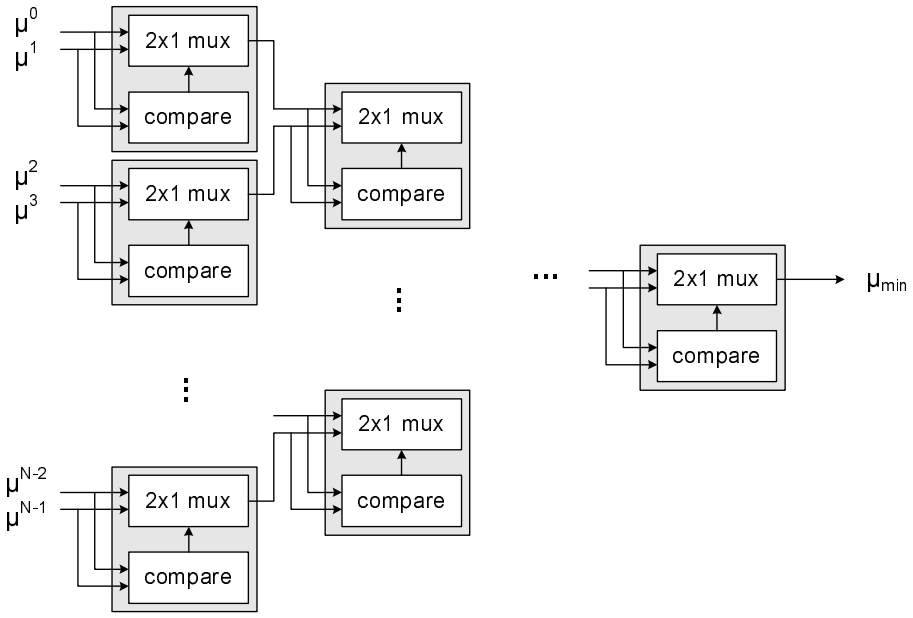


Figure A.6: A compare-selector tree for the minimum value searching function of the ACS normalization procedure

The subtraction operations shown in Equation (A.8) can also easily be parallelized because of its DLP nature. After parallelization, this operations also require to load N elements in parallel for maximum throughput.

A.2.1.3 TB of Viterbi Decoder

The first step of the TB is to find a node with minimum path cost η_{N_c} at the last column of a trellis diagram, as shown in the following equation:

$$\eta_{N_c} = node\{min(\mu^0_{N_c}, \mu^1_{N_c}, \cdots, \mu^{2^K-1}_{N_c})\} \tag{A.9}$$

where N_c is the number of columns in a trellis diagram and $node\{\cdot\}$ is a function that finds a node identifier from path cost information.

The next step is to trace back the selected minimum cost path by exploiting the

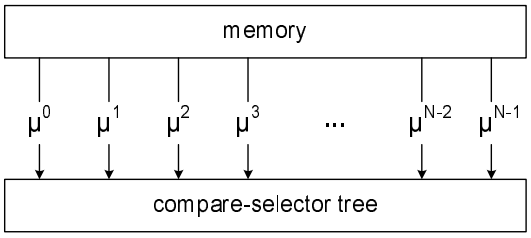


Figure A.7: The data movement pattern appearing in the minimum value searching function of the ACS normalization procedure, after parallelizing with the compare-selector tree

following relation:

$$\eta_{n-1} = source\_node\{\eta_n\} \tag{A.10}$$

Such trace back on a trellis diagram is recursively continued until reaching the first column.

Finally, the input data vector, $\mathcal{I} = (i_1, i_2, \ldots, i_{N_c})$, which triggered the state transitions between nodes $\eta_1, \eta_2, \ldots, \eta_{N_c}$, is the result of TB operation, as shown in the following equation, where $input\{\cdot\}$ is a function that finds an input data from state transition.

$$i_n = input(\nu^{(\eta_n, \eta_{n+1})}) \tag{A.11}$$

From the above equations, it is possible to see that the TB operation is inherently sequential, except for the minimum value searching operation in Equation (A.9). Thus, the operation time of TB can limit the maximum throughput of the Viterbi decoder.

There exists a method to avoid the TB procedure, called the register exchange method [49]. This method requires special hardware that accumulates the input data of survival paths. The accumulation of input data can be performed after the ACS operation on a node. Therefore, after finding a node with minimum cost

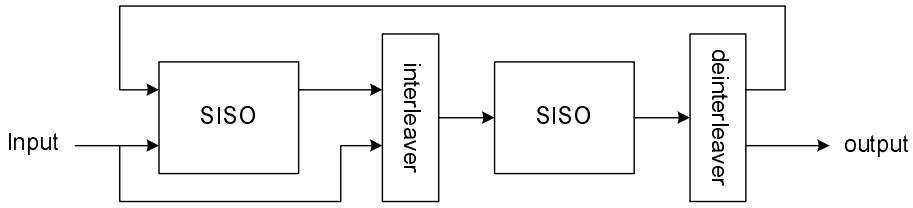


Figure A.8: The structure of Turbo decoder

path of the last column, as shown in Equation (A.9), it is possible to find input data vector $\mathcal{I}$ without the TB procedure, which are shown in Equation (A.10) and (A.11). However, the downside of the register transfer method is an additional power consumption required to accumulate input data at all nodes in a trellis diagram.

A.2.2 Turbo Decoder

The Turbo decoder decodes data encoded by Turbo codes. The Turbo decoder is identical to the Viterbi decoder due the fact that both operations are based on a trellis diagram. However, the Turbo decoder differs from the Viterbi decoder in three factors: producing soft output, iterative operation, and data interleaving between iterations.

The "soft output" means that output data is provided with reliability information, for example binary '1' with probability 0.7. In order to produce such soft output, the Turbo decoder performs more complex operations on a trellis diagram than the Viterbi decoder. Generally, the decoder, which produces soft output from soft input, is called soft input soft output (SISO) decoder.

As shown in Figure A.8, the Turbo decoder consists of two SISO decoders, interleaver, and deinterleaver. The output of the first SISO decoder is fed into the second SISO decoder and the output of the second SISO is also fed into the first one. The

Turbo decoder performs decoding operations iteratively. The number of iterations is related to the signal to noise ratio (SNR) of input data. The decoding of input data with low SNR requires more iterations. Due to mutual data dependency, iterations can not be performed in parallel.

The output of a SISO is interleaved before provided to the next SISO. This interleaving allows time diversity on Turbo codes. The time diversity is the main reason the Turbo codes show superior performance compared to the Convolutional codes. Although a full random interleaving guarantees the best performance, it requires excessively complex hardware. An interleaving scheme with some level of regularity, called block interleaving [26], is adapted in most wireless communication systems. It is because the computation pattern used in the interleaver of the Turbo decoder is identical to that of the frame level interleaver. The detailed computation pattern of the block interleaver will be discussed in the following section regarding the frame level interleaver.

There exist two popular methods on the implementation of the SISO decoder, which are based on two algorithms: soft output Viterbi algorithm (SOVA) [50] and max-log MAP algorithm [51][52]. The max-log MAP algorithm shows better performance than the SOVA, whereas the SOVA requires less computations. Thus, if decoding performance is the primary concern, then max-log MAP is a proper choice or if low power consumption is more important, then the SOVA can be used. Because there are no unique parallelizable computation patterns in the SOVA based Turbo decoder, compared to the Viterbi decoder, this thesis only discusses the operation of the max-log MAP based Turbo decoder.

A.2.2.1 Max-Log MAP based Turbo Decoder

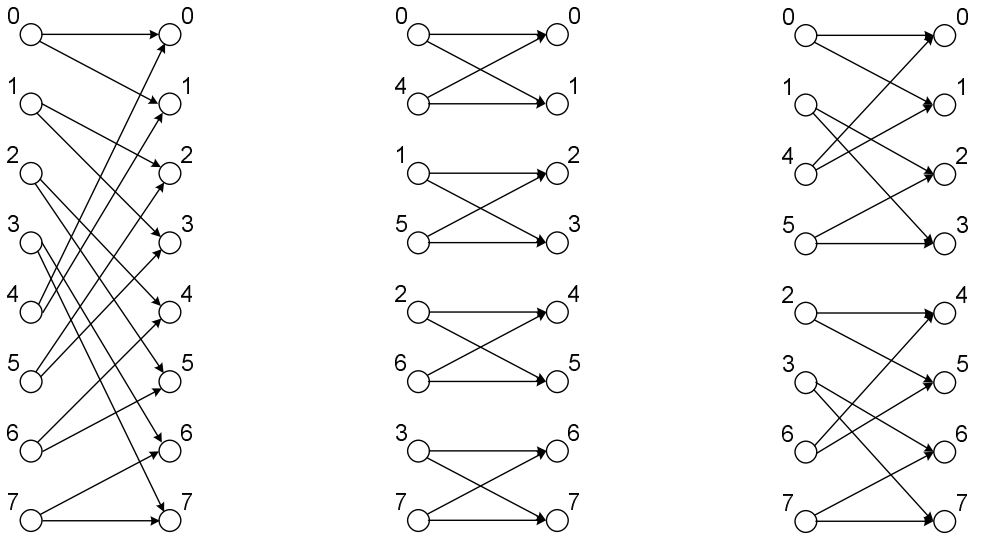
The operation of max-log MAP (from this point referred to as MAP for simplicity) decoder is a superset of the Viterbi decoder. Thus, MAP decoder also performs the BMC, ACS, and TB operations on a trellis diagram. However, additional operations are required to generate soft numbers, output data with reliability. In addition to the ACS operation done in the Viterbi decoder, the MAP decoder performs the ACS operation in a backward direction, from the last column to the first column of a trellis diagram. The computation pattern of the backward ACS operation is identical to that of the forward ACS except for the data movement pattern between columns. In the backward ACS operation, the data of the nodes i of a column need to be forwarded to the nodes of the next with the following relations:

$$i \longrightarrow (i \times 2) \bmod 2^{K-1} \tag{A.12}$$

$$i \longrightarrow ((i \times 2) \bmod 2^{K-1}) + 1 \tag{A.13}$$

where mod denotes modular function, which can easily be implemented by bit masking. The data movement pattern for the backward ACS is depicted in Figure 1.9(a). Similar to the forward ACS case, this operation also can be divided into smaller groups as shown in Figure 1.9(b) and 1.9(c).

Using both the forward and backward ACS results, the reliability of the decoded data is computed. The computation of reliability information is usually performed at the TB procedure because the max-log MAP algorithm converts power expensive multiplications and divisions into additions and subtractions; the arithmetic operations required for reliability computation are only addition, subtraction, and min/max finding.



(a) The data movement pattern (b) 2x2 sub-grouping (c) 4x4 sub-grouping

Figure A.9: The data movement pattern of backward ACS operation when K=4 and sub-grouping of ACS operation into smaller groups

A.3 Interleaver and Deinterleaver

The interleaver in a transmitter randomizes the sequence of source information, and then the deinterleaver in a receiver recovers the original sequence by reordering. These operations scatter errors within a short time interval over a longer time interval to reduce signal strength variation, and thus bit error rate under the same channel conditions.

Among the many existing interleaving schemes, block interleaving is most popular in practical wireless communication protocols. The detailed operation procedure of block interleaver is depicted in Figure A.10. The interleaving procedure consists of the following steps. Data for interleaving are written row by row on a memory space with m rows and n columns. Then, the stored data are read out column by column. The order of column reading is not sequential. The shuffling of the column reading order results in a pseudo randomness on the output data. Deinterleaving

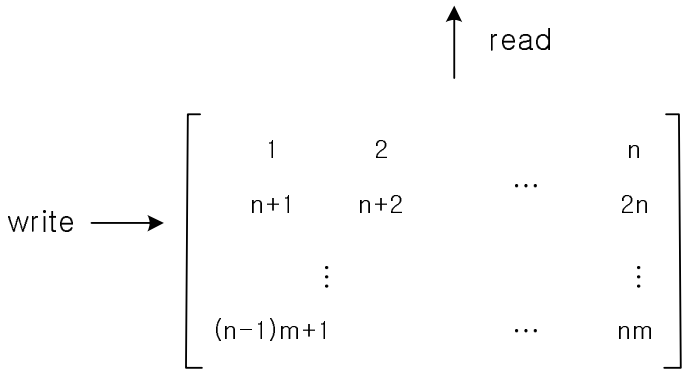


Figure A.10: The operation of the block interleaver

can be done by inverse operation. As shown in the previous figure, the operation of the interleaver and deinterleaver is difficult to parallelize without the assistance of special hardware. The interleaver and deinterleaver are ones of major sequential algorithms at the baseband processing workload.

A.4 Modulator, Demodulator, and Channel Estimator

Modulator maps source information onto signal waveforms so as to be transmitted over wireless links most efficiently. The demodulator extracts the transmitted information from the received signal waveforms, which are distorted while propagating in the wireless link. The channel estimator predicts the status of the wireless link in order to assist demodulator operation. The operations done in modulator, demodulator, and channel estimator are quite different, according to the types of communication technologies such as TDMA, CDMA, and OFDMA.

A.4.1 CDMA Based System

In the CDMA based systems such as IS-95, IS-2000, and W-CDMA, the modulation and demodulation operations are based on the multiplications of code se-

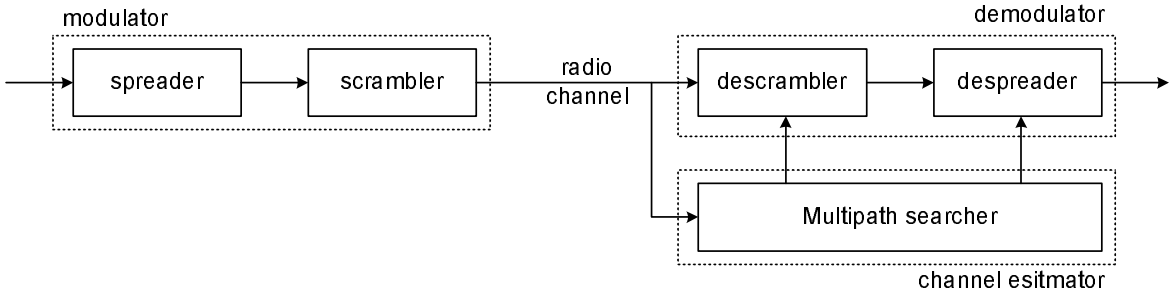


Figure A.11: The structure of modulator and demodulator of CDMA based system

quence to data. In the modulator, one of the code multiplication procedures is called "spreading" because it spreads out the energy of the source information over a wider frequency spectrum. Another code multiplication procedure is "scrambling" because a pseudo random sequence is multiplied on the input data so as to scramble the output. In the demodulator, the corresponding counter operations are "despreading" and "descrambling". The multipath searcher estimates the channel status and provides the estimation results to the demodulator. Figure A.11 shows the structure of the modulator, demodulator, and channel estimator of the CDMA based systems.

A.4.1.1 Modulator (Spreader and Scrambler)

Let us assume a spreading code vector $\mathbb{C}^{sp} = (c_0^{sp}, c_1^{sp}, \cdots, c_{L_{sp}-1}^{sp})$ where $c_i^{sp} \in \{1, -1\}$, and a binary input bit sequence $\mathcal{D}^T = (d_0, d_1, \cdots, d_{N-1})$ where $d_i \in \{1, -1\}$. Then, the output of the spreading operation, $\mathcal{D}^{sp} = (d_0^{sp}, \cdots\cdots, d_{NL_{sp}-1}^{sp})$, is represented by the Kronecker product between code sequence $\mathbb{C}^{sp}$ and input bit sequence

$\mathcal{D}$ as shown in the following equations:

$$\begin{aligned}
\mathcal{D}^{sp} &= \mathbb{C}^{sp} \otimes \mathcal{D} && \text{(A.14)}\\
&= (d_0\mathbb{C}^{sp}, d_1\mathbb{C}^{sp}, \cdots\cdots, d_{N-1}\mathbb{C}^{sp}) && \text{(A.15)}\\
&= d_0(c_0^{sp}, \cdots, c_{L_{sp}-1}^{sp}), \cdots\cdots, d_{N-1}(c_0^{sp}, \cdots, c_{L_{sp}-1}^{sp}) && \text{(A.16)}\\
&= d_0c_0^{sp}, d_0c_1^{sp}, \cdots\cdots, d_{N-1}c_{L_{sp}-1}^{sp} && \text{(A.17)}\\
&= d_0^{sp}, d_1^{sp}, \cdots\cdots, d_{NL_{sp}-1}^{sp} && \text{(A.18)}
\end{aligned}$$

From the view point of hardware, the spreading operation can be implemented by a conditional one's complement, con_comp$(\cdot,\cdot)$, which is represented by the following equation:

$$\text{con\_comp}(\mathbb{C}, d) = \begin{cases} \sim c_0, \sim c_1, \ldots, \sim c_{n-1} & \text{if } d = 1 \\ c_0, c_1, \ldots, c_{n-1} & \text{if } d = 0 \end{cases} \qquad \text{(A.19)}$$

where $\mathbb{C} = (c_0, c_1, \ldots, c_{n-1})$ and $\sim$ is the bitwise complement. Because of the absence of data dependency, it is possible to parallelize all con_comp() operations in the spreading procedure.

After the spreading procedure, the modulator also performs the scrambling operations. Before the scrambling procedure, the outputs of the spreaders are converted into complex numbers by mapping the output of one spreader to the real axis and the output of another spreader to the imaginary axis[2]. Therefore, the input of scrambler $\mathcal{D}^{in-sc} = (d_0^{in-sc}, d_1^{in-sc}, \cdots, d_{N_{sc}-1}^{in-sc})$ has the following relation with the outputs of two spreaders, $\mathcal{D}^{sp,0} = (d_0^{sp,0}, \cdots, d_{N_{sp}-1}^{sp,0})$ and $\mathcal{D}^{sp,1} = (d_0^{sp,1}, \cdots, d_{N_{sp}-1}^{sp,1})$:

$$d_i^{in-sc} = d_i^{sp,0} + j \cdot d_i^{sp,1} \qquad \text{(A.20)}$$

[2]There exist other ways to convert real numbers into complex numbers which map odd terms of input data on the real axis, and even terms of input data on the imaginary axis.

Then, the scrambling operation can be described by the following equation:

$$\mathcal{D}^{sc} = (d_0^{sc}, d_1^{sc}, \cdots, d_{N_{sc}-1}^{sc}) \quad (A.21)$$

$$d_i^{sc} = d_i^{in-sc} \cdot c_i^{sc} \quad (A.22)$$

where $\mathcal{D}^{sc}$ is the output of the scrambler; and $\mathbb{C}^{sc} = (c_0^{sc}, c_1^{sc}, \cdots, c_{L_{SC}-1}^{sc})$, $c_i^{sc} \in \{a + jb | a, b \in \{1, -1\}$, is the scrambling code sequence. Because both input data are complex numbers, one multiplication in the above equation can be converted into four real multiplications[3]. Additionally, because both real and imaginary terms of the operands of the multiplication are binary numbers, the four real multiplications can be simplified into bitwise exclusive ORs. However, the addition and subtraction required for representing complex multiplication need to be a 2bit adder which is not adequate for conventional 8bit or 16bit datapaths. Due to mutual independence of computations, all exclusive OR operations for scrambling can be parallelized.

A.4.1.2 Demodulator (Descrambler and Despreader)

In demodulator, a received signal is descrambled first. This descrambling procedure can be represented by the following equation:

$$\mathcal{R}^{dsc} = (r_0^{dsc}, r_1^{dsc}, \cdots, r_{N_{dsc}-1}^{dsc}) \quad (A.23)$$

$$r_i^{dsc} = r_i \cdot c_i^{*sc} \quad (A.24)$$

where $\mathcal{R}^{dsc}$ is the output of the descrambler; $\mathcal{R} = (r_0, r_1, \cdots, r_{N_{dsc}-1})$ is the received signal; and $\mathbb{C}^{*sc} = (c_0^{*sc}, c_1^{*sc}, \cdots, c_{L_{sc}-1}^{*sc})$ is the complex conjugate of the scrambling code. The real multiplications of the descrambler are converted into four conditional 2's complements, one addition, and one subtraction because $r_i \in$

[3]It is possible to implement the complex multiplications with three real multiplications [53]. However, due to the high register file access energy, it is difficult to expect an energy savings from this scheme.

$\{a + jb | a, b$ are n-bit integer$\}$ and $c_i^{*sc} \in \{a + jb | a, b \in \{1, -1\}\}$. Note that, in the transmitter scrambler, the multiplications were converted into bitwise exclusive OR because input data was a binary complex number. All conditional 2's complement, addition, and subtractions, which consist of the descrambler operation, can be executed in parallel because of the absence of data dependency between computations.

The despreading operation performed in the demodulator can be represented by the following equation where $\mathcal{R}^{dsp}$ is the output of the despreader; $\mathcal{R}^{dsc}$ is the input of the despreader, which is generated by the descrambler; and $\mathbb{C}^{sp} = (c_0^{sp}, \ldots, c_{L_{sp}-1}^{sp})^T$ is the spreading code used for the despreading operation.

$$\begin{aligned}
\mathcal{R}^{dsp} &= \mathcal{R}^{dsc} \cdot \mathbb{C}^{sp} && \text{(A.25)} \\
&= ((r_0^{dsc}, \cdots, r_{L_{sp}-1}^{dsc}), \cdots, (r_{(N_{dsp}-1)L_{sp}}^{dsc}, \cdots, r_{N_{dsp}L_{sp}-1}^{dsc})) \cdot \mathbb{C}^{sp} && \text{(A.26)} \\
&= (r_0^{dsc}, \cdots, r_{L_{sp}-1}^{dsc}) \cdot \mathbb{C}^{sp}, \cdots, (r_{(N_{dsp}-1)L_{sp}}^{dsc}, \cdots, r_{N_{dsp}L_{sp}-1}^{dsc}) \cdot \mathbb{C}^{sp} && \text{(A.27)} \\
&= (r_0^{dsp}, \cdots\cdots, r_{N_{dsp}-1}^{dsp}) && \text{(A.28)}
\end{aligned}$$

The despreading operation includes the inner product of two vectors with length L_{sp} as exemplified in the following equation:

$$r_i^{dsp} = \sum_{n=0}^{L_{sp}-1} r_{(i \cdot L_{sp}+n)}^{desc} \cdot c_n \qquad \text{(A.29)}$$

The multiplications shown in the above inner product can be converted into conditional 2's complement because $r_i \in \{$n-bit integer$\}$ and $c_i \in \{1, -1\}$. All inner products can be done in parallel.

A.4.1.3 Code Generation

There exists a predefined rule on the generation of the spreading code. The number of the spreading code used during a communication session, such as voice call, is limited. So, it is possible to compute the spreading codes at the starting

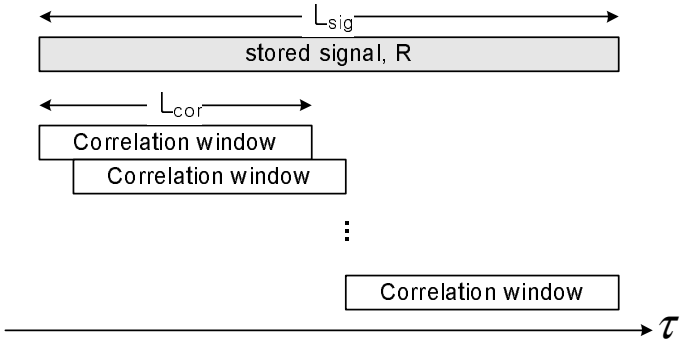


Figure A.12: The correlation computation procedure of the multipath searcher

time of communication session and reuse the pre-computed spreading code for the modulation and demodulation of data. Therefore, the amount of workload related to the generation of the spreading code is negligible.

However, because the scrambling code length is quite long, the pre-computation technique used for the generation of the spreading code is not available. In run time, the scrambling code sequence needs to be generated. The generator of the scrambling code is based on the linear feedback shift register. So, it can not be parallelized without ASIC style special hardware.

A.4.1.4 Channel Estimator (Multipath Searcher)

The channel estimator of the CDMA based system is called "multipath searcher" [29][54]. Under a multipath fading environment, an impulse signal from a transmitter will arrive at a receiver multiple times with random delay and attenuation. The operation of the multipath searcher is to detect the propagation delay and signal strength of the received multipath components. Its operation consists of four steps: correlation computing, average filtering, correlation peak detection, and selecting significant correlation peak points.

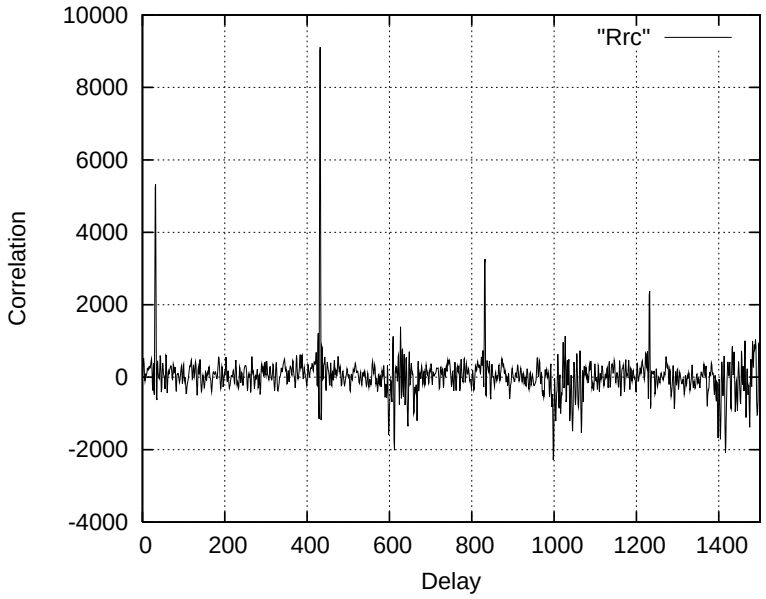


Figure A.13: An example of the correlation results which shows four peaks

At first, the multipath searcher stores received signals. The length of stored signal, L_{sig}, must be enough to cover the delay spread of the wireless link. Then, the multipath searcher scans the received signal, $\mathcal{R}$, by computing cross correlation with the complex conjugate of the scrambling code, $\mathbb{C}^{*sc}$, as shown in Figure A.12. The result of such correlation computation, $R_{rc}[\tau]$, is represented by the following equation:

$$R_{rc}[\tau] = \sum_{i=0}^{L_{cor}-1} \mathbb{C}^{*sc}[i] \cdot \mathcal{R}[i+\tau], \quad \text{where } 0 \leq \tau < (L_{sig} - L_{cor}) \tag{A.30}$$

where L_{cor} is the correlation length. $R_{rc}[\tau]$ shows peaks at the points where multipath components exist as shown in Figure A.13. The multiplications in Equation (A.30) can be simplified into the conditional complement like the multiplications of the descrambler with an identical reason. The computation of correlation on all delay points from $R_{rc}[0]$ to $R_{rc}[L_{sig} - L_{cor} - 1]$ can be done in parallel because there is no data dependency between them.

Second, the computed correlation results are processed by an averaging filter in order to remove high frequency noise terms. Let us denote the $R_{rc}[\tau]$ which is computed at time t as $R_{rc}[t,\tau]$. Then, the output of the averaging filter at time t, $\overline{R}_{rc}[t,\tau]$, is represented by the following equation where L_{avg} is the length of the average filter and δ is the interval between searcher operations:

$$\overline{R}_{rc}[t,\tau] = \sum_{n=0}^{L_{avg}-1} \alpha_n \cdot R_{rc}[t-n\delta,\tau] \tag{A.31}$$

Although both input operands of the multiplication in Equation (A.31) are real numbers, it is possible to convert the multiplications into cheaper shift operations by assuming $\alpha_n = 2^{-n}$. Based on such filter coefficients, the operation of the average filter can be simplified into the following equation:

$$\overline{R}_{rc}[t,\tau] = \frac{1}{2}\overline{R}_{rc}[t-\delta,\tau] + R_{rc}[t,\tau] \tag{A.32}$$

Third, the multipath searcher scans correlation results in order to find correlation peaks by computing $\Delta\overline{R}_{rc}[t,\tau] = \overline{R}_{rc}[t,\tau-1] - \overline{R}_{rc}[t,\tau]$. If $\Delta\overline{R}_{rc}[t,\tau_0] < 0$ and $\Delta\overline{R}_{rc}[t,\tau_0+1] > 0$, then the τ_0 is interpreted as a peak correlation point. Due to the absence of data dependency, the correlation peak searching operations on all delay points can be done in parallel.

Fourth, the multipath searcher selects several delay points where its correlation value is greater than a threshold value among the detected correlation peaks. In Figure A.13, two delay points are selected by the multipath searcher if the threshold value is assumed as 4000, and four delay points are detected if the threshold value is assumed as 2000. Basically this procedure is based on a sorting algorithm. In a practical multipath searcher, this threshold value adaptively varies according to operation conditions and channel status changes. In order to cope with wide algorithmic

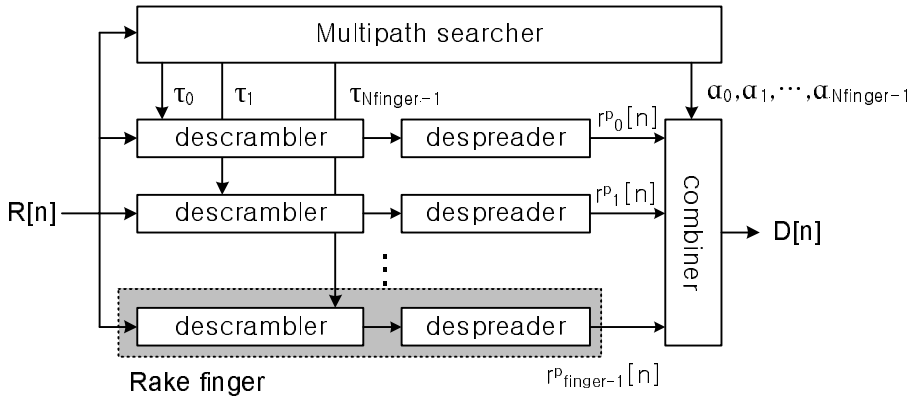


Figure A.14: The structure of the rake receiver

variations, this step of peeking dominant correlation points has been implemented on a programmable hardware even at ASIC based baseband processors.

Among the four steps, the correlation computing step shows a high degree of DLP. Although the other three steps show some level of DLP, it is possible to classify them as scalar workload because the amount of workload induced by these steps is not substantial.

A.4.1.5 Rake Receiver

In the receiver of the CDMA based system, the descrambler, despreader, multipath searcher, and combiner form a subsystem called "rake receiver", which is depicted in Figure A.14 [55][56]. Independent demodulators, which consist of the descrambler and despreader, are assigned to the delay points detected by the multipath searcher, $\tau_0, \tau_1, \ldots, \tau_{N_{finger}-1}$. A descrambler and despreader pair is assigned to τ_i and it produces a partial demodulation result, which is extracted from the multipath component existing at τ_i. The role of combiner is to aggregate the partial

demodulation results with proper compensation[4] as shown in the following equation:

$$D[n] = \sum_{i=0}^{N_{finger}-1} \alpha_i \cdot r_i^p[n], \quad \text{where} \quad \alpha_i = \frac{\overline{R}_{rc}[n,\tau_i]^2}{\sum_{k=0}^{N_{finger-1}} \overline{R}_{rc}[n,\tau_k]^2} \tag{A.33}$$

where $D[n]$ is the output of the combiner and $r_i^p[n]$ is the partial demodulation output of i-th demodulator at time n. The maximum number of parallel computations in the combiner operation is determined by the maximum number of rake fingers. Because both input operands of the multiplications of Equation (A.33) are arbitrary real numbers, there is no room for further simplification of multiplications to cheaper operations.

A.4.2 TDMA Based System

In a TDMA based system such as GSM, GPRS, and EDGE, the modulation and demodulation procedures are based on gaussian minimum shift keying (GMSK) and 8 phase shift keying (8-PSK). From the view point of computation, these modulation and demodulation procedures are table lookup operations which convert input bit sequence into complex numbers and vice versa. Because its workload is not substantial, we classify these workloads as sequential.

The major components of the TDMA receiver are the channel estimator and channel compensation filter, so called "equalizer" [29]. The goal of rhe equalizer is to minimize the signal distortion caused by the wireless channel. Although there exist many ways to implement an equalizer, most TDMA based systems use MLSE equalizer [30].

A.4.2.1 MLSE Equalizer

As depicted in Figure A.15, the MLSE equalizer consists of five blocks, mid-amble extractor, channel estimator, reference sequence buffer, reference sequence filter, and

[4]This operation is usually called maximal ratio combining [29].

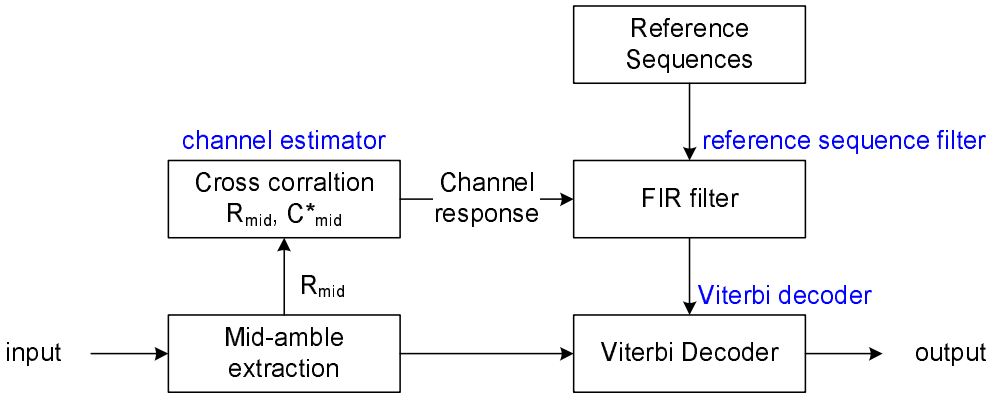


Figure A.15: The structure of the MLSE equalizer for TDMA systems

Viterbi decoder. Briefly, the channel estimator derives the characteristics of the wireless channel, and the Viterbi decoder exploits the channel information while estimating the maximum likely sequence from the input signal.

For channel estimation, the mid-amble sequence is utilized, which is transmitted at the middle of every frame. The impulse response of the wireless channel is extracted by matched filtering with the known mid-amble sequence. Because the mid-amble sequence is specially designed so that its autocorrelation is an impulse function, the matched filtering on the received mid-amble sequence results in the impulse response of the wireless channel in an ideal situation. The computation used at the matched filtering is identical to that of the multipath searcher. Thus, this thesis omits the detailed discussion on the matched filtering to avoid redundancy.

For estimating the maximum likely information sequence, the Viterbi decoder is used. The operation of the Viterbi decoder for equalization is a variation of the Viterbi decoder for the channel decoding. The Viterbi decoder for the MLSE equalization needs to consider the effect of channel distortion and it is done by filtering reference sequences with the channel impulse response function, which was

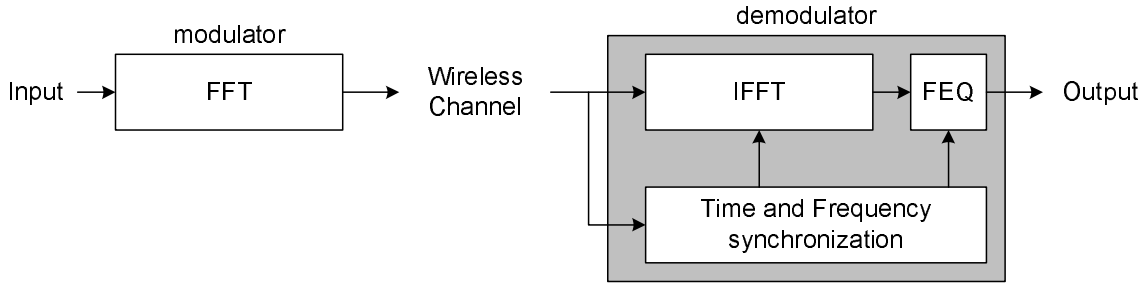
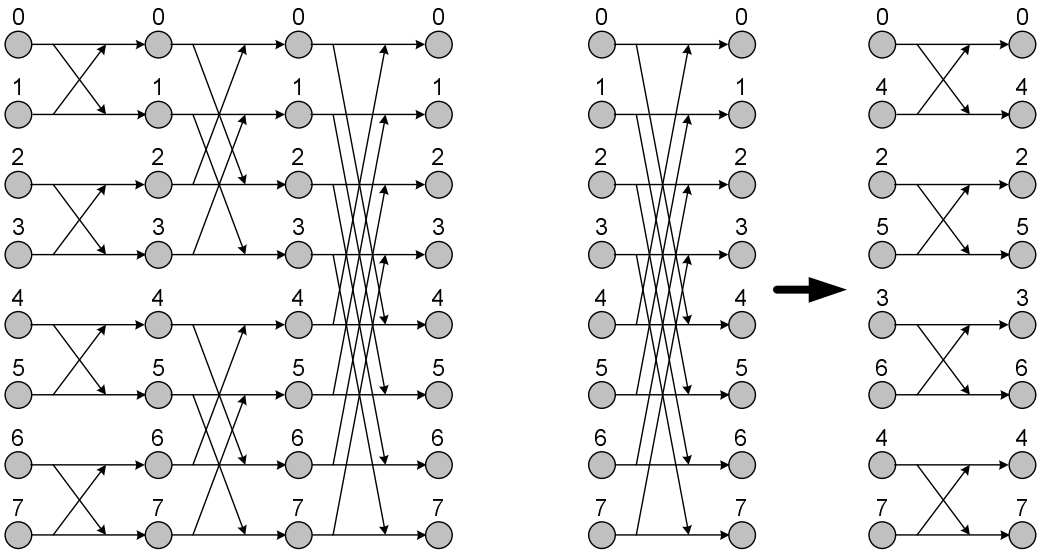


Figure A.16: The outline of modulator and demodulator of an OFDMA based system

computed by the channel estimator [57].

With the filtered reference sequences, the Viterbi decoder searches a maximum likely sequence from all filtered reference sequences by selecting a sequence with minimum path cost. The computation pattern required for the filtering of reference sequences is finite impulse response (FIR) filter. This computation pattern will be discussed in the following section regarding the pulse shaping filter. The Viterbi decoder for the MLSE equalization also performs the BMC, ACS, and TB operations and these operations were already discussed in the previous section. Thus, this thesis also skips detailed discussion on the computation pattern of the FIR filter and the Viterbi decoder for conciseness.

Although the MLSE equalizer is also one of the major kernels in TDMA based system, the computation patterns appearing in its operation are overlapped with other kernels. Thus, from the view point of computation, the MLSE equalizer does not increase the number of major computation kernels.



(a) The data movement pattern (b) sub-grouping

Figure A.17: The data movement pattern of 8 points FFT operation and sub-grouping of FFT operations into smaller groups

A.4.3 OFDMA Based System

In the OFDMA based system, FFT is used in modulation and demodulation procedures [58]. Because the OFDMA system is inherently robust to the multipath fading, the channel estimator is not additionally required in a receiver different from TDMA and CDMA based systems. However, the OFDMA based system is vulnerable to a synchronization error. The receiver of the OFDMA system uses many additional schemes, which compensate frequency, timing, and sampling errors. Figure A.16 shows the structure of the modulator and demodulator of the OFDMA based system.

A.4.3.1 FFT

The FFT is a way to compute discrete fourier transform (DFT) with minimum computation cost. Mathematically the FFT and inverse FFT (IFFT) are identical except for a scaling factor multiplied on the output of operation. Thus, this thesis

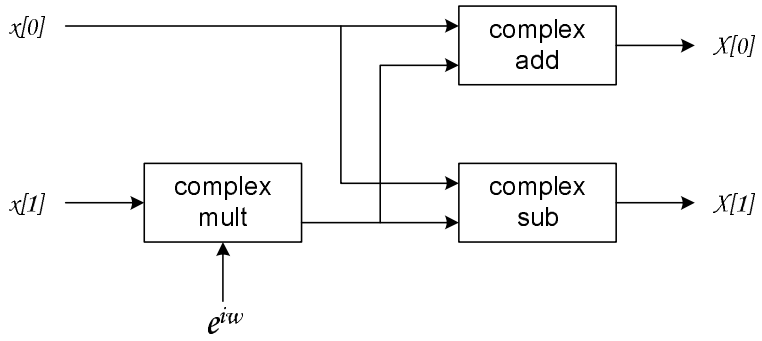


Figure A.18: Computation pattern of radix-2 FFT operation

treats the FFT and IFFT as identical operations.

If we assume an N points FFT, the FFT operation consists of $\log_2 N$ stages. Although it is possible to find regularity, the data movement pattern of each stage is different. Figure 1.17(a) shows the data movement pattern of 8 points FFT. As we discussed, this example shows $3(= \log_2 8)$ FFT stages. Similar to the ACS operation of the Viterbi decoder, the FFT operation can be divided into smaller subgroups. Figure 1.17(b) shows the subgrouping of the third stage operations.

The computation done in the radix-2 FFT operation is depicted in Figure A.18. It consists of one complex multiplication, one complex addition, and one complex subtraction. One complex number multiplication is equivalent to four real number multiplications, one real number addition, and one real number subtraction. Thus, total operations of the radix-2 FFT operation are four real number multiplications, two real number additions, and two real number subtractions. Additional characteristics of the multiplication of the FFT is that one of its two input operands is a number with unit magnitude.

In the OFDMA system, a subset of sub-carriers is assigned to one terminal. Thus, a terminal has no need to demodulate the signals of all sub-carriers. If the number of

sub-carriers assigned to a terminal, N_{sc}, is smaller than $\frac{1}{2}\log_2 N$, then the FFT is not an efficient modulation/demodulation scheme. The total number of complex number multiplications of the N points FFT is $\frac{1}{2}N\log_2 N$. If a direct multiplication is applied instead of the FFT, the number of complex number multiplications required for the demodulation of N_{sc} sub-carriers is $N \cdot N_{sc}$. Thus, if $N_{sc} < \frac{1}{2}N\log_2 N$, then the demodulation using direct multiplications performs the identical operation with less complex number multiplications compared to the FFT. Such situations frequently happen in a real system, especially when a low rate channel is used for voice service. The demodulation using direct multiplications can be represented by the following equation.

$$x[n] = \sum_{k=0}^{N-1} X[k] \cdot W_N^{-nk} \tag{A.34}$$

where $X[k]$ is a sample of the OFDM symbol and $x[n]$ is the signal demodulated from the n-th sub-carrier. This computation pattern is identical to that of the pulse shaping filter.

A.4.3.2 Synchronization

Various synchronization operations are performed in the terminal of the OFDMA system [59]. At first, the receiver needs to find the starting point of the OFDM symbol. In OFDMA systems, symbol timing is detected by computing cross correlation between the received sequence and the predefined preamble sequence whose auto correlation property is like the impulse function [60]. The computation done for the symbol timing detection is identical to that of the multi-path searcher. As this thesis already discussed, it is highly parallelizable.

Second is to compensate a sampling frequency error. This error is caused by the frequency difference between the oscillators for digital to analog (D/A) converter

of a transmitter and that for the analog to digital (A/D) converter of a receiver. The sampling frequency error is estimated by computing correlation between pilot sub-carriers as shown in the following equation:

$$R_{l,k} = P_{l,k} \cdot P^*_{l-1,k} \tag{A.35}$$

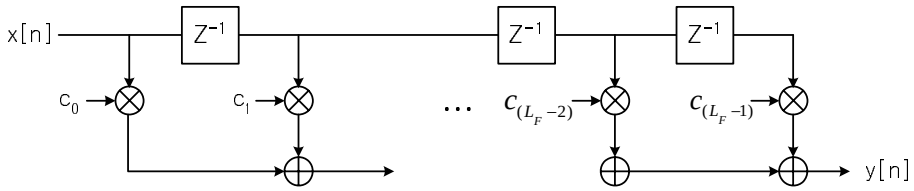
where $P_{l,k}$ is the k-th pilot signal of the l-th OFDM symbol. The sampling frequency error results in phase shift on the pilot signal and the result of the above correlation includes the information on the sampling frequency error [61]. It is possible to parallelize the correlation computations which are performed on different pilot sub-carriers. However, because the number of pilots is limited and the amount of computation required for this operation is not dominant, this thesis classifies this operation as the scalar workload.

Third, an OFDMA receiver must compensate the error on an oscillator for carrier frequency generation, which is used for placing a baseband signal on a transmission band and vice versa. Although there exist several available schemes, this thesis selects a scheme that exploits the nature of the preamble, which repeats the same information. The computation pattern used for the frequency error detection is identical to that of the symbol timing detection procedure.

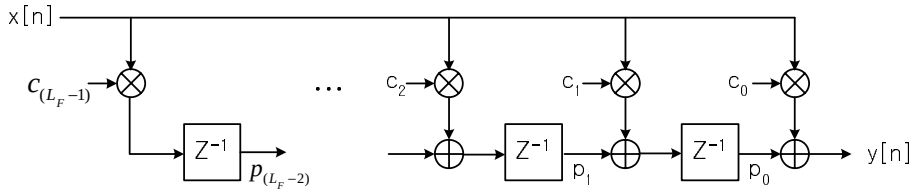
A.5 Pulse Shaping Filter

In most wireless communication systems, the pulse shaping filter is implemented in the form of the FIR filter [62]. There are two ways to implement the operation of the FIR filter: direct form and transpose form. Figure A.19 depicts the structure of the FIR filters based on these two implementation methods.

The operation of the direct form FIR filter is represented by the following equa-



(a) Direct form FIR filter



(b) Transpose form FIR filter

Figure A.19: Two implementation ways of the FIR filter

tion:

$$y[n] = \sum_{i=0}^{L_F-1} c_i \cdot x[n-i] \tag{A.36}$$

where $x[n]$ is the input sequence to be filtered; the $c_i \in \{\text{n bit integer}\}$ are the filter coefficients; and L_F is the number of filter taps. The operation of the transpose form FIR filter is represented by the following equations:

$$y[n] = p_0[n] + c_0 \cdot x[n] \tag{A.37}$$

$$p_i[n] = \begin{cases} p_{i+1}[n-1] + c_{i+1} \cdot x[n-1], & 0 \le i < L_F - 2 \\ c_{i+1} \cdot x[n-1], & i = L_F - 2 \end{cases} \tag{A.38}$$

where $p_i[n]$ is the partial result stored in i-th flip flop.

Although the functionality of these two implementation methods is identical, they have different impacts on hardware implementation. The first difference is the type of information stored in flip flops. The direct form FIR filter stores input data, $x[n]$, whereas the transpose form FIR filter stores partial results, $p_i[n]$. Because the partial results are the result of the multiplication between an input data and

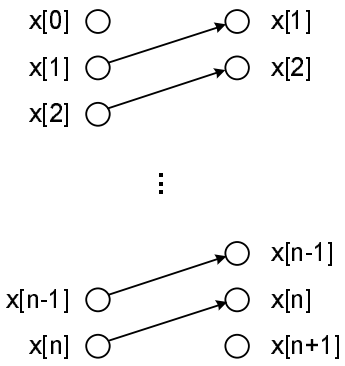


Figure A.20: Data movement pattern of the FIR filter. $x[0], \ldots, x[n]$ are the stored input data and $x[n+1]$ is a new input data

a filter coefficient, the partial result requires more storage elements than that of the direct form FIR filter. However, the direct form FIR filter needs to perform L_F multiplications and $L_F - 1$ additions to generate one output. However, the transpose form FIR filter only needs one multiplication and one addition for one output generation. Thus, the computation time allowed for the direct form FIR filter is much tighter than that of the transpose FIR filter. Therefore, the transpose form FIR filter is suitable for the time critical applications and the direct form FIR filter is appropriate for the applications with sufficient operation time.

The pulse shaping filter is used in both transmitter and receiver. In the transmitter, the data precision required to represent input data is lower than that of the FIR filter in the receiver. It is because the the receiver uses a higher precision number in order to reflect noise terms. In some modulation schemes such as BPSK or QPSK, the multiplications of the FIR filter can be simplified into cheaper conditional complement operations because $x[n] \in \{a + jb | a, b \in \{1, -1\}\}$. In the receiver, the multiplications of the FIR filter can not be simplified into cheaper operations because $x[n] \in \{a + jb | a, b$ are n-bit integer$\}$.

The maximum number of parallel computations available in the FIR filter is L_F due to the absence of data dependency between all computations. The data movement pattern appearing in the FIR filter operation is a simple data shift, as shown in Figure A.20. The FIR filter can be seen as a single input single output system because it requires only one new input data, $x[n]$, to generate one output, $y[n]$. Thus, the data memory with one read port and one write port is sufficient for the FIR filter operation.

A.6 Miscellaneous

In addition to five major blocks, there exist many miscellaneous operations in wireless terminals. In some special cases, the miscellaneous operations can be dominant workload. One example is frame detection operation that is to continuously monitor wireless channels to detect the start of new frame. This operation is used by wireless terminals running in ad hoc networks.

The computation pattern of frame detection operation is sliding window that can be represented by the following equation:

$$y[n] = \sum_{i=0}^{L-1} P_{i,n} \tag{A.39}$$

where $P_{i,n} = x[i+n] \cdot x[i+n-D]$, $x[n]$ is the input signal at time n, L is the correlation length, and D is the delay between input signals. It is equivalent to the auto-correlation of $x[n]$ with delay D and can be implemented with L multiplications and $L-1$ additions per output. However, the computation shown in Equation (A.39) can be further simplified due to the following relation between outputs:

$$y[n] = y[n-1] - P_{0,n-1} + P_{L-1,n} \tag{A.40}$$

Because we can reuse the previously computed $P_{0,n-1}$, the operation shown in Equa-

tion (A.40) additionally requires only one multiplication, addition, and subtraction for the generation of the next output. Therefore, the sliding window operation can be classified as a scalar workload.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Peter Savage. The perfect handheld: Dream on. *IEEE Spectrum*, 40(1):44–46, January 2003.

[2] Diederik Verkest. Machine chameleon. *IEEE Spectrum*, 40(12):41–46, 2003.

[3] Walter H. W. Tuttlebee, editor. *Software Defined Radio: Baseband Technologies for 3G Handsets and Basestations*. John Wiley and Sons, 2004.

[4] Upkar Varshney and Radihika Jain. Issues in emerging 4g wireless networks. *IEEE Computer*, June 2001.

[5] Hyunseok Lee, Yuan Lin, Yoav Harel, Mark Woh, Scott Malhke, Trevor Mudge, and Krisztian Flautner. Software Defined Radio - A High Performance Embedded Challenge. In *Intl. Conference on High Performance Embedded Architecture and Compiler*, Nov. 2005.

[6] John Glossner, Daniel Lancu, Jin Lu, Erdem Hokenek, and Mayan Moudgill. A software-defined communications baseband design. *IEEE Communication Magazine*, 41(1):120–128, January 2003.

[7] Andrew Duller, Gajinder Panesar, and Daniel Towner. Parallel processing – the picochip way! In *Communicating Process Architectures*, September 2003.

[8] Yuan Lin, Hyunseok Lee, Mark Woh, Yoav Harel, Scott Mahlke, Trevor Mudge, Chaitali Chakrabarti, and Krisztian Flautner. SODA: A low-power architecture for software radio. In *International Symposium on Computer Architecture*, pages 89–101, June 2006.

[9] Kees van Berkel, Frank Heinle, Patrick P. E. Meuwissen, Kees Moerman, and Matthias Weiss. Vector processing as an enabler for software-defined radio in handheld devices. *EURASIP Journal on Applied Signal Processing*, 2005(16):2613–2625, 2005.

[10] T. Halonen. *GSM, GPRS and EDGE performance: Evolution towards 3G/UMTS*. John Wiley and Sons, 2nd edition, 2003.

[11] Vieri Vanghi et al. *The cdma2000 system for mobile communications: 3G wireless evoluation*. Prentice Hall PTR, 2004.

[12] Harris Holma and Antti Toskala, editors. *WCDMA for UMTS: Radio Access for Third Generation Mobile Communications*. John Wiley and Sons, 2000.

[13] R. V. Nee and R. Prasad. *OFDM for wireless multimedia communications*. Artech House Publishers, 2000.

[14] J. Heiskala and J. Terry. *OFDM wireless LANs: A theoredical and practical guide*. Sams Publishing, 2002.

[15] WiMAX Forum. Mobile WiMAX-Part I: A Technical Overview and Performance Evaluation. Technical report, WiMax Forum, 2006.

[16] Liam Quinn, Pratik Mehta, and Alan Sicher. Wireless communictions technology landscape. White paper, Dell, February 2005.

[17] Jennifer Bray and Charles F. Sturman. *Bluetooth: Connet Without Cables*. Prentice Hall PTR, 2000.

[18] Ian Oppermann, Matti Hämäläinen, and Jari Iinatti, editors. *UWB: Theory and Applications*. John Wiley and Sons, 2004.

[19] Bob O'hara and Al Petrick. *The IEEE 802.11 Handbook: A Designer's Companion*. IEEE, 2005.

[20] Roger B. Marks. The IEEE 802.16 working group on broadband wireless. *IEEE Network*, 13(2):4–5, March-April 1999.

[21] Samuel C. Yang. *3G CDMA2000 Wireless System Engineering*. Artech House, 2004.

[22] Y. Neuvo. Cellular phones as embedded systems. In *IEEE International Solid-State Circuits Conference*, volume 1, pages 32–37, 2004.

[23] Seok-Jun Lee, Naresh R. Shanbhag, and Andrew C. Singer. A low-power VLSI architecture for turbo decoding. In *International Symposium on Low Power Electronics and Design*, pages 366–371, August 2003.

[24] Texas Instrument. Dsp selection guide. www.ti.com, 2007.

[25] Yuansheng Song, Gongyuan Liu, and Huiyang. The implementation of turbo decoder on DSP in W-CDMA system. In *International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1281–1283, September 2005.

[26] Branka Vucetic and Jinhong Yuan. *Turbo Codes: Principles and Applications*. Kluwer Academic Publishers, 2004.

[27] G. David Forney. The Viterbi Algorithm. *Proc. IEEE*, 61(3):268–278, March 1973.

[28] Stephen B. Wicker and Saejoon Kim. *Fundamentals of Codes, Graph, and Iterative Decoding*. Springer, 2002.

[29] Theodore S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall Ptr, 2001.

[30] John G. Proakis. Adaptive equalization for tdma digital mobile radio. *IEEE Transactions on Vehicular Technology*, 40(2):333–341, May 1991.

[31] Ken Gentile. The care and feeding of digital pulse-shaping filters. *RF Design Magazine*, April 2002.

[32] 3GPP TS 25.331. Radio resource cotnrol(rrc) protocol specification, Release 1999.

[33] TIA/EIA/IS-2000.3. Medium access control (mac) standard for cdma2000 spreade spectrum systems, March 1999.

[34] Hagen Woesner, Jean-Pierre Ebert, Morten Schlager, and Adam Wolisz. Power saving mechanisms in emerging standards for wireless lans: The mac level perspecitve. *IEEE Personal Communications*, 5(3):40–48, June 1998.

[35] Hyunseok Lee. W-CDMA benchmark. www.eecs.umich.edu/∼sdrg, December 2005.

[36] Nathan L. Binkert et al. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52–60, 2006.

[37] Samir Kallel and Cyril Leung. Efficient arq schemes with multiple copy decoding. *IEEE Transactions on Communications*, 40(3):642–650, March 1992.

[38] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantatative Approach*. Morgan Kaufmann, 2002.

[39] Ed Grochowski and Mural Annavaram. Energy per instruction trends in intel microprocessors. *Technology@Intel Magazine*, March 2006.

[40] James A. Kahle, Michael N. Day, H. Peter Hofstee, Charles R. Johns, Theodore R. Maeurer, and David Shippy. Introduction to the Cell multiprocessor. *IBM Journal of Research and Development*, 49(4/5), 2005.

[41] Jung Ho Ahn, William J. Dally, Brucek Khailany, Ujval J. Kapasi, and Abhishek Das. Evaluating the Imagine Stream Architecture. In *Proceedings of the 31th Annual International Symposium on Computer Architecture*, 2004.

[42] Ronny Krashinsky, Christopher Batten, Mark Hampton, Steven Gerding, Brian Pharris, Jared Casper, and Krste Asanović. The Vector-Thread Architecture. *IEEE Micro Special Issue: Top Picks from Computer Architecture Conferences*, November/December 2004.

[43] Simon Knowles. The soc future is soft. In *IEE Cambridge Branch Seminar*, Dec. 2005.

[44] Nam Sung Kim, Taeho Kgil, K. Bowman, V. De, and Trevor Mudge. Total power-optimal pipelining and parallel processing under process variations in nanometer technology. In *International Conference of Computer Aided Design*, November 2005.

[45] Hyunseok Lee, Trevor Mudge, and Chaitali Chakrabarti. Reducing idle mode power in software defined radio terminal. In *International Sympoium on Low Power Electronics and Design*, 2006.

[46] Nir Magen, Avinoam Kolodny, Uri Weiser, and Nachum Shamir. Interconnect-power dissipation in a microprocessor. In *International workshop on System level interconnect prediction*, pages 7–13, 2004.

[47] Claude Berrou and Alain Glavieux. Near Optimum Error Correcting Coding and Decoding: Turbo-Codes. *IEEE. Trans. on Communications*, 44(10):1261–1271, Oct. 1996.

[48] Kershab K. Parhi and Takao Nishitani, editors. *Digital Signal Processing for Multimedia Systems*. Marcel Dekker, Inc., New York, 1999.

[49] Charles M. Rader. Memory management in a viterbi decoder. *IEEE Transactions on Communications*, COM-29(9):1399–1401, SEPTEMBER 1981.

[50] J. Hagenauer and P. Hoeher. A Viterbi algorithm with soft-decision outputs and its applications. In *IEEE Global Telecommunications Conference and Exhibition*, November 1989.

[51] L. Bahl, L. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory*, IT-20(2):284–287, March 1974.

[52] Javan Erfanian and Subbarayan Pasupathy. Reduced complexity symbol detectors with parallel structures for isi channels. *IEEE Transactions on Communications*, 42(2/3/4):1671, February/March/April 1994.

[53] Adly T. Fam. Efficient complex matrix multiplication. *IEEE Transactions on Computers*, 37(7):877–879, July 1988.

[54] E. Grayver, J. F. Frigon, A. M. Eltawil, A. Tarighat, K. Shoarinejad, A. Abbasfar, D. Cabric, and B. Daneshrad. Design and vlsi implementation for a wcdma multipath searcher. *IEEE Transactions on Vehicular Technology*, 54(3):889–902, May 2005.

[55] R. Price and Jr. P. E. Green. A communication technique for multipath channels. *Proceeding of the IRE*, pages 555–570, March 1958.

[56] Ahsan Aziz. Channel estimation for a wcdma rake receiver. Application Note AN2253, Freescale Semiconduntor, November 2004.

[57] Bernard Sklar. How i learned to love the trellis. *IEEE Signal Processing Magazine*, 20(3):87–102, May 2003.

[58] S. B. Weinstein and Paul M. Ebert. Data transmission by frequency-division multiplexing using the discrete fourier transform. *IEEE Transactions on Communication Technology*, COM-19(5):628–634, October 1971.

[59] Hyoungsoo Lim and Dong Seung Kwon. Initial synchronization for wibro. In *Asia-Pacific Conference on Communications*, October 2005.

[60] T. M. Schmidl and D. C. Cox. Robust frequency and timing synchronization for ofdm. *IEEE Transactions on Communications*, 45(12):1613–1621, December 1997.

[61] M. Speth, S. Fechtel, G. Fock, and H. Meyr. Optimum receiver design for ofdm-based broadband transmission-part ii: A case study. *IEEE Transactions on Communications*, 49(4):571–578, April 2001.

[62] John G. Proakis and Dimitris S. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications*. Prentice Hall, 1996.