ABSTRACT

AN-TE DENG. Flexible ASIC Design using the Block Data Flow Paradigm (BDFP). (Under the direction of Dr. Winser E. Alexander and Dr. Clay S. Gloster.)

An Application Specific Integrated Circuit (ASIC) outperforms most processors; however, it is limited to one algorithm. Instruction Level Parallelism (ILP) processors, which include mixed type processors such as the Digital Signal Processor (DSP), the Very Long Instruction Word (VLIW) Processor, the Reduced Instruction Set Computer (RISC), and the Complex Instruction Set Computer (CISC), are popular due to their flexibility and programmability. Thus, they can be used for many different applications. However, their cost-performance can not meet the needs of many real world applications.

In this research, we mapped three different algorithms, the one-dimensional Finite Impulse Response (1D-FIR), the two-dimensional Finite Impulse Response (2D-FIR), and the two-dimensional Infinite Impulse Response (2D-IIR) filter into the same flexible hardware architecture using the Block Data Flow Paradigm (BDFP). The idea of a Flexible Application Specific Integrated Circuit (FASIC) is to design a mixed architecture using an ASIC for the fixed part of the system while using a Field Programmable Gate Array (FPGA) for the part of the system that requires a change of parameters for different algorithms. The Block Data Flow Parallel Architecture (BDPA) design, which has near super computer performance for media processing, is meant to form part of the FASIC library (Intellectual Property, IP) and is to be integrated with a general purpose host computer or mixed processor computer as an accelerator.

The FASIC, designed in this research, not only can accommodate different algorithms but also can flexibly change its configuration to obtain different cost-performance and frequency response outputs according to each system specification. We used the block data overlap-save algorithm for implementing the FIR filter system. This allows linear speedup performance if proper data input/output (I/O) and block size are given. We used the state space concept to implement the 2D-IIR filter system,

which also has the characteristic of linear speedup to a saturation point. In order to solve the data flow bottleneck problem which exists in most multiprocessor systems, we have designed a hierarchical data flow control architecture, an input data distributor, and a data source regulator. These designs are flexible and asynchronously controllable so that they can easily accommodate different algorithms. We used a one dimensional array architecture to reduce wafer area, pin connections, and power consumption (compared to a 2D array architecture). We used a four processor module array in the 2D-FIR filter system, which was designed on a multiprocessor system using a clock with two different frequencies. This system has throughput performance of 7.975 samples per processor clock cycle and the processor utilization is 78.53%.

FLEXIBLE ASIC DESIGN USING THE BLOCK DATA FLOW PARADIGM (BDFP)

by

An-Te Deng

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Department of Electrical and Computer Engineering

Raleigh

2002

APPROVED BY:

Dr. Winser E. Alexander

Dr. Paul D. Franzon

Dr. Clay S. Gloster

Dr. Zhilin Li

To my beloved wife,

Vicky,

and my two children,

Bobo, and Joshua.

BIOGRAPHY

AN-TE DENG was born to Mr. and Mrs. Shang Hun Deng on December 24, 1953, in Nan-To, Taiwan. He is the seventh child in the family. He attended the Naval Postgraduate School, Monterey CA, in 1989 and received his M.S. in System Engineering in Fall, 1991. He began his Ph.D. work at North Carolina State University in Fall, 1995.

ACKNOWLEDGEMENT

But they that wait upon the LORD shall renew their strength;
they shall mount up with wings as eagles;
they shall run, and not be weary;
and they shall walk, and not faint.

Isaiah 40:31

I thank God who gave His inspiration to me while I was in despair at the last stage of my research. I could not have completed this research without His companionship.

I would like to thank Dr. Winser E. Alexander for his patient mentoring, for his encouragement, and for his keen insights that have supported me through out my research and thesis writing. I would also like to thank Dr. Clay S. Gloster, Jr. for his generous advice and help. I also thank Dr. Paul D. Franzon and Dr. Zhilin Li for their assistance throughout my dissertation research.

I especially want to thank my beloved wife Vicky, who was always there when I needed her most.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The on-the-fly image processing system is embedded on an air-borne or a sea-borne vehicle. The image processing for this kind of system demands powerful computing performance in the range of 1 GFLOPs/s to 50 TFLOPs/s [19]. The solution to this real world demand is not only a high performance FASIC but also a small size and low power consumption system.

G. E. Moore has been recognized for the derivation of a law that has been valid for the past 20 years [4]. The density of integrated circuits increases at a rate of 50% every year, hence quadrupling every 3.5 years [5]. At the same time, the peak clock speed of the circuits doubles every three years.

Scientists in New Mexico have successfully transported an atom's nucleus from one area in a molecule to another without moving matter [27]. It is stated that this achievement, in which subatomic information was controlled and processed, could lead to extremely high-speed subatomic-based computers and communications systems. Also, physicists at MIT have announced that they have successfully controlled the atom in a laser-like stream [21]. It is called an atom laser. The laser has a much smaller wavelength than that of typical CO_2 lasers used in industry. The third, recently unveiled Extreme Ultraviolet Lithography technology [11], gives a hope to render a processor that is ten times faster in the near future. All these discoveries give strong indication that Moore's law will continue to be correct for the next decade.

The trend of integrating a complete system [5], which consists of different modules

with different algorithms on a single chip, will finally come true in the near future. The developing technologies mentioned above give certain confidence that there will be a great demand for system-on-chip design, which at present has a bottleneck due to the current scarcity of talented designers [25, 30].

The industry recently has been anxious to integrate many handy applications such as real-time voice/image communication, internet surfing, games, personal housekeeping management, and other functions together in one wireless hand-held machine. This information technology expansion and the demand for real-time image transmission have provided motivation for designers to develop faster and more efficient computers and coprocessors. As a result, the multi-processor, DSP chip, ASIC, paralleling instructions, Single Instruction Multiple Data (SIMD), and Multiple Instruction Multiple Data (MIMD) technologies have been implemented in attempts to solve these demanding computational problems.

1.1 Instruction Level Parallel Processor

In order to solve a computational-intensive problem in real-time, a designer would normally use one of two methodologies. The first one is to use available technology, which is more traditional, more general purpose, and more popular. We call it Instruction Level Parallelism (ILP) computation. One would select the mathematical representation of an application, program it with a popular high level computer language (i.e., C, C^{++}), which normally uses sequential operations, and identify the parallelism in the program instruction code for mapping the application to a multiprocessor/multifunctional unit system at a later point. This kind of design normally can be found in embedded system, Hardware/Software (HW/SW) co-design system, and multimedia processors.

The natural characteristic of ILP system design is that the operation deals with small granularity data, and the work load normally concentrates on the parallelization of instruction code in which the code itself has a sequential nature. The small granularity data processing increases the possibility of memory to processor and processor to processor inter-communication load. This communication load increases the pro-

cessing latency and therefore decreases the system performance. The design process and the control mechanism for the ILP system are complex whether using a special compiler or manually decoding the instruction to investigate the loop parallelism. The cost-performance of the ILP system is normally low and not efficient [14].

1.2 Algorithm-Hardware Architecture Mapping Processor

The second approach, which is implemented in this research, is to start the design by partitioning the input data geometrically and identifying the parallelism in the mathematical algorithm at the same time. The optimized and partitioned small algorithms are then mapped into a multiprocessor system. In December, 2000 [15] Kurt Keutzer et al. asserted that it is not a good approach to worry about hardware/software boundaries without considering higher levels of abstraction. They emphasize that capturing a design at higher levels of abstraction generates better designs in the end. They also believe that the most important point for functional specification is the underlying mathematical model. The concept that is emphasized by Kurt Keutzer has been carried out in the design process in this research. The mathematical algorithm partitioning, data allocation, data-processor path management, and concurrent data processing are considered simultaneously at the early design stage [14, 20]. This algorithm mapping, hardware architecture design methodology starts with understanding the application's arithmetic operations and its data path characteristics, and then continues with developing the proper architecture or micro-architecture specifically for its operation. This design methodology is also a future focus work mentioned by Kurt Keutzer's team [15]. The 2D-FIR filter system designed in this paper has a two-level hierarchical processor architecture. The hierarchical processor architecture, with each processor having an on-board FIFO memory and hierarchical data path control, is the new architectural design trend for the VLIW system. This concept has recently been proposed by Jacome and Veciana [14].

It is well known that once an ASIC design is finished, changing its architecture is

not possible. The traditional ASIC is designed specifically for only one application. The objective of this paper is to use the same piece of mixed ASIC/FPGA hardware to accommodate three different algorithms. The special feature in our research is that, although the designed system is a mixed ASIC/FPGA hardware system, the system can be configured into another algorithm architecture which runs a different functional operation by changing several parameters or a few modules in the processor core. This same hardware is not only scalable but also generates near supercomputer throughput performance. In order to reach this goal, we have divided our task into two parts. The first part is to solve the data flow bottleneck problem, which includes the data block formatting operation, data flow control, and asynchronous but concurrent data block processing. The second part is to design the processor core for three different algorithms with the effort of minimizing the variation of the processor architecture.

A one dimensional or two dimensional media signal normally is transmitted in a bit stream pattern. This bit stream data source, according to the algorithm that is in use, is first transformed into a proper blocked data format. All three algorithms presented in this research have their own blocked data format. We have designed a special Input Module (IM), its distribution module controls the input data format and it is reconfigurable to accommodate three different algorithms. A two way regulator is designed in the IM to control the data flow and to provide sufficient data for the processors. It monitors the data flow and handshaking with both the data source and the processor array.

The speed with which data flows into the processor array is very important. The key to the BDPA is to provide processors with enough data to keep them busy at all times. This is the basic requirement for a BDPA to generate a linear speedup (Amdahl's law) characteristic, high utilization, and near supercomputer performance. To meet this basic requirement of the BDPA, we used a dual clock rate system to provide processors with sufficient data and we obtained results that were almost up to forty times faster than we obtained with a mono-clock rate system as a result. We investigated the small scale linear speedup (single-digit speedup) characteristic of the BDPA when the system used a mono-clock rate and the data flow bottleneck problem existed. We also investigated the large scale linear speedup (double-digit speedup)

characteristic of the BDPA when the data flow bottleneck problem was solved by using the dual clock rate multiprocessor system.

In order to avoid clock skew, which normally happens in a systolic array system, and to avoid connection delay (as opposed to the traditional gate delay [30]), which easily happens in a deep sub-micron system on chip design, we developed a hierarchical asynchronous data flow control architecture. This architecture has the asynchronous data driven property, which satisfies the high performance computing, linear speedup BDFP basic requirements. It also accommodates a system with dual clock rates. If an asynchronous circuit is implemented in the system, the power consumption of the system will be lower than it would be with the normal synchronous one [10]. The asynchronous circuit system is more resistant to magnetic interference when the system clock frequency goes higher. The concurrent operation, throughput performance, and the efficiency for the asynchronous data driven system are also higher [10].

The dissertation presents the one dimensional finite impulse response (1D-FIR) filter, the two dimensional finite impulse response (2D-FIR) filter, and the two dimensional infinite impulse response (2D-IIR) filter designs as examples to explain the innovation of the block data flow paradigm. We have shown that this systematic algorithm mapping architecture design methodology can be implemented for different algorithms. The dissertation begins with the mathematical representation of the algorithm and considers different implementation schemes that can be used with a multiprocessor system. Secondly, it presents a way to partition the mathematical equations in order to simplify data access at the processor level. It discusses various architectures in the algorithm mapping process with a goal of obtaining a simple, flexible parallel architecture. Lastly, it describes a way to develop different algorithms that can fit in the same hardware without changing too many architectural parameters.

Digital filtering algorithms using a parallel block processing method were introduced in the early 1990s [29]. A ring type network, 1D processor array was investigated. We have modified this method with algorithm partitioning, block data input/output distribution, hierarchical data flow control, and the asynchronously task

processing mechanism to form the BDFP. This dissertation starts with the introduction, which emphasizes that the designed architecture of the BDPA matches with the trend used for many media/embedded processors. The second chapter gives a history of the previous related work. The characteristics of the BDFP, the basic block diagram of the BDPA and its function are introduced in chapter three. Chapter four explains a new hierarchical data flow control (HDFC) architecture. It describes the advantages of the HDFC and how a simple HDFC is implemented in the BDPA. Chapter five introduces a dual clock rate multiprocessor system concept and how it solves the data flow bottleneck problem in the BDPA. Chapter six describes the three algorithms that we present in this paper. Chapter seven addresses the data partitioning, the hardware architecture design and mapping. Chapter eight emphasizes the scalability and flexibility of our system architecture. It describes both the architectural difference between different algorithms and the variability within the architecture itself. Chapter nine introduces the design methodology we used and its validation. Chapter ten analyzes the performance of the three designed systems. The last chapter addresses our conclusions and future potential work.

1.3 Contribution

The purpose of this research is to map different algorithms on the same multiprocessor hardware. Several contributions were discovered/developed during the research.

- Three different algorithms have been successfully mapped onto the same hardware multiprocessor architecture.
- The flexibility and scalability of the hardware has been proven.
- A hierarchical data flow control and a hierarchical processor architecture (HPA) have been introduced.
- A dual clock rate BDPA multiprocessor system has been introduced.

- High throughput performance and the large scale linear speedup characteristic of the BDPA have been validated.
- A neat algorithm mapping and hardware architecture design research environment has been introduced to the High Performance Computing Laboratory at North Carolina State University.
- A remote real-time reconfigurable multi-function system concept has been introduced.
- The concept of extending the asynchronous characteristic of the HDFC to solve a normal inter-connect delay problem in a deep sub-micron System on Chip (SoC) has been introduced.

Chapter 2

Previous Work

Pierpaolo, et al., selected five versions of RISC workstations to demonstrate their image processing capability in 1996 [9]. Some source inefficiencies were found after a preliminary set of experiments.

- The intensive computing parts of Image Processing and Pattern Recognition (IPPR) programs are organized as processing loops of limited size. In such pieces of code, the loop control processing overhead takes up most of the total processing time.

- The iterative organizations of IPPR programs generate many data dependency problems in pipelined architectures.

- Image processing and pattern recognition programs have specific functional units (e.g., integer/floating point units). RISC machines have different data paths (e.g., from integer to floating point registers and vice versa). The use of the data types that naturally match the task characteristics in the source programs may turn out to be an un-optimized solution.

- Using arrays to keep large tables of pre-computed functions (e.g., trigonometric functions) in memory for fast on-line access may introduce unnecessary load/store instructions which slow the program execution. Most effective mechanisms to maintain their tables (e.g., arrays, variables, or constants) depend

both on the characteristics of the host architecture and on the compiler used.

- The choice of a specified order in input data structures (e.g., the images) allows defining program and data partitioning strategies. Their strategies depend on the size of the cache memory and are aimed at reducing the average latency of memory accesses.

The inefficiencies mentioned above exist for most traditional general purpose computers. As the computer-oriented modifications (i.e., partitioning the existing program into larger parallel loops, arranging the pre-computed variables in a way easier to be accessed) are made, problems associated with the compiler, data accessing and I/O contention still exist.

Hammerstrom and Lulich developed a one-dimensional SIMD (1D-SIMD) processor array with the precision of 8-bit fixed point in 1996 [12]. This 1D-SIMD chip was specially designed for image processing and pattern recognition. Bit-parallel arithmetic was used because programming it is easier and faster for high precision. On-chip or off-chip memory was reviewed and the system used on-chip memory. Nevertheless, the system had the following weaknesses:

- An 8-bit input/output port was used, making it inadequate for many applications.
- More on-chip taps were needed to provide more buffering for more processors.
- A more powerful, data-driven, barrel shifter was needed in the processor.
- A simpler, register-register RISC-like instruction set needed to be developed in order for the compiler to generate code easier.

Kneip et al. showed that VLIW processors are practical, parallel, programmable architectures for the image processing field in 1997 [17]. VLSI technology and a high-level language compiler, which automatically explore the instruction level parallelism, dominate the VLIW architecture. There are four constraints that limit the improvement of VLIW parallel performance. First, the typical image processing algorithms

have limited parallelism at the instruction level. Second, as the parallelism increases, the wiring connectivity required on the VLIW chip increases due to the use of a single global register file and the global crossbar network. This wiring becomes a performance-limiting factor. Third, as the internal instruction bandwidth increases, the I/O capacity increases as well. Small cache miss rates will lead to significant stall on executing the instructions. Fourth, the ability of compilers to schedule multiple threads onto a single VLIW stream is poor. In order to improve performance on an instruction level parallelism architecture, Kneip concludes that VLIW methodology will have to make the best use of the algorithm's parallelization potential. This leads to the need for implementing additional levels of parallelism, especially the use of data and task parallel processing. This conclusion is consistent with the concept of the BDFP. That is, the data should be partitioned in blocks and processing should be done with parallel pipeline processors.

The special Very Long Instruction Word (VLIW) Application Specific Instruction-Set Processor (ASISP) was proposed by Margarida F. Jacome et al. in 2000 [14]. The research emphasized adding as much functionality as possible to embedded software in order to meet the demand of designing complex systems within the short time-to-market windows. The traditional memory accessing problem with VLIW was addressed and a distributed memory solution was proposed. The VLIW ASISP architectural design direction was described in detail, which is very similar to the actual architecture of the FASIC. Data partitioning, memory allocation and assignment, scheduling of data operations, loop transformations, and other complex tasks were mentioned in the research. A library of hierarchically parameterized fundamental components was also proposed. The ongoing research for VLIW ASISP involves studying in more detail specific algorithms, data partitioning schemes, effective memory synthesis systems, and powerful optimizing compilers.

Frank Vahid and Tony Givargis at the University of California Riverside introduced their work on platform tuning for embedded system design in March, 2001 [31]. In their design, they could change the parameters on a tuning platform, which included cache capacity, cache line size, code/decode resolution, data block size (32,64 byte), LCD duty cycle, and memory size. They were not able to change the micro-

processor architecture, which was stated as one of their future interest.

Giobanni De Micheli et. al. published their research in "Hardware/Software Co-Design" in March, 1997 [20]. The paper discussed the defects that normally result during the design procedures for hardware/software co-designers. It is very common to design a system starting from applying task partitioning before thinking of scheduling/binding with the resource. This kind of design approach only introduces the loss of parametric accuracy, and the final result is not controllable. Only concurrently performing the task partitioning, scheduling, and binding to resources will generate an optimal resource utilization result. This paper gave us inspiration for designing the FASIC in that the data flow bottleneck, processing bottleneck problem, and limited resources should always be in mind when designing any module in the FASIC system.

The rapid prototyping of application specific signal processors (RASSP) program started in 1993 and ended in 1998 and was managed by the DARPA Electronics Technology Office. The program concentrated on improving the process for developing high-performance embedded digital signal processors. Four benchmarks were used during the five year RASSP program. They were Synthetic Array Radar (SAR) Image Processor (Benchmarks-1 & 2), Upgrade of an Embedded Sonar Processor (Benchmark-3), and Upgrade of an Embedded Image Enhancement Processor and ATR (Benchmark-4). The C and VHDL simulators of the SAR Image processor were available for companies and individual personal, but not for the public. The processors were basically integrated with Commercial Off The Shell (COST) components, such as DSP chips, micro-processors, and FPGAs. The tools used were upgraded from time to time. Graphical programming and auto-code tools were used for signal processing applications. Functional analysis of system requirements was carried out and resulted in a reduction of the number of operations in the image enhancement code by 2.8 times, and in the pattern matching code by 2.4 times. A 20 chip/second demonstration was achieved. A chip represents a 79 x 79 pixel sub-image.

The BDFP was developed in the late 80's. Dabbagh and Alexander derived four different realizations for 2D denominator-separable digital filters [6]. Comparing the mathematical performance of these four realized filters gives us a general idea that the state space realization is a good candidate for implementation on a multiprocessor

architecture. Digital filtering algorithms using a parallel block processing method were introduced in the early 1990s [29]. A ring type network, 1D processor array was investigated.

Xu and Alexander introduced the criteria for mapping algorithms into the BDFP [36]. The data partitioning, as well as algorithm-partitioning concept, was emphasized. A block data flow parallel architecture (BDPA) block diagram was formed. The concept of block data flow, the data transmission protocol, the linear array topology, the skew-operation, and the I/O data communication were discussed. A functional level simulation demonstrated that the expected performance of a BDPA system is very promising.

Since the BDFP has matured, several signal processing applications and matrix operation have used this concept to form a BDPA and each has obtained very promising performance. Wilburn [33] developed the algorithm partitioning for the Discrete Wavelet Transform (DWT) and applied the BDPA to real-time occluded object matching [35]. Yoon [37] extended this to a 2-D DWT architecture block diagram design. Howard [13] applied the BDFP to the Kalman filter and developed a modularized BDPA to solve the weather problem. Wilburn [34] also used the BDFP to develop a parallel solution for solving systems of linear equations. A systematic evolution of the BDFP, including an order graph method responsible for processing scheduling, was introduced by Alexander et. al. [2].

Among all the algorithm mappings mentioned above, a programmable simulator for analyzing the block data flow architecture [3] was developed in order to prove the performance of the architectural design. The Erg simulator used the C programming language to specify the target system architecture and the algorithm. Then, it was compiled with the host C compiler/linker. The executable version of the simulator was created from the state code, the architecture description, and a simulation kernel. The Erg compiler included timing estimation for each operator based on benchmark execution or on parameters obtained from a processor data book. Architectural features such as a cache, branch target buffers, limited register set size, etc. were not included in the Erg simulator.

Many algorithms have been mapped into a hardware architectures using the BDFP. However, the structured C simulator (Erg) was not sufficient to verify the actual timing or critical path for the hardware system designs. We have extended this study by capturing the design of the FASIC using the Verilog HDL and by performing more detailed simulations to verify the actual timing and to identify critical paths that may limit throughput performance.

Chapter 3

Block Data Flow Paradigm

The Block Data Flow Paradigm (BDFP) involves partitioning data into large granularity blocks, scheduling and processing the data within different processors in which partitioned algorithms are embedded. The Block Data Parallel Architecture (BDPA) is an example of the implementation of the BDFP [2]. It reduces the communication overhead problem between memory and processors. It uses the asynchronous data flow characteristic so that the control overhead is reduced. The data communication time is overlapped with the computing time. Therefore, it gives high performance. The BDPA adopts the 1D linear array pipeline characteristic so that the system utilization is very high and easy to scale up. The number of input/output pins per-processor will not increase as the number of processors increases. The power dissipation is lower than for the 2D array system, yet the performance is similar [12]. The BDPA has the following special characteristics:

- Different but similar DSP application algorithms are studied and broken down into small parallel processes.
- The small algorithmic parallel processes are translated into small programs and installed into different processors.
- Massive data is partitioned geometrically into relatively small blocks that fit the size of each identical processor. A different block of data is assigned to each processor and brought into the designated processor.

- The processors are arranged in a one directional linear array pipeline.
- The hierarchical data flow control concept is implemented to enable individual processors to operate concurrently and asynchronously. The processing of a block begins as soon as it is available. No global control or complicated scheduling is needed to indicate the explicit start and stop times for block processing.
- Each processor computes the output derived from its assigned input block and transmits this output to the output device as soon as it is ready.
- Partial or intermediate results, which must be exchanged between processors, pass through a point-to-point simple (single stage) interconnection network.

3.1 Block Data Parallel Architecture (BDPA)

The theme of the BDPA is to process the data in large granularity blocks. We developed the BDPA to implement the BDFP for a variety of one-dimensional and two-dimensional DSP applications. The architecture is composed of three main parts, the Input Module (IM), the Processor Module Array (PMA), and the Output Module (OM) in order to manage the block flow smoothly through the processors.

In figure 3.1, the IM receives the data string coming from the input device. The input device could be an audio/video recorder, a sound/graphic file installed on a disk, or a host system data file. The IM acts as an I/O buffer which splits and forms the incoming data string into data block format. It has a distributor at the front end so that the data blocks can be distributed into the corresponding FIFO buffer. The FIFO outputs transfer data blocks into each of the different processor modules in a direct individual channel without any interference. A data flow regulator is responsible for handshaking at both sides of the IM, which is the input device and the PMA.

The PMA contains adequate processors for real-time data processing computational power. The FIFO in the IM sequentially sends the designated blocks of data into the corresponding PMA processor. Each processor has a time-divided separate

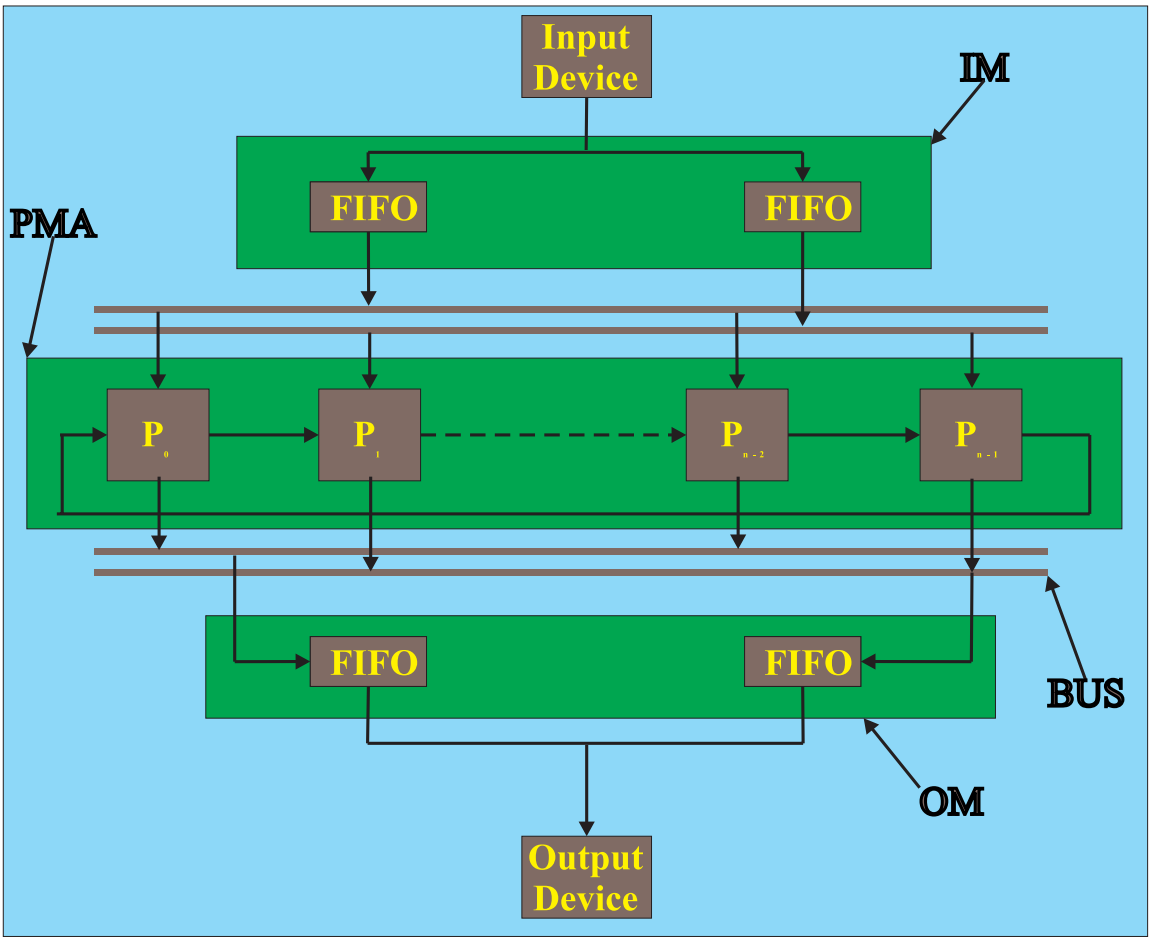

Figure 3.1: Block Data Parallel Architecture

input channel and an output channel. The processors are divided into two processor groups, namely, an odd number processor group and an even number processor group. Each processor group is directly connected to one of the input FIFO buffers and one of the output FIFO buffers. The purpose of dividing the processors into two groups is to make sure that the two IM FIFO buffers alternatively write and read at the same time. The individual processor will process its own mathematical operations (small programs) on the data and update the values inside itself until the whole block of data is finished. The partitioned algorithm and the partitioned data determine the number of iterations within a block. The intermediate values from each processor are transferred to the FIFO between the processors as needed according to the algorithm being implemented. The transferring of the intermediate values is only in one direction. The calculated output value of each processor is output sequentially to the FIFO in the OM.

The output blocks of data coming from different groups of processors are fed into two-separated FIFO buffers and are multiplexed into a synchronized output data stream. The OM may contain a post-processing sub-module, which handles the data management and communication service. This post-process sub-module may also contain different functional modules that flexibly adapt to different applications (i.e., wavelet coding or voice identification).

Chapter 4

Hierarchical Data Flow Control Concept

Traditional processor control normally uses a centralized control technique which is designed with complex scheduling schemes, such as interrupt, queue, semaphores, round robin. When the system grows larger, it is very difficult to synchronize the multi-tasking.

The control concept in our system is a Hierarchical Data Flow Control (HDFC). We divide the system control into different hierarchical levels. The control in every hierarchy is self-dependent. The hand shaking between each hierarchy depends on the data flow, meaning the operation of different hierarchy control is decided by the availability of the data. Once the data is available, the controller in that hierarchy will start its processing automatically. The data flow controls the whole system operation. Therefore, a globally asynchronous control and locally synchronous control is obtained. The HDFC architecture is totally modularized and gives the designer a certain degree of freedom to dynamically integrate different modules to accommodate different algorithm mappings and architecture designs. It also facilitates to design a small local controller, and integrate the small controller later on at another higher level module. If the logic circuit is implemented with asynchronous logic, then the system power consumption can be reduced [1, 10]. This is a very important factor in wireless personal digital assistant (PDA) applications in the future. In addition,

the asynchronous design renders more concurrent operations and higher efficiency (average-case performance as opposed to worst-case performance). When the system design gets more and more complicated and different function modules are integrated for a system on a chip using deep sub quarter micron silicon, the timing delay problem between different connected modules will get more serious. An asynchronous HDFC, modularized design, will eliminate a lot of these potential problems [30].

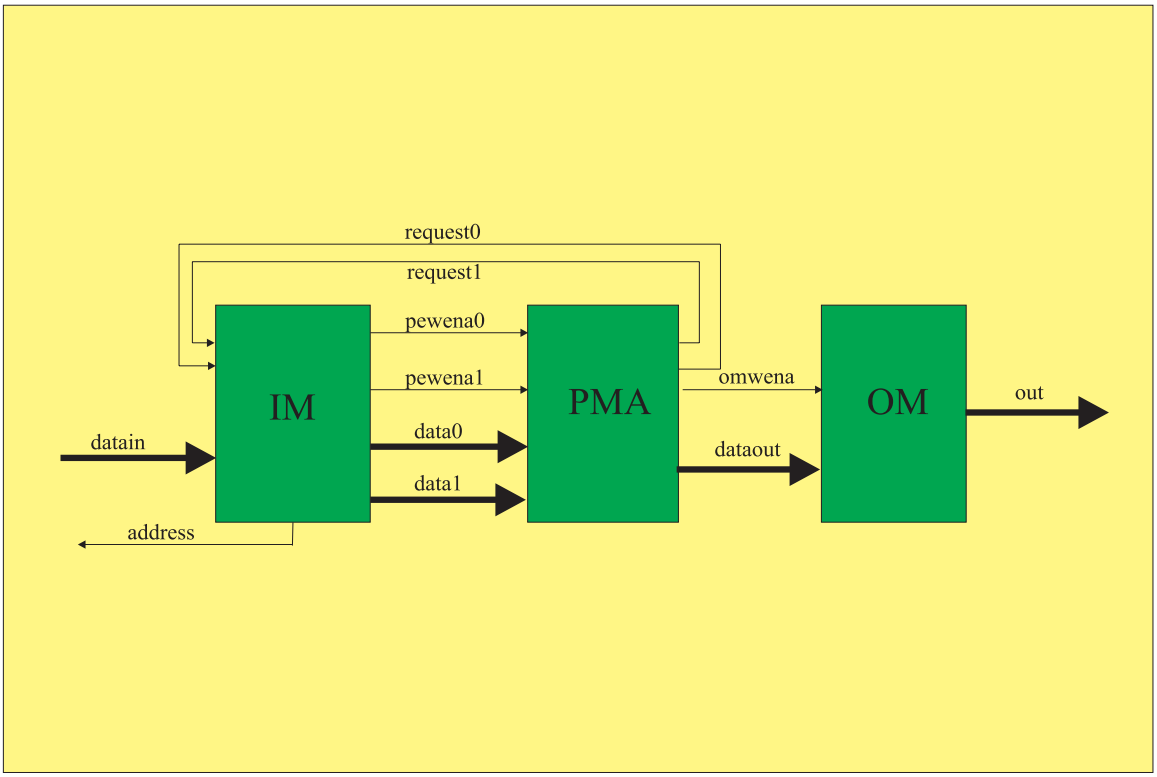

Figure 4.1: System Top Level Block Diagram

Figure 4.1 illustrates the top level block diagram of our system. There is no synchronization or scheduling for multiple tasks. The data stream is input from the memory and distributed by the distributor into the two IM FIFOs. The actual input timing is determined by the two FIFO write enable signals, which are generated by the IM distributor. The speed of the block data flow is determined by the address line from the IM which is controlled by the IM regulator. The IM regulator monitors the IM FIFO and controls the block data flow from the memory to the PMA.

The IM FIFO guarantees adequate data block are provided for the PMA. Only when both of the IM FIFOs are full will the IM regulator, which monitors the IM FIFOs status, fire a freeze signal to stop the data coming from the data source. Therefore, the regulator control signal is determined by the data flow situation.

In figure 4.1, the signals exchanged between the IM and the PMA are the request signals, the data buses, and the FIFO write enable signal lines. The only active (as opposed to passive) signal generated from the PMA to the IM is the data request signal. This signal will stay alive until one block of data is fed into the requesting processor. There will be no request signal if all the processors are busy. Other than the request signal, the PMA has no control over the data flow.

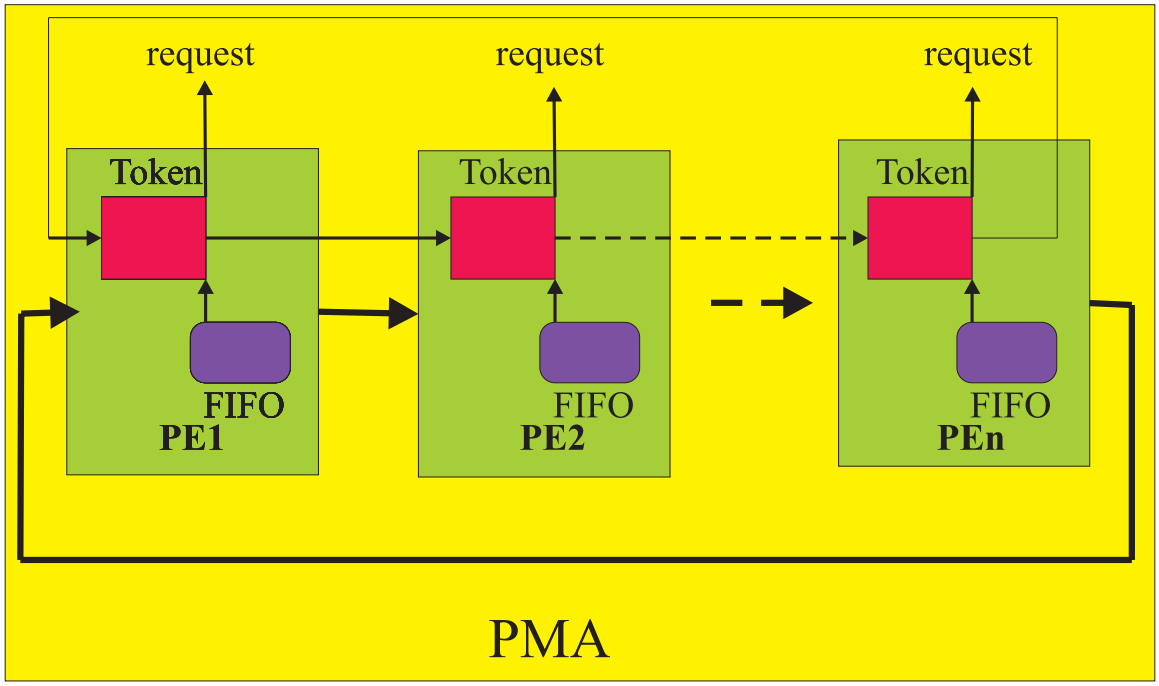

Figure 4.2: Processor Module Array Block Diagram

In the PMA as shown in figure 4.2, every processor has one token control circuit. The ring type token passing controller will decide which processor has the right to access the data bus coming from the IM. The token controller also monitors the individual FIFO status. Once the processor receives its own data block, it will shut itself off to the outside environment, and process its data block automatically un-

til the block of data is output to the OM. The processor architecture itself can be hierarchical. Whether it is one level processing or two level hierarchical processing depends on the algorithm. The algorithms we use have architectures with no more than two hierarchical levels. Therefore, the hierarchical processing controller is easily implemented in our system with a variable counter combined with a finite state machine controller. The controller follows the data processing cycles, controls the data processing flow, data path, and feeds the data back to the processor FIFO.

Since the data block assignment (the path of the data block) is controlled by the token passing controller, there is no need for complex task scheduling, synchronizing controllers, or complex data path assigning. Thus, the PMA is a self-sufficient control, pipeline parallel processing, one dimensional linear array, and is a fully data block driven module.

The control between the PMA and the OM is flexible, and depends upon the output application. The output of the PMA can be input into another algorithm processor or input into a memory buffer. Normally, it is a memory device, which can be controlled by the PMA output enable signal.

Chapter 5

Dual Clock Rate Multi-processor System

Linear speedup and high throughput performance are the goals and characteristics of a BDPA. In order to reach these goals, it is necessary to provide the processors with sufficient data and keep them busy at all times. However, most of the time, a multiprocessor system bottleneck occurs either due to the speed of the data bus or instruction scheduling (i.e., branch prediction). We categorize this kind of bottleneck as the data flow bottleneck. This data flow bottleneck will saturate the multiprocessor operation and prevent the system from obtaining any more speedup. In other words, no matter how many processors are added to the multiprocessor system, there is no increase in throughput.

Another category of bottleneck is the processing bottleneck. Assuming that the data flow bottleneck is solved and the processors are provided with as much data as they need, then, the processors are busy all the time, and the throughput can not be improved by providing more data. This is called the processing bottleneck. Normally, it is easier to solve the processing bottleneck problem in most of the multiprocessor system by adding more processors. Multiprocessor systems with inter-communication between processors is an exception to speeding up the system performance by adding more processors. It is more difficult to solve the data flow bottleneck problem than to solve the processing bottleneck problem. The use of two FIFOs in the IM and the

OM was done to reduce the data flow bottleneck problem. The use of a dual rate clock multiprocessor can further reduce this problem.

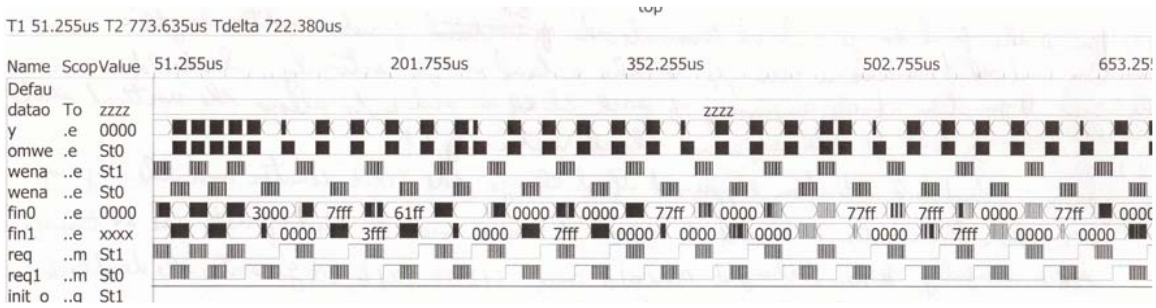

Figure 5.1: 2D-FIR Filter Data Flow Bottleneck

A data flow bottleneck example is shown in figure 5.1. The signal waveforms in row 8 and row 9 in figure 5.1 are the PMA request signal. The normal operation is to provide a processor its requesting data block whenever it raises a request signal. This normal operation is also the basic principle for a BDPA to have the linear speedup characteristic and high throughput performance. Now, the request signal is suspended after five processors obtain their data blocks. We were using six processors for this simulation. That means the IM FIFO is not big enough and is already empty before the sixth processor tries to obtain its data block for the first time.

The algorithm we used in our design for the FIR filter system is the overlap-save algorithm. With this algorithm, there is an overlapped area for each data block. Before the data block is input into the PMA, the overlapped data block must be formed. While this data block is formed, in the 1D-FIR filter system case, the overlapped area data is broadcast to both IM FIFOs simultaneously. In order to let the PMA acquires its data block first, during this broadcasting time, the data source is stopped because the FIFO cannot be read and written to at the same time. This data flow bottleneck is an unavoidable situation with the 1D-FIR filter system unless the data blocks are formed outside of the system in advance.

Several possible solutions to this unavoidable data flow bottleneck problem were considered. The traditional way to solve this problem is enlarging the IM FIFO. This solution only delays occurrence of the data flow bottleneck. Eventually, the IM FIFO

will still be empty. The second solution is changing the FIFO memory into dual port RAM memory, since dual port RAM can be read and written to at the same time. However, the read/write controller for the dual port RAM must not read and write at the same address at the same time. The read/write controller itself is another complicated design and consumes much of the wafer area and power.

A third solution is to use the dual clock rate multiprocessor system. The procedure of using a dual clock rate system is to use a very high clock rate to acquire the data block, process the data at the slower clock rate required in the processor, and output the processed data block from the processor FIFO to the OM at the original high clock rate. Since system clock rates are increasing at a very fast pace, it is simple and easy to add a clock divider in our design to implement the dual clock rate multiprocessor system to obtain high throughput performance. This method is better than adding complicated control circuitry or a large memory to the system. The linear speedup performance of the whole system was improved dramatically by using a dual clock rate multiprocessor system.

The BDPA processor architecture and its HDFC architecture are well designed to accommodate globally asynchronous, locally synchronous processing, and it is totally modularized. After the processor gets its own data block, it is isolated from the outside environment. The only interface of a processor to the outside environment is the on board FIFO. Therefore, it is feasible to use the system clock, which is faster, to input/output the data block into/from the on board processor FIFO while using another slower rate clock to drive the data through the processor computational primitives.

Since the system data flow speed is faster than the data processing speed, the data flow bottleneck problem is solved. That is why the dual clock rate multiprocessor system can obtain a large scale (double-digit) linear speedup throughput performance where the mono-clock rate multiprocessor system achieves a small scale (single-digit) linear speedup throughput performance. The detail information is shown in the performance analysis paragraph. This performance improvement also proves that the data flow bottleneck problem has great impact on the system throughput performance. The concept of using the dual clock rate to drive the BDPA is like increasing

the data block buffer but without actually adding any memory or a complicated controller in the system circuitry. We can call it a **phantom memory**. The data flow speed to the data processing speed ratio is adjustable. Therefore by adjusting this ratio and adjusting the number of processors in the system, we can obtain the targeted performance and processor utilization for the BDPA.

The dual clock rate multiprocessor system concept can be extended to a multirate clock system on chip (SoC). Different algorithms can be driven by different clock frequencies.

Chapter 6

Algorithm Partitioning and Mapping

The purpose of partitioning the algorithm is that the partitioned task can be run on different processors concurrently. This is different from parallelizing a sequential program and running it on a super computer, or parallelizing the instructions and running them on a general-purpose computer with VLIW technology. The algorithm partition strategy for the BDFP starts partitioning at the mathematical equation level while the other parallelism methods are based on the program or instructions. Note that the program or the instruction itself is normally sequential and follows a Von Neumann architecture in nature. Therefore, the cost-performance of the BDFP is expected to be better than that of the other methods.

6.1 1D-FIR Filter Algorithm

A mathematical equation is the very basis of any algorithm. It contains all the computational information we need. In order to have a concurrent processing characteristic, we need to find a way to form the mathematical equation in structural parallel form. In the BDFP, we have found a systematical method for partitioning the mathematical equation in small tasks, and for putting these small tasks into different processors to be processed concurrently.

The transfer function of a 1D-FIR filter is

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{k=0}^{L} b(k) z^{-k}, \tag{6.1}$$

where $Y(z)$ is the Z–Transform of the output data and $X(z)$ is the Z–Transform of the input data. L is the filter order and $b(k)$ represents the filter coefficients. z^{-k} is the kth delay element. The transfer function can then be expanded as

$$\begin{aligned} Y(z) &= \sum_{k=0}^{L} b(k) X(z) z^{-k}, or \\ Y(z) &= b(0)X(z) + b(1)X(z)z^{-1} + b(2)X(z)z^{-2} + \cdots + b(L)X(z)z^{-L}. \end{aligned} \tag{6.2}$$

The equation above does not give us much information on parallelism; it only describes an input data multiplied by all the designated filter coefficients with designated delays. However, if we use a simple direct implementation for the equation, the difference equation can be easily represented with the following signal flow graph in figure 6.1.

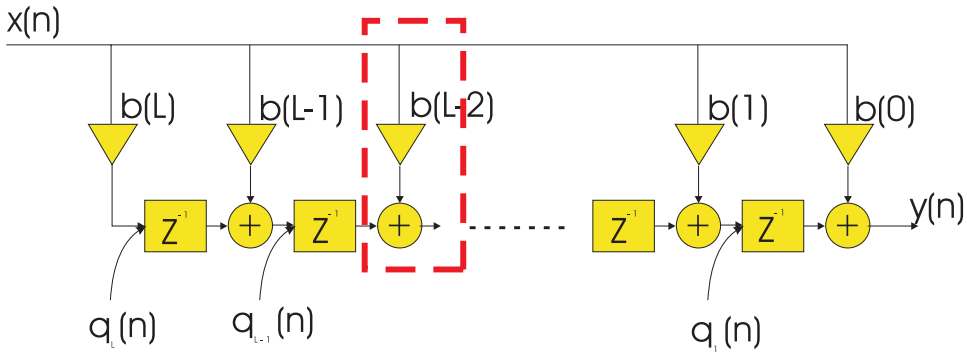

Figure 6.1: Signal Flow Graph of 1D-FIR Filter Direct Implementation

The triangle in the signal flow graph is the multiplier, whose inputs are the input data and the associated filter coefficient. There are delay elements in between all the adders. This signal flow graph can be used to simply illustrate the assignment of the state variables $q_i(n)$ for a state space representation of the filter [2, 8, 16, 26]. The position of the state variables in the signal flow graph can be located at the output of the delay elements or at the input of the delay elements. It is reasonable to implement the state variables as data stored in registers in the hardware circuit. The computed

state variables can be stored in these registers and subsequently used for the next computation iteration. Therefore, it is more logical to locate the state variables at the input of the delays.

According to the signal flow graph and the assignment of the state variables, the state variable equations are as follows:

$$\begin{aligned} q_L(n) &= b(L)x(n) \\ q_{L-1}(n) &= b(L-1)x(n) + q_L(n-1) \\ q_{L-2}(n) &= b(L-2)x(n) + q_{L-1}(n-1) \\ &\vdots \\ q_1(n) &= b(1)x(n) + q_2(n-1) \\ y(n) &= b(0)x(n) + q_1(n-1). \end{aligned} \tag{6.3}$$

6.2 2D-FIR Filter Algorithm

We also start our 2D-FIR filter system design by partitioning the mathematical equation, trying to optimize these equations in a sense of parallelism, and mapping the parallel smaller tasks into different processors. At the same time, we need to consider the partitioning of an image into blocks of data that can flow smoothly through our processors.

The 2D-FIR transfer function is

$$H(z) = \frac{Y(z_1, z_2)}{X(z_1, z_2)} = \sum_{k_2=0}^{L_2} \sum_{k_1=0}^{L_1} b(k_1, k_2) {z_1}^{-k_1} {z_2}^{-k_2}, \tag{6.4}$$

where $X(z_1, z_2)$ is the transform of the input image data, $Y(z_1, z_2)$ is the transform of the output result, $b(k_1, k_2)$ represents the filter coefficients, and z_1^{-1} and z_2^{-1} are horizontal and vertical delay elements. The filter is $L_1 + 1$ by $L_2 + 1$ in size. The order is L_1 by L_2 . If we separate the input and the output transforms on different sides of the equation, we obtain

$$Y(z_1, z_2) = \sum_{k_2=0}^{L_2} \sum_{k_1=0}^{L_1} b(k_1, k_2) X(z_1, z_2) {z_1}^{-k_1} {z_2}^{-k_2}. \tag{6.5}$$

It is not so easy to see any parallelism with the difference equations. Therefore, we expand the difference equation into a two-dimensional signal flow graph as shown in figure 6.2.

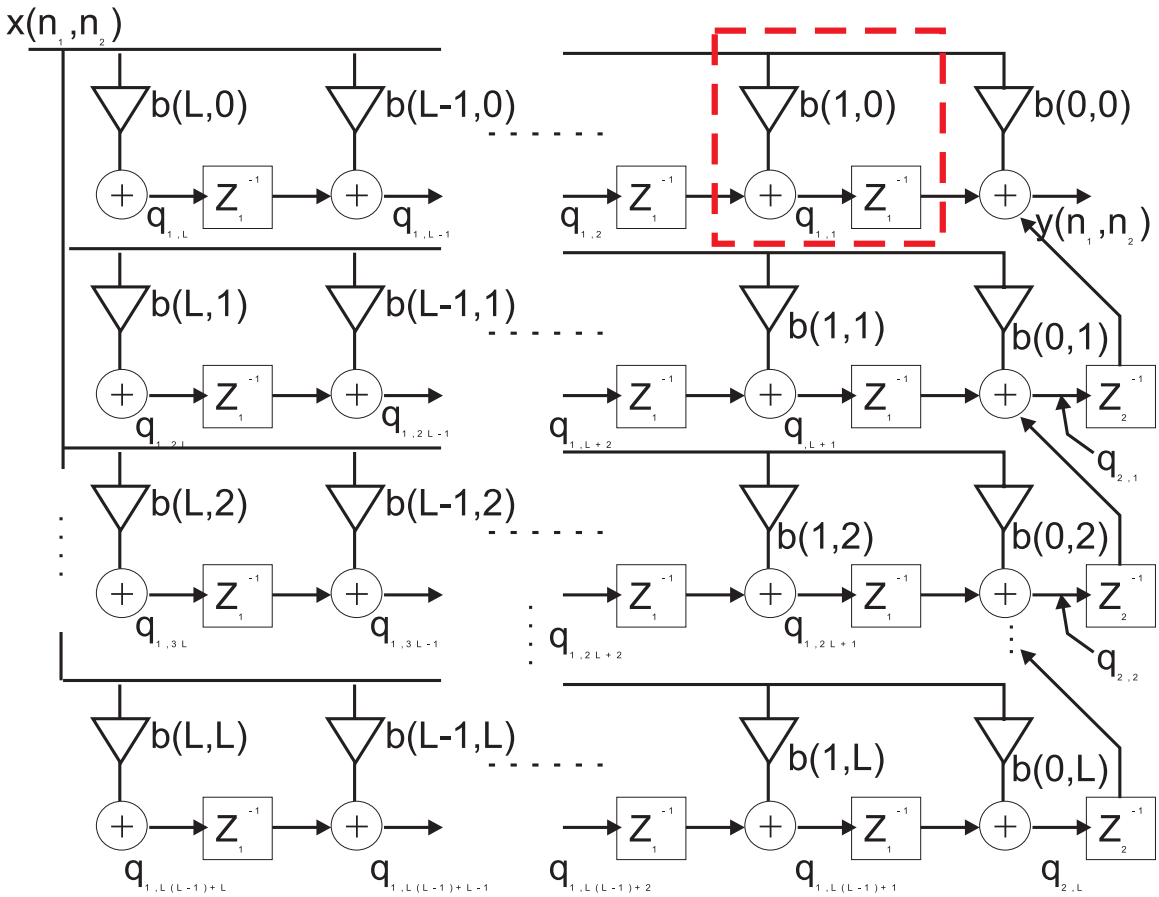

Figure 6.2: 2D-FIR Filter Signal Flow Graph

From figure 6.2, we can compute the present cycle state variable and store it into the delay element for the usage of the next cycle state variable. Therefore, the state space representation of the signal flow graph and the output equation is

$$
\begin{aligned}
y(n_1, n_2) &= x(n_1, n_2)b(0,0) + q_{1,1}(n_1 - 1, n_2) + q_{2,1}(n_1, n_2 - 1) \\
q_{1,1}(n_1, n_2) &= x(n_1, n_2)b(1,0) + q_{1,2}(n_1 - 1, n_2) \\
q_{1,2}(n_1, n_2) &= x(n_1, n_2)b(2,0) + q_{1,3}(n_1 - 1, n_2) \\
&\vdots
\end{aligned}
$$

$$
\begin{aligned}
q_{1,L}(n_1,n_2) &= x(n_1,n_2)b(L,0) \\
q_{2,1}(n_1,n_2) &= x(n_1,n_2)b(0,1) + q_{1,L+1}(n_1-1,n_2) + q_{2,2}(n_1,n_2-1) \\
q_{1,L+1}(n_1,n_2) &= x(n_1,n_2)b(1,1) + q_{1,L+2}(n_1-1,n_2) \\
q_{1,L+2}(n_1,n_2) &= x(n_1,n_2)b(2,1) + q_{1,L+3}(n_1-1,n_2) \\
&\vdots \\
q_{1,2L}(n_1,n_2) &= x(n_1,n_2)b(L,1) \\
q_{2,2}(n_1,n_2) &= x(n_1,n_2)b(0,2) + q_{1,2L+1}(n_1-1,n_2) + q_{2,3}(n_1,n_2-1) \\
q_{1,2L+1}(n_1,n_2) &= x(n_1,n_2)b(1,2) + q_{1,2L+2}(n_1-1,n_2) \\
&\vdots \\
q_{2,L}(n_1,n_2) &= x(n_1,n_2)b(0,L) + q_{1,L(L-1)+1}(n_1-1,n_2) \\
q_{1,L(L-1)+1}(n_1,n_2) &= x(n_1,n_2)b(1,L) + q_{1,L(L-1)+2}(n_1-1,n_2) \\
&\vdots \\
q_{1,L(L-1)+L}(n_1,n_2) &= x(n_1,n_2)b(L,L)
\end{aligned} \tag{6.6}
$$

The dashed square in figure 6.2 shows a possible computational primitive to be used to implement the filter.

6.3 2D-IIR Filter Algorithm

The third algorithm that is used in this paper is a second order 2D-IIR filter system. It is also a filtering algorithm, but the data computation itself is different from the previous two.

The transfer function of a 2D-IIR filter is

$$
H(z_1,z_2) = \frac{\displaystyle\sum_{j_1=0}^{L_1}\sum_{j_2=0}^{L_2} b(j_1,j_2)z_1^{-j1}z_2^{-j2}}{1 + \displaystyle\sum_{\substack{j_1=0 \\ j_1+j_2>0}}^{L_1}\sum_{j_2=0}^{L_2} a(j_1,j_2)z_1^{-j_1}z_2^{-j_2}}. \tag{6.7}
$$

The example we use is a second order 2D-IIR filter, where $L_1 = L_2 = 2$. The transfer function is expanded as

$$\begin{aligned} Y(z_1, z_2) &= b(0,0)X(z_1, z_2) + \sum_{\substack{j_1=0 \\ j_1+j_2>0}}^{2} \sum_{j_2=0}^{2} [b(j_1, j_2)X(z_1, z_2) \\ &- a(j_1, j_2)Y(z_1, z_2)]z_1^{-j_1} z_2^{-j_2}, \end{aligned} \tag{6.8}$$

and is represented in a two dimensional signal flow graph as shown in figure 6.3, where $x(n_1, n_2)$ is the input pixel data and the $y(n_1, n_2)$ is the computed output data. The coefficients a and b can be generated by using a Matlab filter design program.

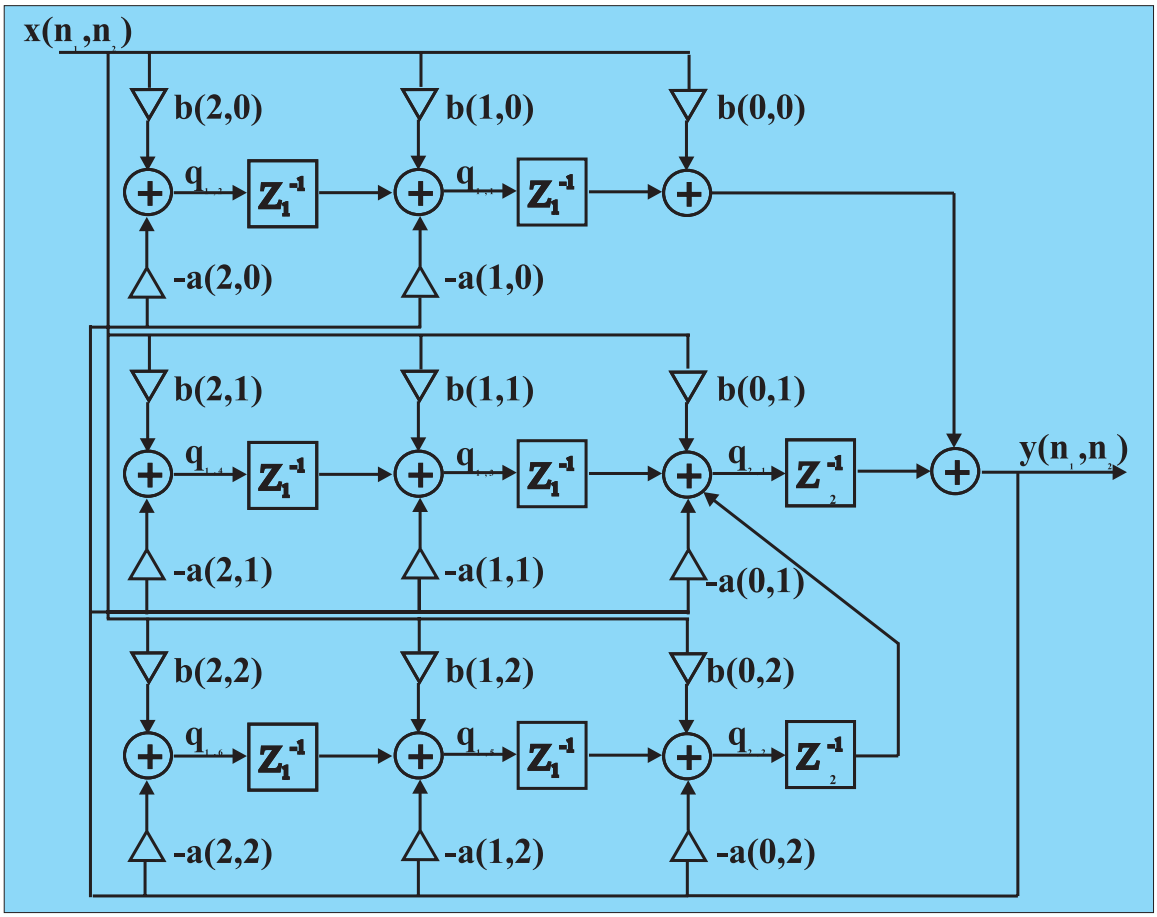

Figure 6.3: Second Order 2D-IIR Filter Signal Flow Diagram

In figure 6.3, the input of each delay is chosen as a state variable q. There can be horizontal state variable equations as follows

$$\begin{aligned}
q_{1,1}(n_1,n_2) &= b(1,0)x(n_1,n_2) - a(1,0)y(n_1,n_2) + q_{1,2}(n_1-1,n_2) \\
q_{1,2}(n_1,n_2) &= b(2,0)x(n_1,n_2) - a(2,0)y(n_1,n_2) \\
q_{1,3}(n_1,n_2) &= b(1,1)x(n_1,n_2) - a(1,1)y(n_1,n_2) + q_{1,4}(n_1-1,n_2) \\
q_{1,4}(n_1,n_2) &= b(2,1)x(n_1,n_2) - a(2,1)y(n_1,n_2) \\
q_{1,5}(n_1,n_2) &= b(1,2)x(n_1,n_2) - a(1,2)y(n_1,n_2) + q_{1,6}(n_1-1,n_2) \\
q_{1,6}(n_1,n_2) &= b(2,2)x(n_1,n_2) - a(2,2)y(n_1,n_2). \qquad (6.9)
\end{aligned}$$

The equations for the vertical state variables are given by

$$\begin{aligned}
q_{2,1}(n_1,n_2) &= b(0,1)x(n_1,n_2) - a(0,1)y(n_1,n_2) + q_{1,3}(n_1-1,n_2) + q_{2,2}(n_1,n_2-1) \\
q_{2,2}(n_1,n_2) &= b(0,2)x(n_1,n_2) - a(0,2)y(n_1,n_2) + q_{1,5}(n_1-1,n_2), \qquad (6.10)
\end{aligned}$$

and the output equation is

$$y(n_1,n_2) = b(0,0)x(n_1,n_2) + q_{1,1}(n_1-1,n_2) + q_{2,1}(n_1,n_2-1). \qquad (6.11)$$

Chapter 7

Architecture Mapping and Design

7.1 Commonality and Differences between Algorithms

The purpose of generating the state space equations mentioned in the previous section is to see how the data path is managed in the processor computational primitive. Other than considering the data path and the coefficient path in the computational primitive, the block data formation and its data flow is also another key factor to high performance computing. Since the goal of the research is to fit three algorithms into a same piece of hardware, we need to investigate the architectural commonality and differences among these three algorithms. The common parts of the algorithms can be mapped into reusable micro-architectures. These micro-architectures can be used by all three algorithms and are also good candidates to be implemented in an ASIC. The differt parts of the three algorithms are tasks for us to find ways to minimize the required system hardware architecture variations so that the three algorithms can run on the same hardware.

7.1.1 Commonality

According to the basic structure of the BDPA, all three algorithms shall have the same system operation, which is shown in figure 3.1. The system operation includes

the following operations:

- The formatted data blocks are to be distributed into the two IM FIFO buffers.
- The buffered data blocks are to be assigned to the designated processors asynchronously.
- After a data block is processed within the designated processor, it is output to the OM asynchronously.

The first operation includes the data block distribution control and the data flow control. The second operation includes the assignment control that inputs data blocks into the PMA asynchronously. The third operation includes the assignment control for the processor output. All these controls are common reusable micro-architectures for the three algorithms.

There is a computational commonality between the 1D-FIR and the 2D-FIR filter system if we compare state space equation 6.3 with 6.6, or the signal flow graph in figure 6.1 with the one in figure 6.2. Note that the one dimensional signal flow graph architecture in figure 6.1 is also used in figure 6.2 for computing the 2D-FIR filter horizontal state variables. Therefore, the processor architecture of the 1D-FIR filter system can be reused in part of the 2D-FIR filter system processor architecture.

7.1.2 Differences

The input data format of the designed filter system in our research is different from algorithm to algorithm. One dimensional data is different from two dimensional data. The filter lengths and the block sizes for each algorithm are different. Therefore, the data block formation and the data partitioning for the three algorithms are different. The second dimensional (vertical) data processing architecture in the 2D-FIR filter system is different from the 1D-FIR filter processor architecture. Since the processor architecture is different, the independent self-sufficient control on a processor is also different for these two algorithms.

The processor architecture of the IIR filter system is different from the FIR filter system. The $a(i, j)$ coefficients and the feedback intermediate value $y(n_1, n_2)$ for

the processing in the 2D-IIR filter (figure 6.3) makes its computational primitive architecture different from that for the FIR filter system. The overlap-save algorithm is used in the FIR filter system. There is no inter-communication between processors in the FIR filter system. This is different from the 2D-IIR filter system, which has inter-communication between processors. The onboard FIFO size, which is the block size, in the processor for each system is different. All the different architectures mentioned above have to be resolved in our research.

7.2 Input Module Architecture

7.2.1 1D-FIR Filter

We implemented the FIR digital filter using the overlap-save algorithm. According to the overlap-save algorithm, each input block of data is overlapped. The overlap length is equal to the order of the filter. The 1D-FIR filter simulation in this paper uses a 63 tap filter and a block size of 512 samples.

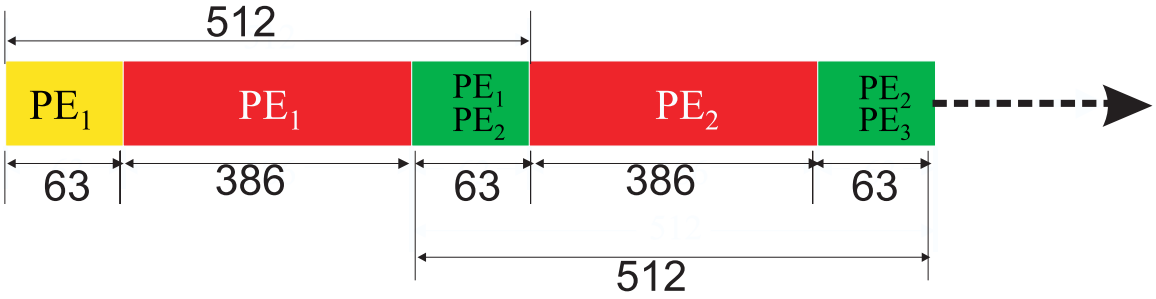

Figure 7.1: Overlapped Block Graph

Figure 7.1 shows an overlapped block graph, which explains the overlapped area of the data block and the designated processor for each block of data. In order to reduce the transient effect caused by unspecified initial conditions, we padded zeros at the beginning of the input data and at the end as well. The first block of data goes to the odd processor group, while the second block data goes to the even processor group, and so on. This means the distributor is responsible for partitioning the incoming

stream into blocks and for alternatively distributing the blocks of data into the odd FIFO buffer and the even FIFO buffer.

There are two different sizes of data in the overlapped block graph, namely 63 (N_1) and 386 (N_2). It is our desire to let the block size as well as the filter length be variable. Therefore, two variable counters are used to resolve the block size and filter length variation requirements. There is a correlation between the two variable counters due to the characteristic of the alternatively distributing data stream into two FIFOs. When the first counter counts from 0 to 62, the second counter is frozen, and vice versa. A Flip-Flop counter is designed to meet this requirement.

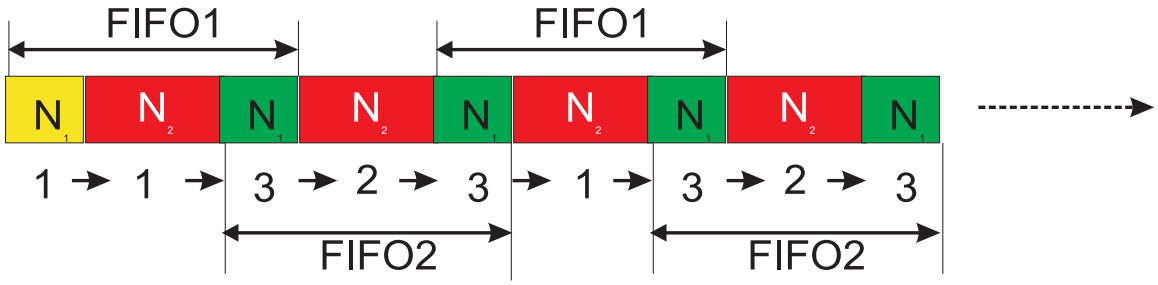

Figure 7.2: FIFO Distribution Sequence

The output of the Flip-Flop counter, which behaves like a pseudo code, is used to access data from the input device. These data must be sent to the designated FIFO. The first period of data N_1 is solely sent to FIFO1, the second period of data N_2 is sent to FIFO1, the third period of data N_1 is broadcasted to FIFO1 and FIFO2, etc. Figure 7.2 shows a detail sequence for the Flip-Flop counter distributing data into the FIFOs. A selector is responsible for selecting the correct sequence and sending the corresponding package of data into the designated FIFO. The index numbers under the graph represents the select sequence. "1" represents sending the data block into FIFO1, "2" represents sending the data block into FIFO2, and "3" represents broadcasting the data block into both FIFOs.

Figure 7.3 shows the IM block diagram, which includes the regulator. The speed of the block data flow is closely related to the system clock speed and the processing speed. If the clock speed in the PMA is faster than the IM clock speed, or there are

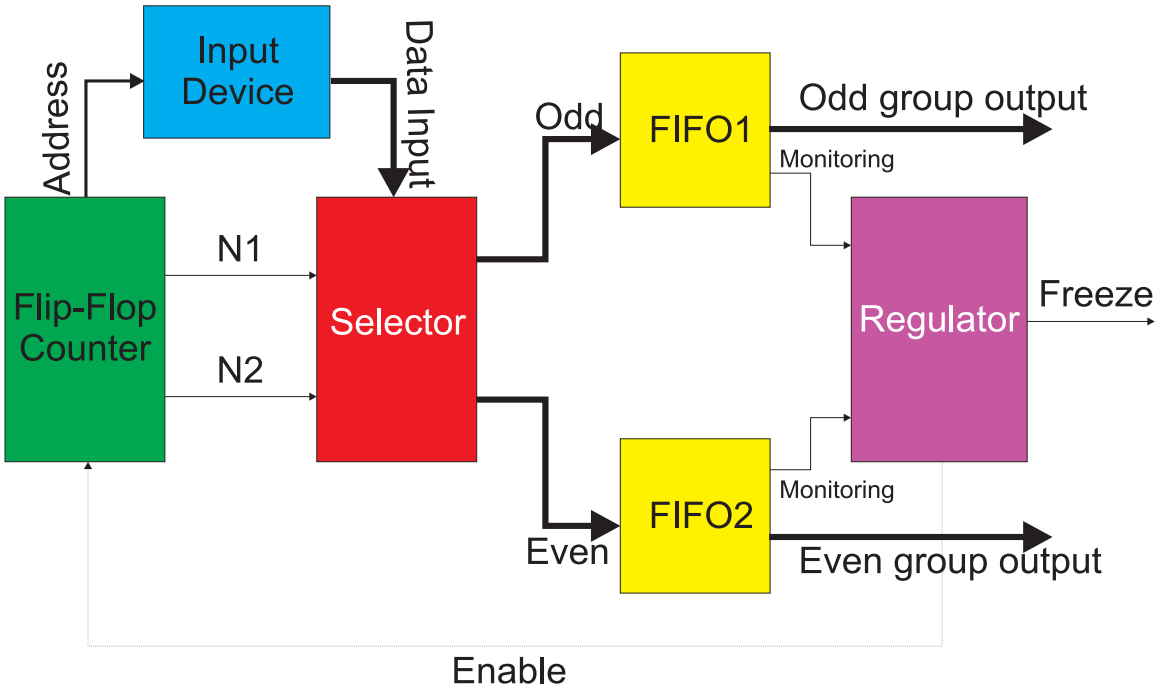

Figure 7.3: Input Module Block Diagram

more than a sufficient number of processors in the PMA, the FIFOs in the IM will be empty after some processing time. In this case, the processor, which is sending a request signal from the PMA, will stop and wait for the FIFO to be filled up with at least one block of data. At this FIFO filling time, the rest of the processors in the PMA still keep on working normally. On the other hand, if the clock speed in the PMA is slower than the IM clock speed, or the number of processors in the PMA is not sufficient, the FIFOs in the IM will be full after some processing time. Then, the system must stop the source stream from sending more data. The source stream will be released when at least one block of data is being sent into the PMA. The collaboration of the regulator, the clock speed, the block size, and the number of processors in the PMA have a direct influence on the system throughput performance. The ideal case for high performance throughput is having higher clock speed in the IM and a sufficient number of processors in the PMA, which matches the block data flow speed.

7.2.2 2D-FIR Filter

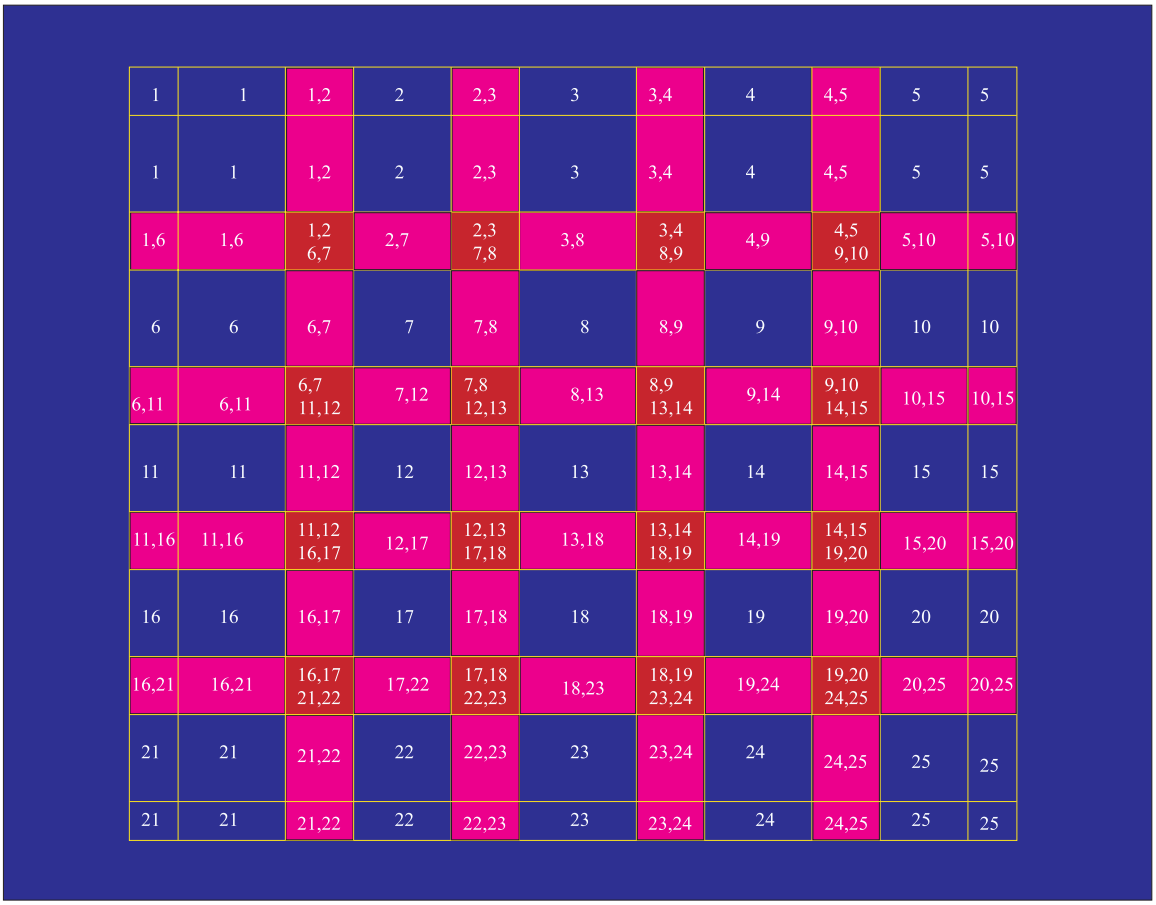

Figure 7.4: 2D Overlapped Image with FIFO Index

The architecture mapping and design for the 2D-FIR filter system is based upon the original 1D-FIR filter architecture. We will address the difference between these two architectures and its modifications. Before image data is input into different processors, the data block format must be pre-processed according to the image geometry, the filter length, and the block size that is used in the system. We will explain how image data in a raster scan bit stream is block formatted, distributed, processed, and reconstructed again in this section.

The block data distribution for a two dimensional image is more complicated than for a one dimensional signal. For the overlap-save algorithm, the most complicated

situation in the one dimensional FIR filter system is where the overlapped adjacent area covers only two FIFOs (figure 7.2), meaning the overlapped area is broadcast into two FIFOs at the same time. The overlapped adjacent area in the two dimension image filter system covers up to four FIFOs (figure 7.4). Therefore, we cannot just use two variable counters and two FIFOs to solve a two dimensional image block data distribution problem.

The goal of this research is to use the same hardware architecture to accommodate different algorithms in the same category. Therefore, it is our desire not to change too many parameters/structures in the hardware in our different algorithm mappings. Based on this principle, we need to add another image pre-processing module for the 2D-FIR filter system in front of its IM in order to preserve the original architecture in the IM.

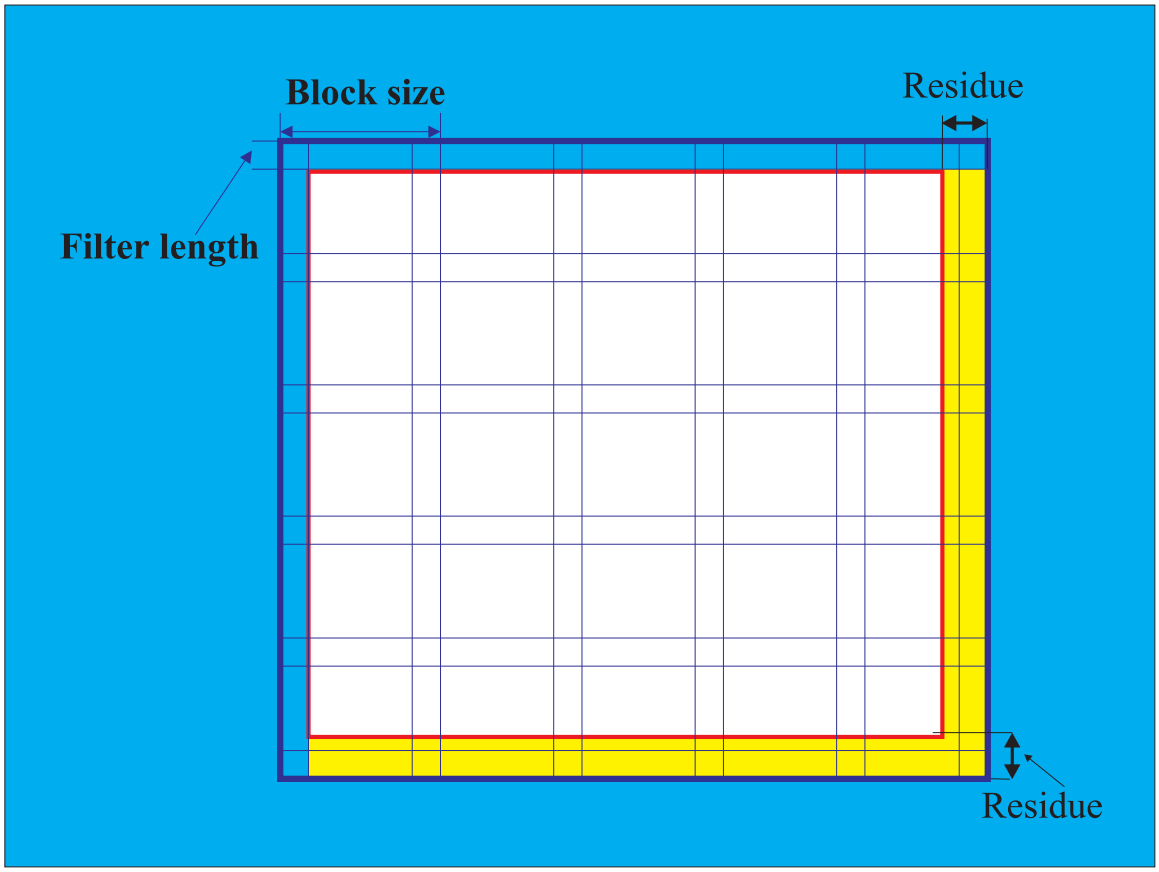

Figure 7.5: 2D Overlap Image

Normally, a video image is transmitted in a bit stream, raster scan format. Figure 7.5 illustrates the procedure to overlap a video image into a block data image and reconstruct it back to the original image later. The inner square area is the original input image. Once the block size and the filter length have been decided, the zero padding around the image, the number of data blocks in row and column, and the block overlapped area can be determined. The zero padding is to replace data that is unavailable at the edges of the image. The blocks of data are overlapped with each other and the overlapped area is equal to the filter length as shown in figure 7.5. There are five data blocks, row and column wise, in figure 7.5.

After padding zeros around the original image, the area of the image to be processed is the area that newly formed as circled by the outside square. This image is then sent in blocks, which overlap as appropriate, to the designated FIFOs. These blocks are then routed to the designated processors by the IM. Once the input image has been processed, the processor will eliminate the overlapped areas in order to reconstruct the output image which is the same size as the original image. The remaining area is the square area in figure 7.5 without the upper and the left hand side filter length area. There are still some residue areas at the right hand side and at the bottom. A post image processing module is needed to reconstruct the image back to the original image size. For the sake of speeding up the post image processing, it is better to choose the block size and the filter length, which collaborate with the original image size, in order to minimize the residual extra area. For example, if the input image size is 512 x 512, the filter length is 25, and the block size is 80, then length of one side of the zero padded overlapped image is calculated as:

$$\begin{aligned} 512(mod)(80-25) &= 9, \\ 25+(80-25)\times(9+1) &= 575. \end{aligned} \tag{7.1}$$

After the overlap-save block processing, the front end of each overlapped block image is eliminated and the overall image size is 550 x 550. In addition, we need to eliminate the real end residue, which is at the right and bottom of the image. That is, the residue on each real end side is equal to $550-512=38$. We can adjust the block size or the filter length in order to obtain a smaller residual area.

2D-FIR Image Pre-processing

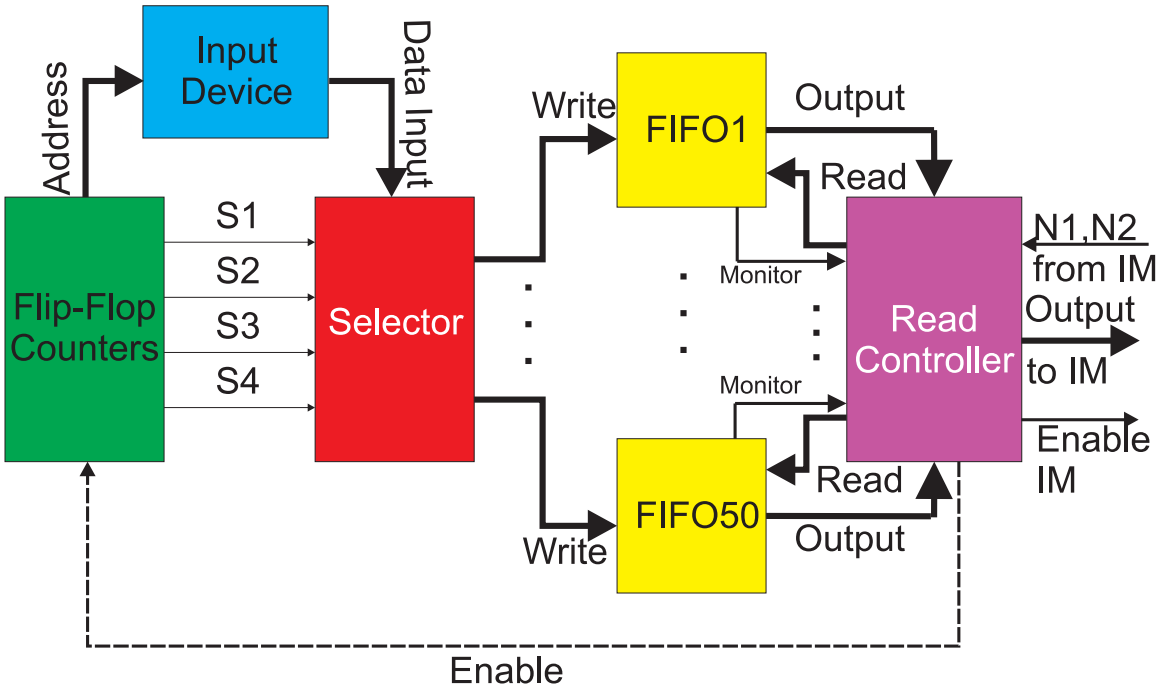

Figure 7.6: 2D-FIR Filter Image Preprocessing Block Diagram

Figure 7.6 shows a block diagram of the 2D-FIR filter image preprocessing module. This module is at the front end of the IM and is used to divide the bit stream data into data block format and to distribute them into the different FIFOs. Figure 7.4 shows an example of a two dimensional image which is divided into 25 overlapped data blocks with different FIFO indices. The number of FIFOs on the image preprocessing module is twice the number of data blocks for one image frame (i.e., 50 FIFOs for figure 7.4). There are two image frames pre-stored on the module at the initial phase. The reason to have double the number of FIFOs as data blocks on the image preprocessing module is to ensure that this module can both input and output image data at the same time. In figure 7.4, there are times when one input sample data is distributed into four different FIFOs simultaneously (i.e., FIFO1, FIFO2, FIFO6, and FIFO7 in figure 7.4). Therefore, it is not possible to use only one set of FIFOs (25 FIFOs for figure 7.4 image frame) to both output image frame data blocks and

input another image frame data blocks at the same time. Due to the large memory size in the image preprocessing module, we suggest that to separate this module from the designed filter system and to form it as another sub-system.

In figure 7.4, both horizontal and vertical directions of the image have two different data lengths. Therefore, it is feasible to use two Flip-Flop counter pairs to implement the data block formation and distribution. The first Flip-Flop counter, which is the same as in the 1D-FIR filter system generates signal $S1$ and $S2$ in figure 7.6, counts the two data lengths horizontally. The second Flip-Flop counter generates the signal $S3$ and $S4$, records the two data lengths vertically. A Finite State Machine (FSM) controller group is responsible for managing the selector to distribute the assigned data block into the designated FIFO. The read controller monitors the status of the FIFOs and generates the enable signal to the IM. This enable signal allows the IM to output the $N1$ and $N2$ signals from the IM and read the data block accordingly from the proper pre-stored FIFO to the IM.

The following pseudo code explains how this sub-system works with the IM:

1. At initial phase, if FIFO50 is full, then freeze the data source and enable the IM to access to proper FIFOs.

2. Else if more than one of the FIFOs is empty, then release the data source.

This sub-system only works properly under the condition that the data block storing speed is faster than the retrieving speed. If it happens the other way around, then the whole system must be stopped. This condition is achievable by adjusting the dual clock ratio and it is also consistent with the BDFP basic requirement, which is to provide sufficient data block for the processor array.

Another alternative design is to send the image preprocessing output to a memory buffer, allowing the IM to retrieve the proper data block from there. The size of this memory buffer can be the size of one overlapped image frame. The FIFO in the image preprocessing module can be used repeatedly once it is emptied by the IM. In doing so, the number of FIFOs in the image preprocessing module can be reduced to a reasonable smaller value.

Pseudo Code Counter

The principle of designing the 2D-FIR filter IM is to modify the 1D-FIR filter IM and preserve its architecture as much as possible. The 1D-FIR filter system IM has a specially designed flip-flop counter. There are two variable counters in the flip-flop counter, which counts two different numbers. These two different numbers are used to control the selector and to distribute different lengths of block data into different FIFOs. For the case of the 2D-FIR filter, the block data sizes in the FIFOs are all the same in the image pre-processing module. We just transfer the blocked data in the image pre-processing module FIFO to the IM FIFO. Therefore, it is not necessary for the two counters, which are in the flip-flop counter, to count different numbers. This is the first modification we need to make in the 2D-FIR filter IM: setting the two counter numbers equal to the same block size.

Selector and Distributor

The next modification in the 2D-FIR filter IM is the selector controller. The control sequence for the 1D-FIR filter system, which distributes data blocks in different lengths to two different FIFOs, is $1 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 3$ and so on. The number '1' in the sequence represents FIFO1, '2' represents FIFO2, and '3' represents both FIFO1 and FIFO2. The reason sequence 3 exists is that, in the 1D-FIR filter, the input block data is sometimes simultaneously distributed to both FIFO1 and FIFO2. However, the input block data is distributed specifically to either FIFO1 or FIFO2, not both at the same time for the 2D-FIR system. Therefore, we need to modify the FSM controller sequence as $1 \rightarrow 2 \rightarrow 1 \rightarrow 2$, and so on. The pseudo code generated from the two counters in the flip-flop counter will retrieve the proper input data block from the FIFOs in the image pre-processing module. These input data blocks will then be distributed into two FIFOs in the IM by the FSM controller and the selector. The selector architecture is exactly the same as that used in the 1D-FIR filter IM.

Regulator

The purpose of designing a regulator in the IM is to guarantee that the IM provides sufficient data to the PMA whenever the PMA requests. The regulator is a two way controller, which is situated between the image pre-processing module and the PMA, and is in charge of the data flow. The regulator monitors the status of the IM FIFO. If the data flow speed is faster than the processing speed, both IM FIFOs will be full of data from time to time. It is necessary at that specific time to disable the Flip-Flop counter pseudo code generator, which retrieves data from the image pre-processing module. Thus, the streaming data source is regulated by the IM regulator.

The FIFO in the IM cannot be read from and written to at the same time. Therefore, in order to handle the read and write situation at the same FIFO, a special circuit is designed to always give the priority to feed the PMA with sufficient data, thus stopping the data source from writing data into the IM FIFO and allowing the IM FIFO to output one block of data into the PMA processor when both the data source module and the PMA are trying to access the same IM FIFO. There are two quality control counters, which monitor exactly one block of data being output from each individual IM FIFO. The size of these counters must be exactly one data block and therefore, is modified accordingly with the block size.

The on board FIFO architecture in the 2D-FIR filter system is different from that in the 1D-FIR filter system. The 1D-FIR filter system processor is a one-level hierarchical processor while the 2D-FIR filter system processor has a two-level hierarchy. The 2D-FIR filter system has to divide one block of data into smaller row blocks and put them into second-level hierarchical row-processors. Thus, for the 2D-FIR filter system, the IM data block quality control counter size is the size of the second-level hierarchical row-processor FIFO.

It is obvious that changing the 1D-FIR filter IM into the 2D-FIR filter IM is very easy. All we have done is change all the variable counter parameters and the distributor control sequence. All the other components are the same. The goal of designing the architecture to be flexible, reusable, and simple is achieved for the IM.

7.2.3 2D-IIR Filter

2D-IIR filter system input data is a two dimensional image. The image is input into the system in a raster scan bit stream format. There are many ways to divide the input image into blocks of data. In order to simplify the system control, the input image is divided in a way that each block of data contains only one row of image data. Therefore, all the data blocks are the same size. There is no block overlap situation as in the 2D-IIR filter system.

According to the IM design experience, the architecture described in the previous two algorithms, the two counters in the flip-flop counter will count the same number and distribute the same size data block into the two IM FIFOs. The regulator does the same function as described in the two previous FIR algorithms. Therefore, the 2D-IIR filter IM architecture is exactly the same as the one used in the 2D-FIR filter system.

7.3 Processor Module Array Architecture

7.3.1 1D-FIR Filter

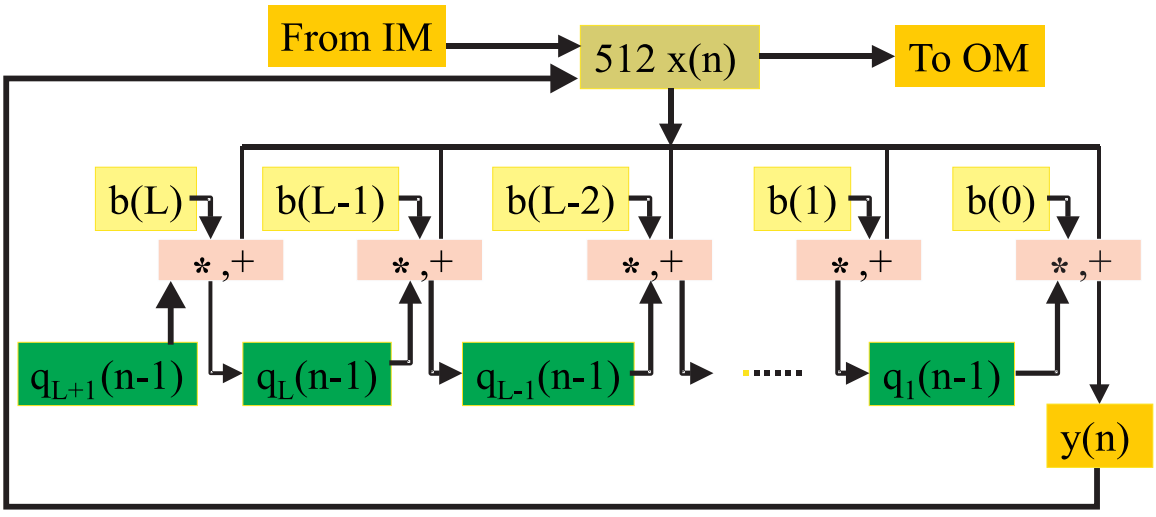

Figure 7.7: Processor Block Diagram

The processor is the heart of the whole system. It contains the information as to how the data is being processed, it contains the information on how the data flows through the processor, and it determines when the data is needed.

In figure 6.1, $x(n)$ and $y(n)$ correspond to input data and the processed output data. The dashed rectangle includes a multiplier and an adder. This is interpreted in figure 7.7 as '$*, +$' computational module. There are state variables $q_i(n)$ ahead of each delay component 'z^{-1}'. The input of the multiplier is the input data and the proper filter coefficient $b(j)$. The multiplier result and the delayed state variable are the inputs of the adder. This delayed state variable is a processed result from the previous sample and its iteration. Therefore, it is practical to store the state variables (the adder output) in the register '$q_i(n-1)$' in figure 7.7.

In figure 7.7, the block data is distributed from the IM and stored in the on board memory FIFO ($512x(n)$), which has exactly the size of one block. The example block size in this paper is 512 samples. $q_x(n-1)$ and $y(n)$ are registers to store the state variables and the output results. The '$*, +$' computational modules calculate the proper state variables and the output result. The '$b(L)$' are coefficients, which are stored in the memory. The on board memory FIFOs are located on each processor in the PMA. Once the FIFO is filled with one block of data, the processor is isolated from the rest of the system and it starts processing its data. The data path, coefficient path and the state variable path management are controlled by a FSM controller. The output data $y(n)$ uses the same FIFO memory as used for the input data. The FIFO will start outputting its computed block of data to the OM when the whole block of data has been processed. Using the same FIFO memory to store the input and output block data saves a lot of wafer area compared to using a separate input and output FIFO architecture [28]. This is practical because we use the state space model. The block data flow in or out of the processor is controlled by the Token Passing Controller (TPC).

The block of data in each processor is overlapped. Therefore, after data computation, it is necessary to delete the overlapped data in order to reconstruct the correct output. It is rather easy to delete the overlapped data by using a counter. The counter is activated right after the whole block of data is processed and stored

in the FIFO. The resulting block of data will not be output until the counter reaches its fixed number, which is equal to the filter length. Thus, the first part of the output block, which is the overlapped part, is eliminated in the final output.

Token Passing Control

It is well known that the use of synchronous control in a systolic array system results in problems due to clock skew [2]. The BDPA system uses asynchronous control to avoid problems with clock skew. This eliminates the need to use a complex architecture like the clock manager in the newest product of Xilinx's Virtex II FPGA board. We refer to the asynchronous control manager for the BDPA as the Token Passing Control (TPC).

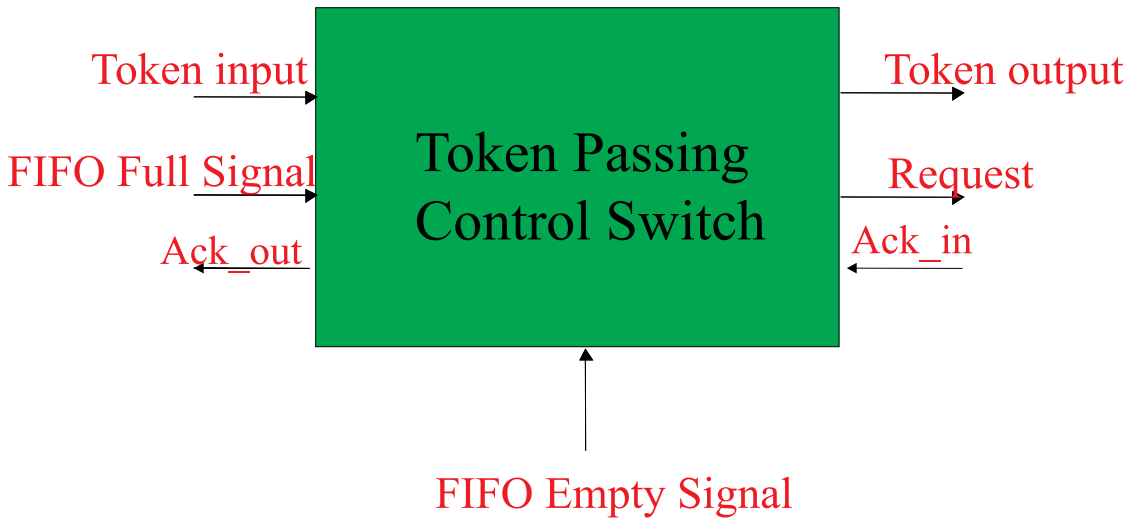

Figure 7.8: Token Switch Block Diagram

Block data in the IM FIFO is sent through buses to the designated processor under the control of the TPC. Only the processor with the token has the right to access the bus and download its block of data. The system processors are connected in a one directional, one dimensional ring type architecture. The token is generated at the first processor while the system is being reset. Each processor sequentially passes the token to the next processor after it has received its assigned block of data. Figure 7.8 shows a token switch block diagram. The switch, which is located on the processor

board, cooperates with the individual processor. It senses the status of the on board memory FIFO and accesses the bus for block data according to the token. When the FIFO is empty and the token is one, the switch will output the request signal and wait for the block of data. The processor will reset the request signal and the token, and output the token to the next processor when the FIFO full signal arrives. The next processor receives the token, and then sends an acknowledging signal to the present processor to reset the token output signal. The processor will start its data processing and continue until the processed block of data in the FIFO is sent to the OM. The processor then waits for the arrival of another token.

7.3.2 2D-FIR Filter

Rational for Using a One Dimensional Processor Array

The input image of the 2D-FIR filter system is partitioned according to the geometry, the block size, and the filter length, and is divided into data blocks also containing two dimensional information. These data block sub-images overlap with each other at the surrounding edges as shown in figure 7.5 and figure 7.4. It is natural using a two dimensional array processor (i.e., systolic array) to process the two dimensional sub-image. However, we intend to use a one dimensional processor array in order to avoid clock skew problems, larger chip size, more power consumption, and pin complexity. The BDFP supports a one dimensional array to achieve high performance.

A very important issue in the 2D-FIR filter processor (block processor) is its FIFO size, the data block size, and the way the data block is formed. The size of the FIFO and where to put the FIFO memory normally depend on how the data block is partitioned. The FIFO size is not necessarily equal to the data block size. If the FIFO size is smaller than the data block size, then there must be accommodation such as a block of data being distributed across two FIFOs, or two processors. This situation increases the complexity of the controller, which should be avoided in the BDFP design. It is workable if the FIFO size is greater than the block size. But, just for the sake of a simpler controller, it is convenient to have them both the same size.

In the 1D-FIR and 2D-IIR filter system, there is only one FIFO, which contains exactly one block of data, on each processor. In the 1D-FIR filter, the data is partitioned in blocks, which contain only one dimensional information (figure 7.1). Each individual processor will process its own data block without any communication between other processors due to the orthogonal characteristic (data computation independency) between the processor computations. Therefore, the block size FIFO is on a processor board and the PMA of the 1D-FIR filter system is in a one dimensional processor array architecture.

In the 2D-IIR filter system, even though the data is a two dimensional image, we partition the data blocks in such a way that one block of data consists of only one row of pixels of an image frame. That is, there is only one dimensional information on each processor data block and the processor will process only one dimensional information. The second dimensional information is passed between processors. Therefore in the 2D-IIR filter system, each processor also has a block size FIFO and its PMA architecture is a one dimensional processor array architecture.

In the 2D-FIR filter system, it is important to keep the one dimensional processor array architecture so that the same piece of hardware can accommodate three different algorithms. With this requirement in mind, the data block is partitioned into sub-images, which overlap as shown in figure 7.5 and figure 7.4. There is two dimensional information in each data block. This means if we still want to keep the concept that one processor processes one block of data, we will have to deal with two dimensional information within one processor.

According to the description in the last paragraph, a one dimensional row of data can be processed in a processor and the second dimensional information can be passed between the processors. Therefore, it is clear that for the sub-image partitioned in the 2D-FIR filter system, a sub-image row can be processed in a row-processor and its column information can be passed between the row-processors. In figure 7.9, a block processor contains the row number of a sub-image of row-processors and each row-processor has its own on board FIFO. Each sub-image row is input from the IM FIFO onto the row-processor FIFO. Each row-processor has its own TPC. The token is passed from one block processor to another through each individual row-processor's

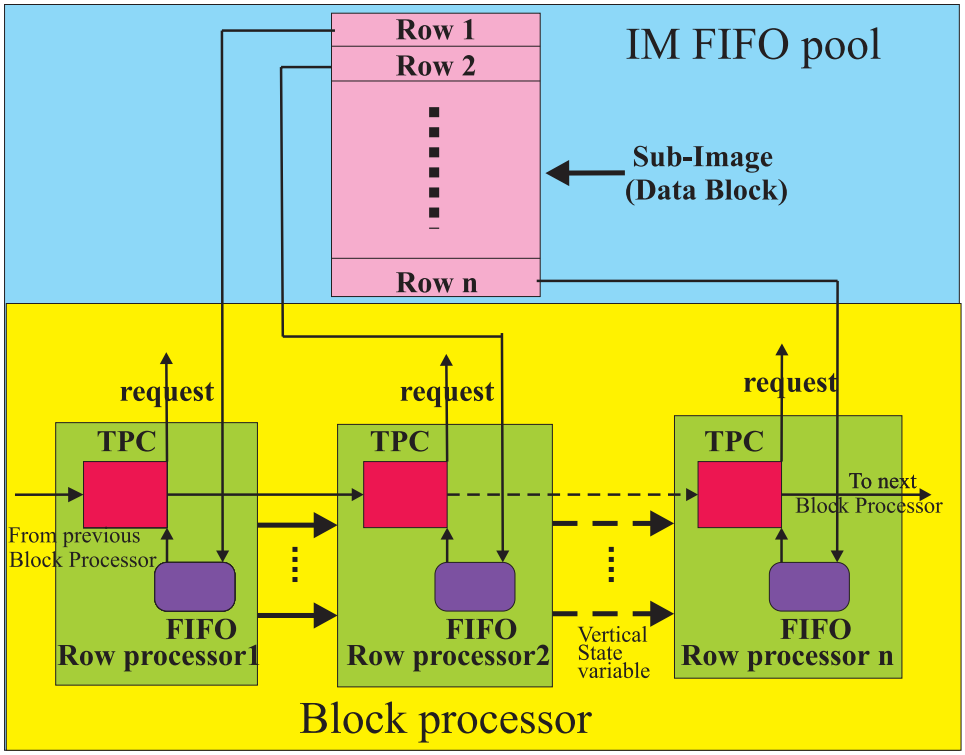

Figure 7.9: Sub-block Data Distribution

TPC.

The overlap-save algorithm is also used in the 2D-FIR filter design. There is no data dependency (data block processing is orthogonal) between each data block. There is no inter-communication required between each block. The PMA architecture for the 2D-FIR filter system is still a one dimensional processor array.

Two Level Hierarchical Processing

Since the sub-image is two dimensional, according to figure 6.2 and equation 6.6, there will be horizontal state variable as well as vertical state variable computations. Note that the configuration of the row wise signal flow diagram in figure 6.2 is the same signal flow diagram configuration in figure 6.1. This means the 1D-FIR filter processor architecture can be used as the 2D-FIR filter processor computational primitive. Its register architecture, data and coefficient path control, and the finite state machine

controller are the same as used in the 1D-FIR filter system. Thus, the 2D-FIR filter horizontal data processing can be done by the 1D-FIR filter processor.

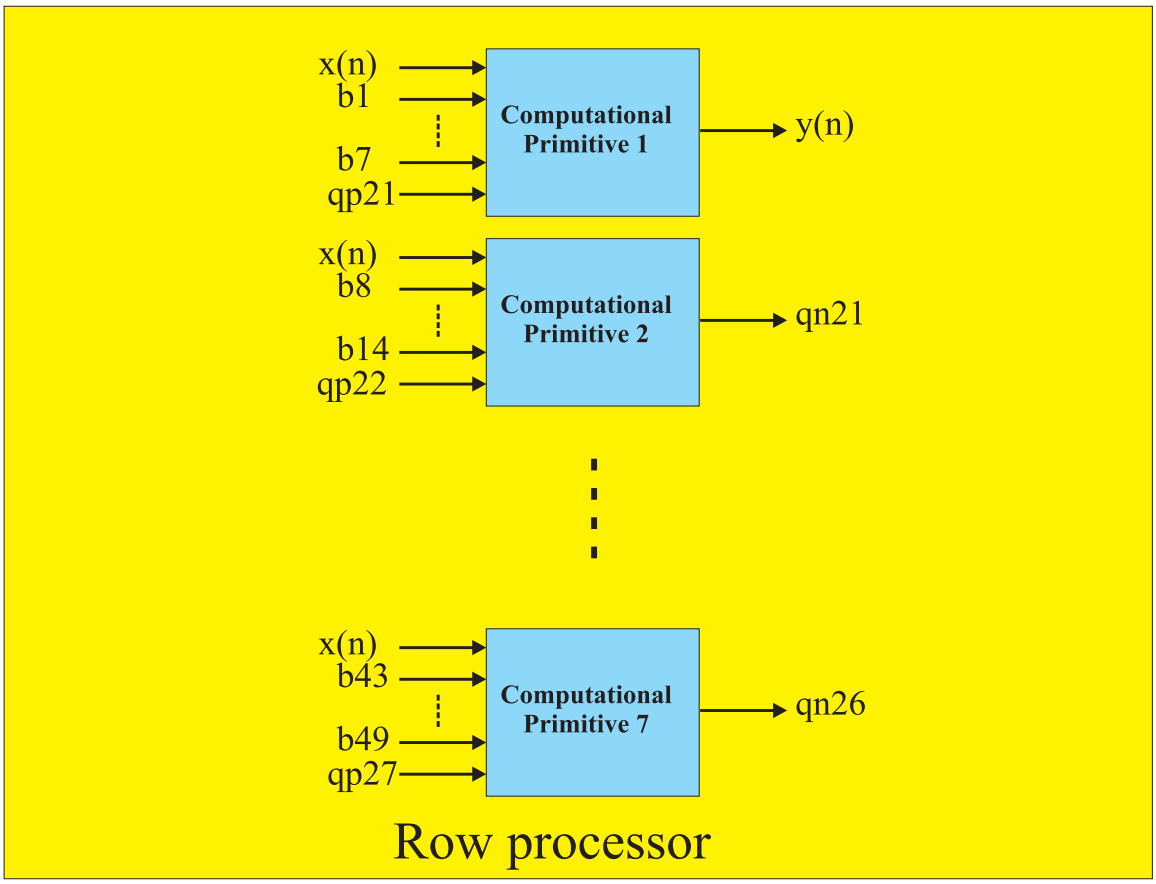

Figure 7.10: Row-processor Example

In figure 6.2, if the filter order is L, then there are $L+1$ row wise signal flow diagrams. This also means that there are $L+1$ computational primitives in the 2D-FIR filter row-processor. Figure 7.10 is a row-processor example for the 2D-FIR filter with a filter order $L = 6$. $x(n)$ is the input sample data, b_i are filter coefficients, $qp2i$ are the vertical state variables which are passed from the previous row-processor, $y(n)$ is the processed data output, and $qn2j$ are the vertical state variables that are to pass to the next row-processor.

The vertical state variables are passed between rows in the sub-image after one row of filtering is finished. The row computations are done in the row-processor

(figure 7.10). The vertical state variables are passed between row-processors (figure 7.9). Thus a block-processor, which processes the horizontal state variable as well as the vertical state variable, is a two-level hierarchical processor. The number of row processors in a block-processor is equal to the number of rows in a sub-image (data block).

Four-section Computational Primitive Design

The number of different data types, such as 8-bit, 16-bit, or 32-bit arithmetic operations is always an important concern in multimedia processing. Techniques such as multimedia extensions (MMX) [23, 24], which are used in most VLIW processors, superscalar processors, or vector processors [18], normally implement more than one data type in their systems. They need a more complicated controller to achieve their purpose. It is our design principle to implement the system with circuits as simple as possible.

64-bit arithmetic operations in computer architecture are now popular. Many companies have such products on the market already. However, as the bit size for the hardware architecture increases, the power consumption as well as the circuit complexity increases. This is acceptable for a general purpose computer but is not acceptable for many applications including mobile devices (i.e., wireless PDA). Besides, the precision of using 16-bit arithmetic operations for media processing is sufficient. Therefore, we stick to the use of 16-bit integer arithmetic operations for our hardware architecture design.

One Dimensional Signal As shown in the figure 6.1, only horizontal state variables $q_x(n)$ are used for the direct implementation of the 1D-FIR filter. The dashed circle encloses a ***one-section computational primitive*** (multiplication addition accumulator), which has two 16-bit inputs to the multiplier. The results for the multiplier is a 32-bit word and it is input into the adder later. After the two 32-bit numbers are added together, the result is a 33-bit integer. This result is then scaled back to a 16-bit integer, stored in a register, and prepared for the next computational

primitive computation. There is only one level of computational primitive hierarchy in the 1D-FIR filter system, namely, the horizontal state variable hierarchy.

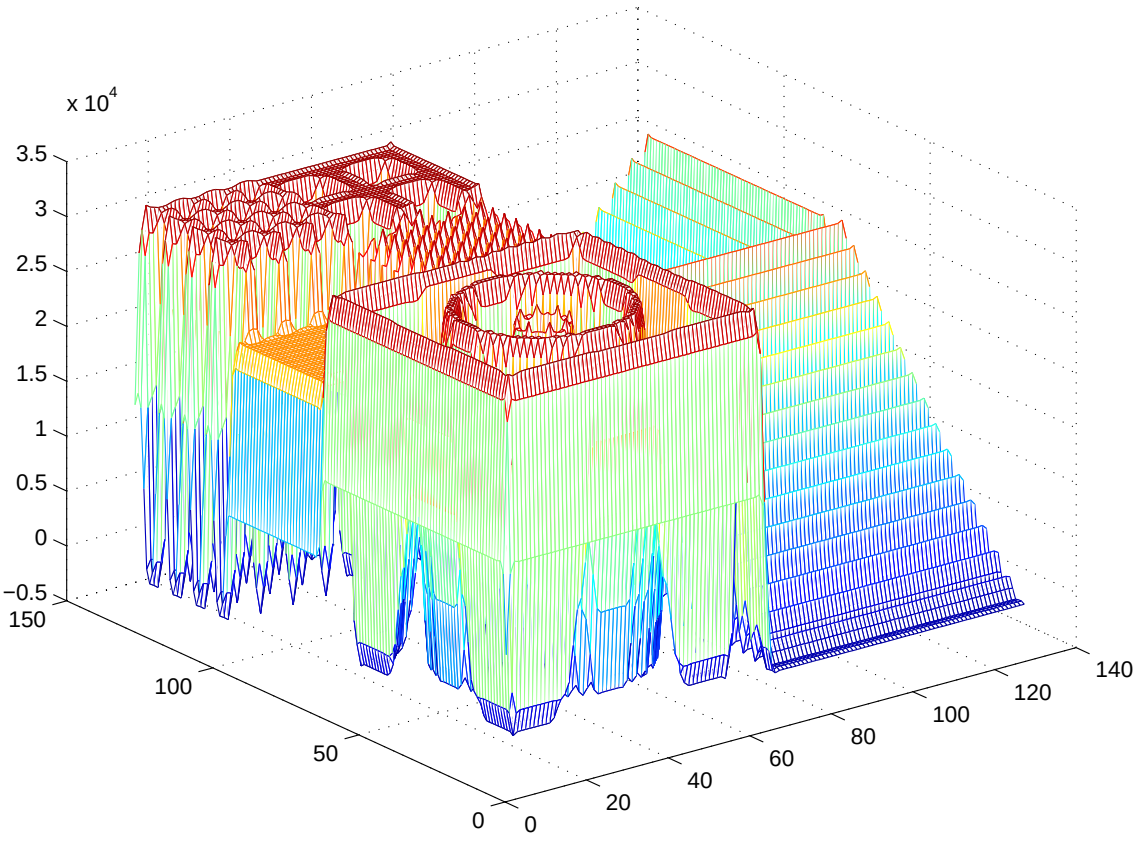

Figure 7.11: 2D-FIR Filtered Image using Floating Point Computation

Two Dimensional Signal A 2D-FIR signal flow graph is shown in figure 6.2. There are two hierarchies of computation in a block processor. One hierarchy is for the horizontal state variable and the other is for the vertical state variable. A 2D-FIR filtered test image using floating point computation is shown in figure 7.11. If we use one-section computational primitives to process the two-dimensional image, after all the computational stages, we will get an output image with round off errors as shown in figure 7.12. We can exchange the original one-section computational primitive operation with a higher order computational primitive operation to reduce roundoff errors.

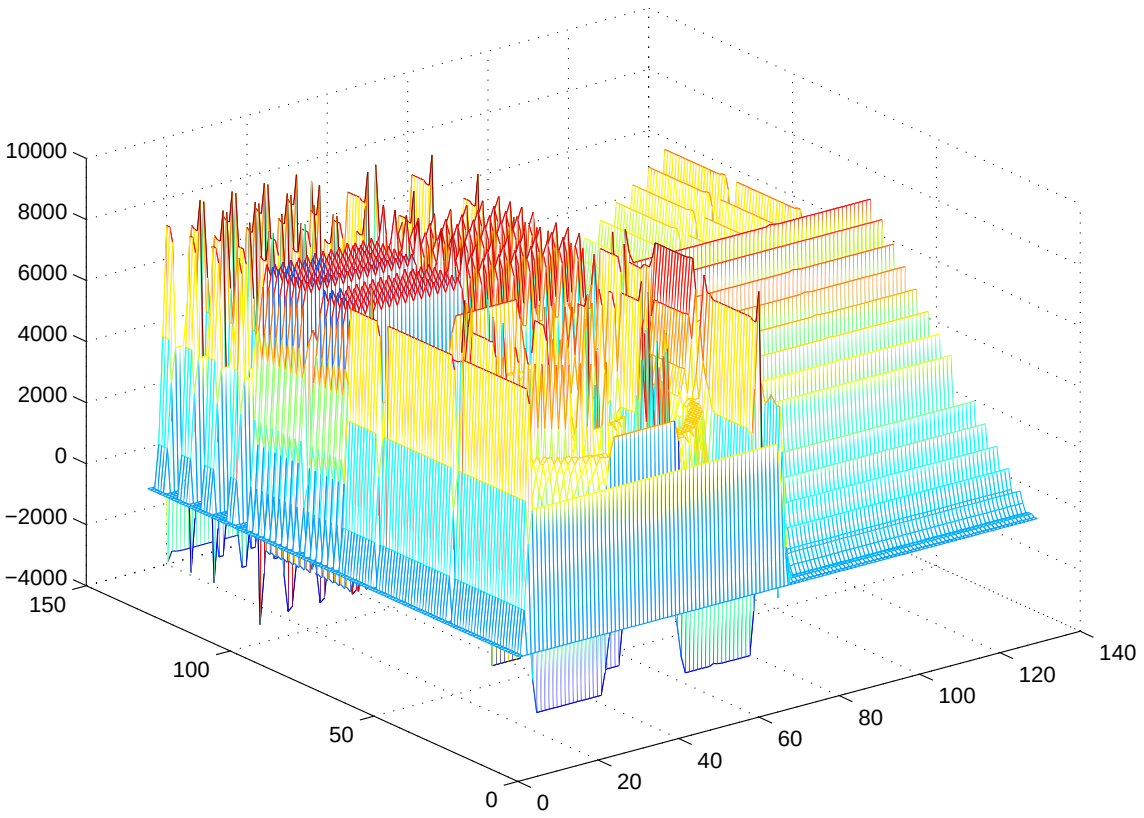

Figure 7.12: 2D-FIR Filtered Image using One-section Computational Primitive Operation

According to figure 6.2, there are several ways to modify the operation in the computational primitive. It is easily recognized that a modification of vertical state variable computation hierarchy could not solve the round off and overflow error since every vertical state variable includes one row of horizontal state variable computations. The number of computations for one horizontal state variable row is equal to the image filter length. If we modify the vertical state variable computation but don't modify the architecture for the horizontal state variable computation, the results would be changed very little. Therefore, we choose to modify the horizontal state variable computation hierarchy.

The universal arithmetic operation in our system is a 16-bit integer operation. We can modify the computational primitive with a two-section, three-section, or four-section computation operation, that is, scaling back to the universal arithmetic

operation (16-bit integer) after using a two-section, a three-section, or a four-section computational primitive. The reason we choose four-section computational primitive is that we select the filter order which is six. As a result, there are seven multiply-add operations in a row. The processing of one pixel data through the row operation can be finished within two clock cycles, using a four-section computational primitive. The processed precision is within the dynamic range of our arithmetic operation. This is well explained in the following example.

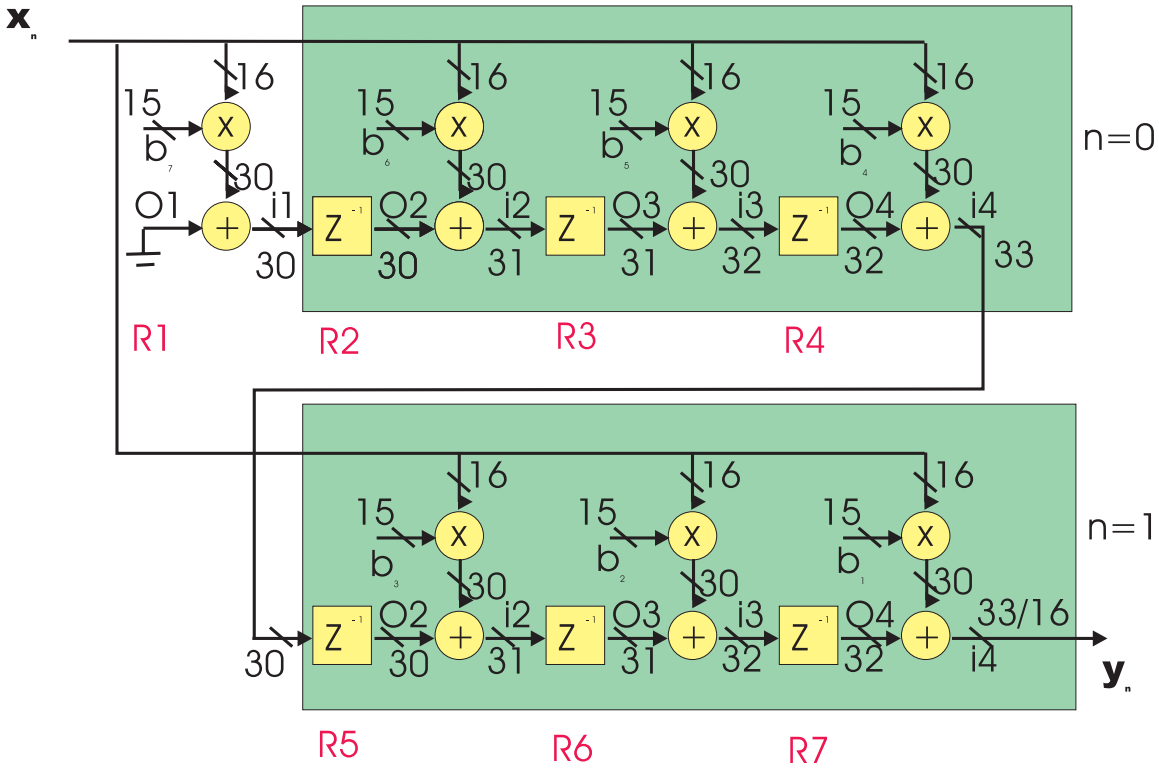

Figure 7.13: Four-section Computational Primitive Signal Flow Graph

Figure 7.13 shows a one row horizontal state variable signal flow diagram with four-section computational primitive notations on it. Once the filter coefficient length is decided, the coefficient values are decided according to the filter frequency response. This value will not change during the filtering process. Therefore, we can use a 15-bit integer representation, which is computed with Matlab in advance, stored in the memory and retrieved to the computational primitive input while it is needed.

The reason for using the 15-bit integer representation is to make the processor input operand as small as required and to avoid the possibility of having the processing results overflow. Normally, a 16 bit number multiplied by a 15 bit number yields a 31 bit result. However, we can ensure a 30 bit result by not allowing the inputs to have the most extreme negative inputs. This is -32768 for a 16 bit number and -16384 for a 15 bit number.

Before designing a circuit for a module, it is advantageous to think thoroughly about the design and make it reusable if feasible. The definition of a reusable module design here is to design a module using the same parts at different times and let the controller manage the data path between the memory and the processor primitive. Whether it is a small module or a large circuit block, reusable circuits reduce power consumption and wafer size and speed up the reconfiguration time (if the design is implemented on configurable components).

In order to have a four-section computational primitive, we have four different adders with different input/output bit-widths, and we have one type of multiplier as shown in figure 7.13. The upper part of the signal flow graph, where n=0, represents the first clock cycle operation. Data as well as the designated filter coefficients will be input into four different sections of the computational primitive. Their output is stored in the designated registers. The number of the registers, where the intermediate horizontal state variables are stored, is equal to the filter length. The data, which is stored in these registers, have different bit widths, meaning the registers are connected with different bit width buses.

The lower part of the signal flow graph, where n=1, is the second clock cycle operation. The data and the designated filter coefficients are input into the same three section computational primitive, which has been used during the first clock cycle as shown in the dark square block in figure 7.13. The outputs are stored in the designated registers or sent to the next row-processor as the intermediate vertical state variable or as the output data.

The four-section computational primitive data path controller can be implemented with a Finite State Machine (FSM). It is important to manage the input data and the designated filter coefficients so that they arrive at the associated processor primitive

ports at the specific time. For example, the delay that occurs at the FSM where the previous state variable is generated, the retrieving of the data, and the retrieving of the designated filter coefficient must all be matched carefully. Figure 7.14 shows an example of a FSM controller, which is designed according to the signal flow graph in figure 7.13. At n=0, the data in registers one through four, which are the pre-

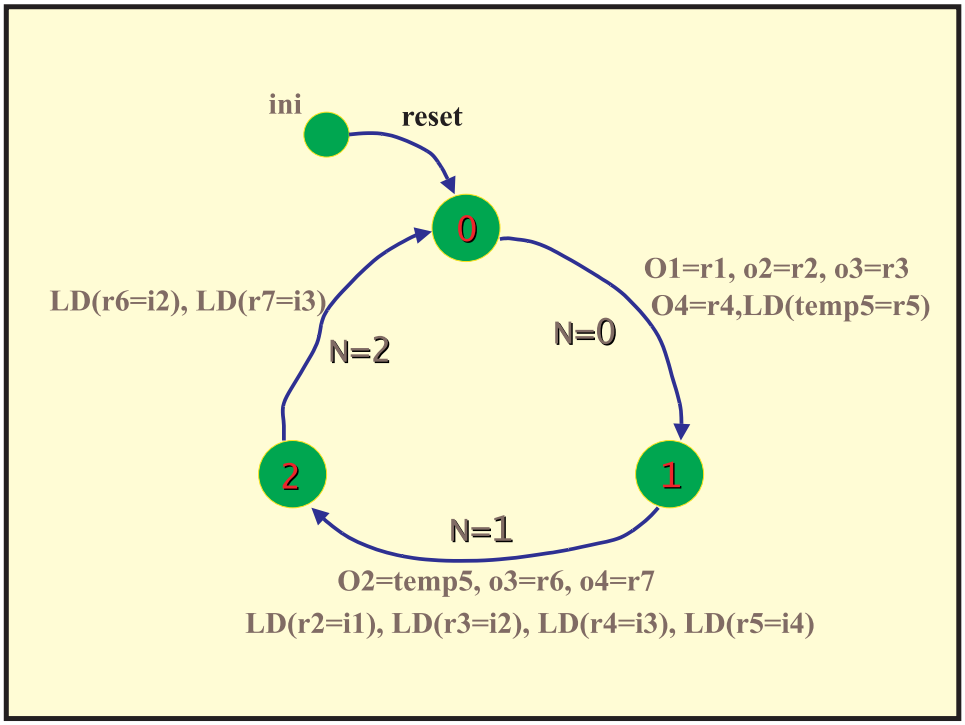

Figure 7.14: Four-section Computational Primitive Finite State Machine Controller

vious horizontal state variables, are output from the FSM as inputs to the adder. Meanwhile, the correspondent input data and the filter coefficients are already at the input of the multipliers. When n=1, the computed horizontal state variables are stored back into registers two through five, ready for the next cycle computation. While the values in the register five through seven are output to the lower part of the four-section computational primitive, its return intermediate state variables will be stored into registers six and seven when n=2. $\boldsymbol{n}$ is a count number. The concept behind this counter is that for an asynchronous module/processor, the module has its own special cycle (if a special module is designed), which is generated from the

basic system clock. Every different module has its own cycle and works accordingly with asynchronous/data-flow control.

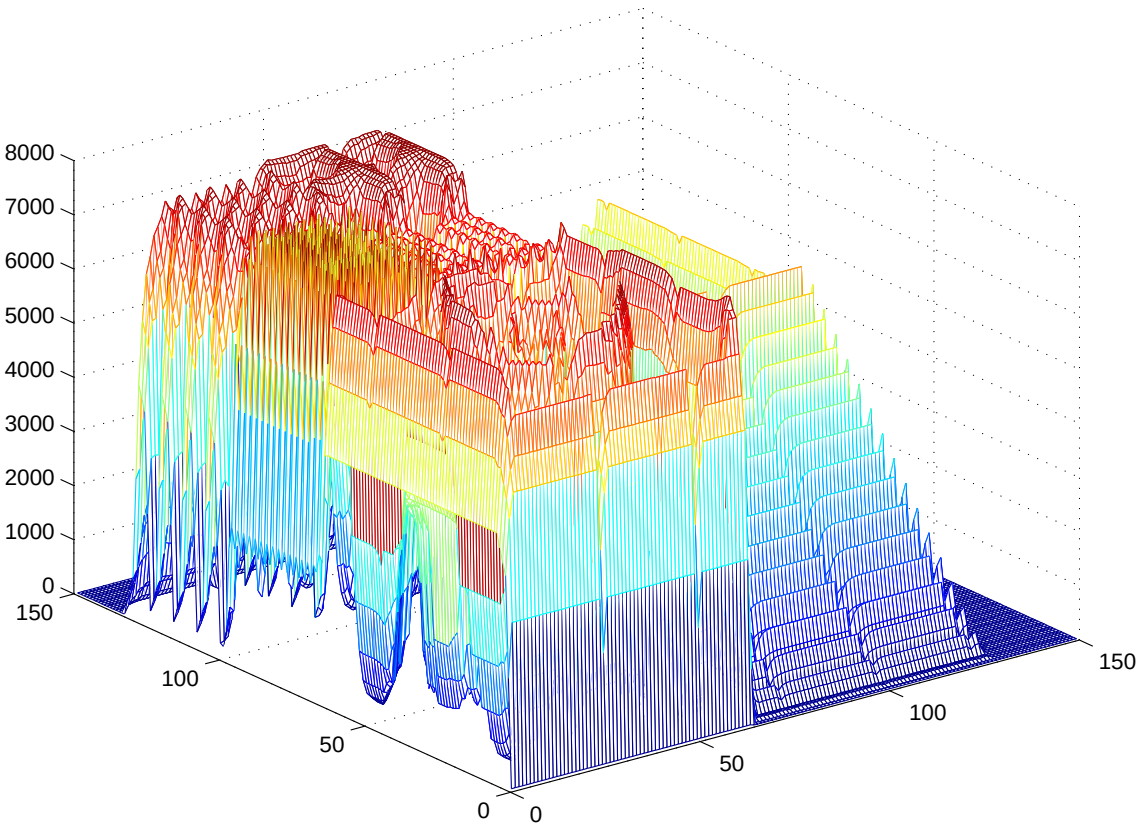

Figure 7.15: 2D-FIR Filtered Image using Four-section Computational Primitive processing

The reuse of the same architecture concept can be implemented in processors with different filter lengths. Note that there are only two right shifters that adjust (scale) the operational bit width instead of seven shifters (if a one-section computational primitive is implemented). This reduces the round off and overflow errors substantially. The satisfactory output image is shown in figure 7.15. We compared both the one-section and the four-section computational primitive processing with the floating point output image and found that the four-section computational primitive has 7.3dB higher signal to round off noise error than the output image using the one-section computational primitive.

Intercommunication between Row Processors

The data block in the 2D-FIR filter system is a two-dimensional sub-image. Vertical state variables are passed between each row processor. The architecture of how to pass the vertical states variable is critical. According to the experience obtained from the design in the 2D-IIR filter system, it is reasonable to choose similar and simpler processor architecture as well as a controller for the 2D-FIR filter system. We pass the vertical state variables between two row processors, as shown in figure 7.9. The buffer memory along the vertical state variable path is simply a word register, which is simpler and different from the architecture in the 2D-IIR filter system. Since there is an orthogonal characteristic between the two sub-image computations, there is no inter-communication between two block processors. The inter-communication between the row processors happens within the block processor. It is local and the memory size is small. Because the inter-communication is local, the communication bottleneck problem, which happens in most of the VLIW processor designs, has been solved.

Data Input Token Passing Controller (TPC)

The Token Passing Controller is used to assign data blocks through the data bus to the designated processor. The data block is stored in the on board FIFO of a processor and the TPC monitors the states of the FIFO. The processor with the token while its FIFO is empty will send out a request signal to the IM and be entitled to receive one block of data. In the 1D-FIR and 2D-IIR filter system, there is only a one-level hierarchical processor architecture and there is only one FIFO on the processor. Therefore, the TPC is situated at the top level hierarchy of the processor, which is close to the FIFO, and controls the FIFO's write enable signal.

In the 2D-FIR filter system, each block processor has a number of row processors that is equal to the number of rows in a sub-image. Each row processor has a FIFO with a size that is equal to the size of one row of a sub-image. This is shown in figure 7.9. It is reasonable to keep the TPC close to the on board FIFO such as the design in the 2D-IIR filter system, and use one TPC to control one FIFO memory's

assignment. The individual request signal is still fired from the row processor to the associate FIFO (odd or even) in the IM, which contains blocked data. As the data is output from the IM FIFO, the sub-block of data is distributed to the *row processor* FIFO which has the token. This design mechanism, which sets the processor FIFO size equal to the data block size or sub-block size, flexibly accommodates different algorithms.

Data Processing Sequence Control

The system requires the data processing sequence on a processor to be independent from the outside control signal once the onboard processor FIFO is full of one block of data. There is no ambiguity when dealing with the 1D-FIR or 2D-IIR filter systems since they have only one FIFO on each processor. As soon as the FIFO is full with the one input block of data, a start processing signal will be generated and the processor will start its data processing without any interference. Sample data in the FIFO will be retrieved to the computational primitive one at a time. State variable intermediate values will be passed between the correspondent registers or FIFOs. The computed results will be fed back to the same FIFO and output to the OM accordingly.

In the 2D-FIR filter system, each row processor has one FIFO. Therefore, the number of FIFOs on each block processor is the same as that of rows in a sub-image block. Vertical state variable values are passed from one row processor to another. The block processor local data processing control can be either traditionally synchronous or asynchronous control, with the latter similar to the global asynchronous control mechanism. The advantage of using the asynchronous controller is that the row processor can start its processing whenever it obtains its sub-block of data. The disadvantage is, by adding an extra controller and memory, bigger chip area and more power consumption will be required. This is especially true when the extra memory is a FIFO.

Since the sub-block size of a sub-image is small (i.e., 32 in figure 7.9), it is more efficient to use a synchronous controller to process the small size data locally. The vertical state variables are passed from one row processor to another, and are stored

temporarily in one-word registers. The block processor still gets its one data block which is divided and distributed to the FIFO on the row processors. This means that the block processor will start its data processing, once all the row processor FIFOs on the block processor are full. A start processing signal is fired from the block processor and is disseminated to all the row processors. All the row processors will start processing their sub-block data simultaneously.

Data Output Token Passing Controller

In the 1D-FIR filter system, the output data blocks from the PMA are guaranteed not to overlap when they arrive at the OM. In the 2D-FIR filter system, there are multiple FIFOs, which contain processed sub-block data and they are ready to output from the block processor at the same time. Therefore, a data output token passing controller is needed at the output side of the block processor to output the processed sub-blocks of data.

The mechanism and design of the data output token passing controller is the same as the data input token passing controller; only the sequence is reversed. It monitors the status of the sub-block FIFO. If the output token is available on a row processor and its FIFO is full with its output sub-block of data, then the system will allow the FIFO to output the data to the OM. When the FIFO is emptied, the output token will be passed to the adjacent row processor or to the row processor of an adjacent block processor as appropriate. The hand shaking between the data output token passing controller is the same as the data input token passing controller. The only difference in the circuitry is that the data output token passing controller does not have a request signal.

Overlapped Data Elimination

The overlapped areas of the 2D-FIR filter system are shown in figure 7.5 and figure 7.4, which are different from the 1D-FIR filter system (figure 7.1). In the 2D-FIR filter system, the original image frame can be recovered by eliminating the top filter length's rows and the left side filter length's columns for each sub-image block, as is

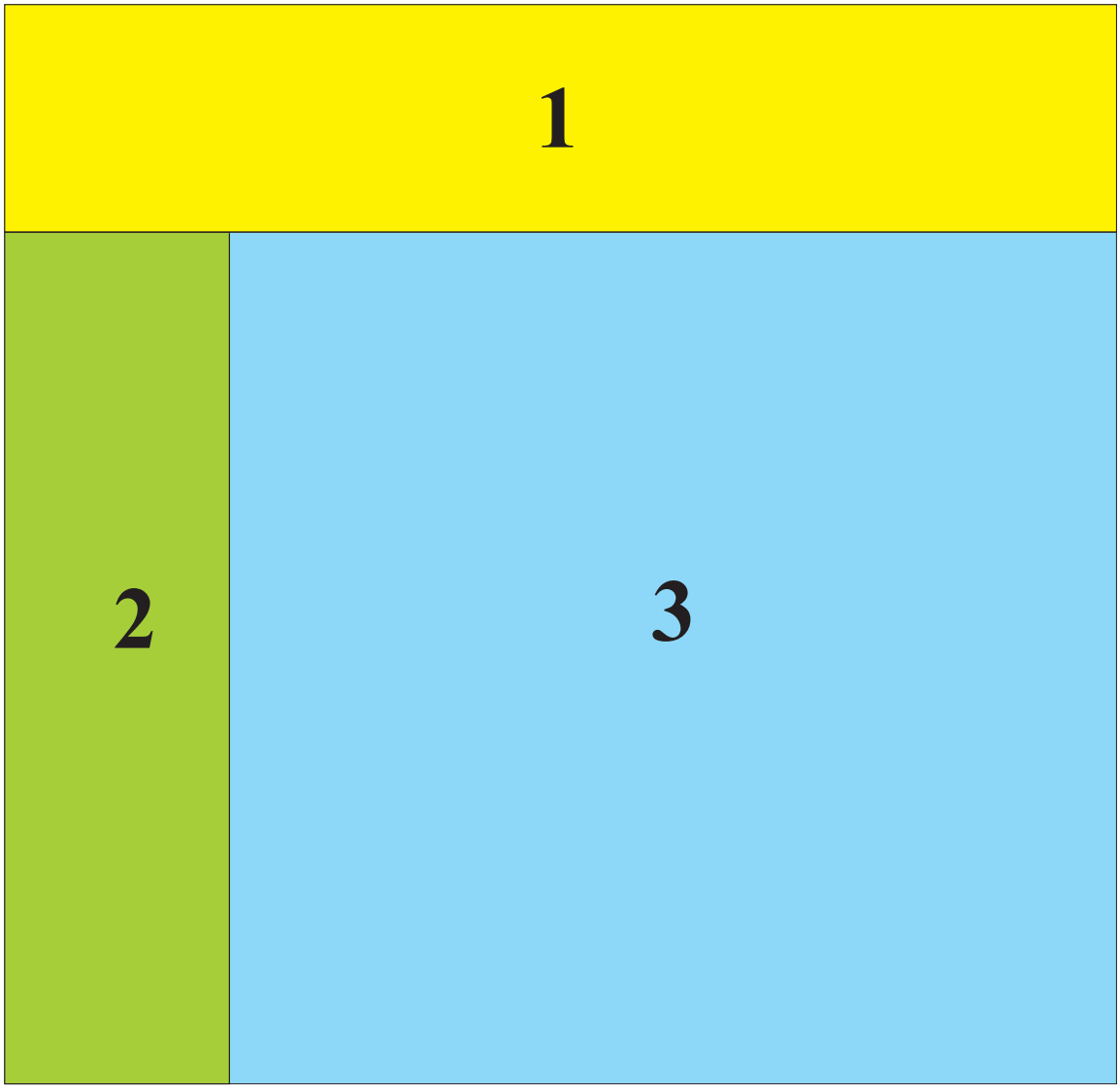

Figure 7.16: Sub-image Data Block

shown in figure 7.16 area '1' and area '2'.

The data blocks of area '1' in figure 7.16 are processed by the beginning several row processors of a block processor. There is no intercommunication between block processors in the 2D-FIR filter system. The processor for the first row of area 3 needs to receive vertical state variables from the processor for the last row in area 1. However, the output from the processors in area 1 is not needed. The output data of the overlapped row processors can be suppressed by an external signal when the 2D-FIR filter length is decided. The output data block is controlled by the output token controller. The overlapped row processors must be taken out from the output token control ring.

The configuration of the rest of the area, which includes area '2' and '3' in figure 7.16 is the same as in the 1D-FIR filter system. Every row processor, which processes one row of data, will eliminate the beginning filter length's output samples of a row of data. Every 1D-FIR filter processor does the same process when its data is output. So, we keep the 1D-FIR filter processor architecture for this purpose.

7.3.3 2D-IIR Filter

Token Passing Control

Data blocks, which are the same size, are transferred from the IM into the PMA's odd or even group of processor arrays. The data block transferring control is asynchronous, and is the same token passing control ring architecture that was used in the previous two FIR filter systems. The 2D-IIR filter processor is a one-level hierarchical processor. So, the token control module is located on every processor module.

Formation of Computational Primitive

There are nine different state space equations in equations 6.9, 6.10, and 6.11. One method is to use nine computational primitives to do the nine equation computations, which cannot be operated in parallel due to the state variable dependency. Another way is to use only one computational primitive to do all nine equation computations. The other way is to check the data dependency relationship and use more

than two computational primitives to do the processing. We choose to use only one computational primitive to do all the processing with a simpler controller.

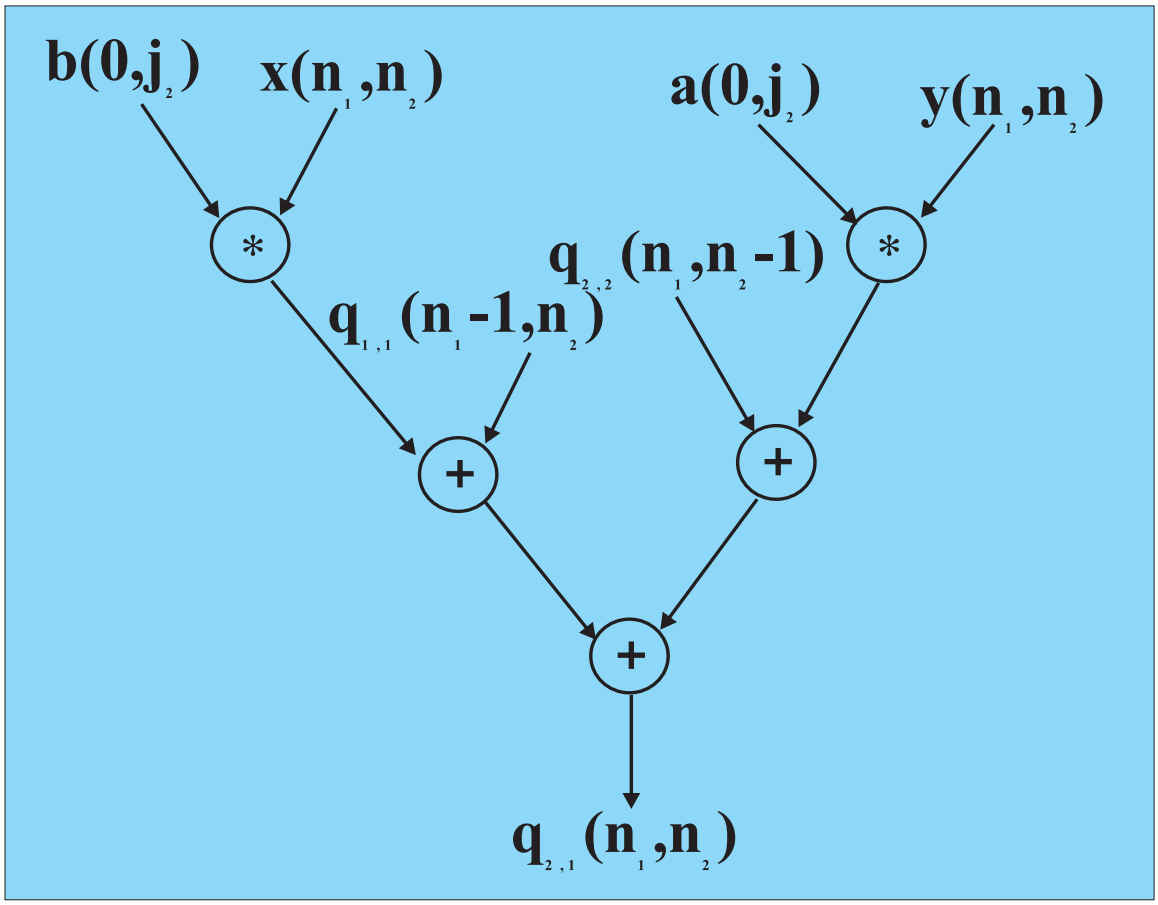

Figure 7.17: 2D-IIR Computational Primitive

When examining equations 6.9, 6.10, and 6.11, the greatest lower bound on computational complexity is given by the vertical state variable equation $q(2,1)$ which requires two multiplications and three additions (or subtractions). Therefore, the computational primitive shown in figure 7.17 is a candidate structure for the primitive processor. Now, it is a fact that not every equation has the same number of variables. Equations $q_{2,1}(n1,n2)$ has six variables and equation $q_{1,2}(n1,n2)$, $q_{1,4}(n1,n2)$, $q_{1,6}(n1,n2)$, each have only four variables. In order to use the same processor structure, zeros are needed as a substitute for the absent variables.

The analysis is based on one computational primitive computing all nine equa-

tions. These nine equations consist of six horizontal state variables, two vertical state variables, and one result value. Therefore, the sequence of computing the nine equations is very important. The horizontal state variable will be recycled only in its own processor, while the vertical state variable will be passed to the next stage processor. It is wise to consider passing the horizontal state variables as well as the vertical state variables evenly. In other words, the next processor is expected to start its processing as soon as the vertical state variable from the previous stage processor is ready.

Equation 6.11 is the first equation to be calculated because equation 6.9 and 6.10 need the results of equation 6.11 in their computation. The result of equation 6.11 is stored in a temporary register and the register is accessed by the other state variable equations. The value in this register will be updated whenever new pixel data appear in the computation cycle. Notice that there are six horizontal state variable equations and two vertical state variable equations. In order to drive the in-processor computation and the next processor computation evenly, two vertical state variable equations are inserted evenly along with the horizontal state variable equations. The sequence of the state variable equation is arranged according to the sequence when the state variable is needed. For example, the horizontal state variable $q_{1,1}$ is needed in the first computational equation. Consequently, it earns the first place to be computed in the sequence of the horizontal state variable equations. The calculated result will be put into a temporary register ready for the next pixel's loop. The same situation happens with the vertical state variable as well. The reasoning above is concluded in the "Computational Table" as shown in table 7.1.

In table 7.1, 'ck' represents the clock cycle number; 'b_in' is the b filter coefficient generated from Matlab software; 'i/p' represents the input pixel data; 'a_in' is the a filter coefficient generated from Matlab; 'o/p' is the computational result; 'H_reg' is the horizontal state variable generated from the previous input state; and 'V_reg' is the vertical state variable generated from the previous processor (or previous row). The last column is subdivided into four columns, which represent feedback temporary registers, the output data register, the horizontal state variable register, and the vertical state variable register.

ck	b_in	i/p	a_in	o/p	H_reg	V_reg	out_reg			
0	$b(0,0)$	$f(n1,n2)$	0	0	$q1,1$	$q2,1$	$g\_out = out\_reg$	$out = out\_reg$	$H = q1,2$	$V = 0$
1	$b(1,0)$	$f(n1,n2)$	$-a(1,0)$	g_out	$q1,2$	0	$q1,1 = out\_reg$	$out = g\_out$	0	$V = 0$
2	$b(2,0)$	$f(n1,n2)$	$-a(2,0)$	g_out	0	0	$q1,2 = out\_reg$	$out = g\_out$	$q1,3$	$V = q2,2$
3	$b(0,1)$	$f(n1,n2)$	$-a(0,1)$	g_out	$q1,3$	$q2,2$	$q2,1 = out\_reg$	$out = g\_out$	$q1,4$	0
4	$b(1,1)$	$f(n1,n2)$	$-a(1,1)$	g_out	$q1,4$	0	$q1,3 = out\_reg$	$out = g\_out$	0	0
5	$b(2,1)$	$f(n1,n2)$	$-a(2,1)$	g_out	0	0	$q1,4 = out\_reg$	$out = g\_out$	$q1,5$	0
6	$b(0,2)$	$f(n1,n2)$	$-a(0,2)$	g_out	$q1,5$	0	$q2,2 = out\_reg$	$out = g\_out$	$q1,6$	0
7	$b(1,2)$	$f(n1,n2)$	$-a(1,2)$	g_out	$q1,6$	0	$q1,5 = out\_reg$	$out = g\_out$	0	0
8	$b(2,2)$	$f(n1,n2)$	$-a(2,2)$	g_out	0	0	$q1,6 = out\_reg$	$out = 0$	$q1,1$	$V = q2,1$

Table 7.1: 2D-IIR State Table for State Variable Equations

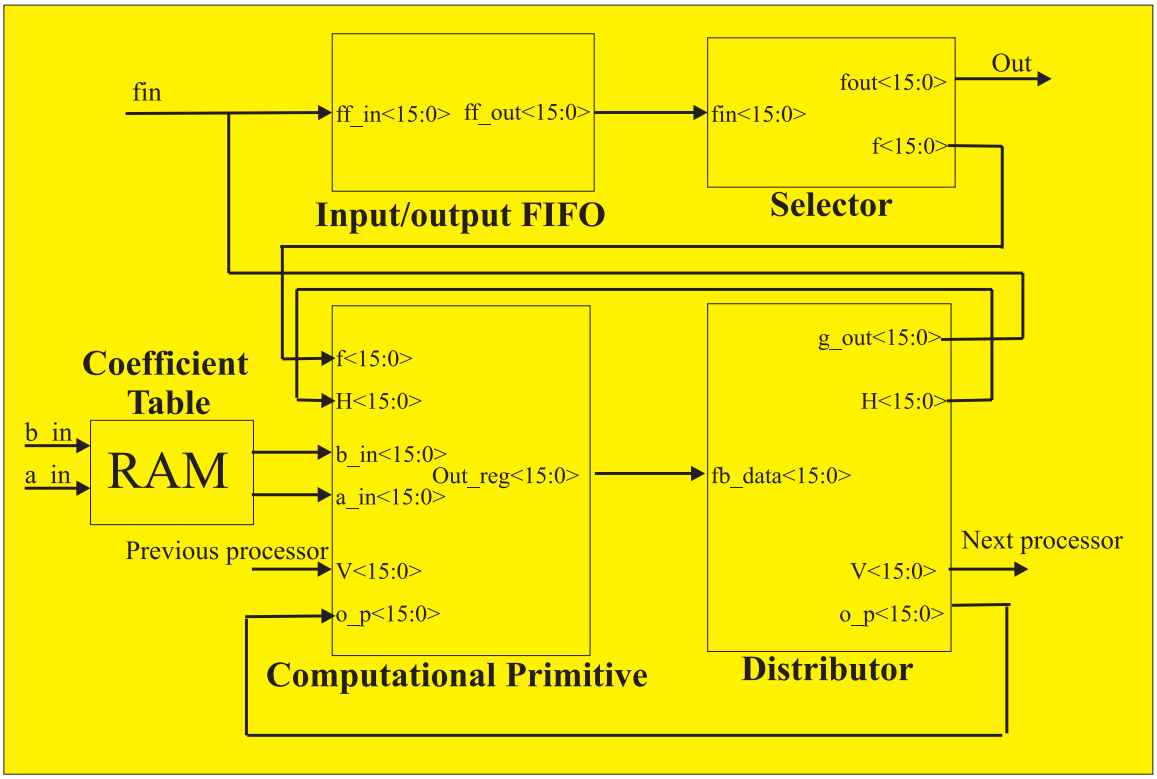

Figure 7.18: 2-D IIR Processor Block Diagram

Computational Primitive Feedback Distributor Architecture

Figure 7.18 is the block diagram of one processor. The Computational Primitive (CP) takes nine clock cycles to process one pixel of the input image. The output of the computational primitive is a one string word on a data bus. This bus consists of nine different computed results, which are then distributed into different registers or a FIFO at different clock cycles.

The horizontal state variable registers and the vertical state variable registers are registers that hold the computed state variables temporarily. These registers prepare data for the next state's request. The choice of using different registers to hold different output variables is to have clear, separated data and simple control. Another alternative design is not distributing the output variable into different registers, but feeding it into only a universal memory, as it is done in most of the Instruction Level Parallelism (ILP) processors. In doing so, there should be a controller, or a program counter. This program counter manages the data path among output data, the feedback horizontal state variable, and the vertical state variable. The universal memory design will increase the control complexity and communication latency.

FIFO Data Input/Output Control

The input pixel is input from the IM in blocks according to the request of the processor. Originally, this data block is stored in the Input/Output FIFO. One pixel is read into the computational primitive every 9-clock cycles (9 equations). After computing the pixel value, the value will be fed back into the Input/Output FIFO. This read and feed period is determined by the processing time of each pixel in the processor primitive. If a few more clock cycles are added for processing one pixel, then the read-feedback period of a pixel will be prolonged.

In order to reduce the size of the memory, the same FIFO is used to store the input block data and the computed output results. The selector at the output section will select the correct data to feedback into the Input/Output FIFO or to the OM FIFO.

Coefficient Table

The previously developed [8] suppression method expects to reduce the transient effect caused by unspecified initial conditions or truncation of data at the boundary area. This method divides the input image into four section areas. Each section has its own unique set of filtering coefficients and state table.

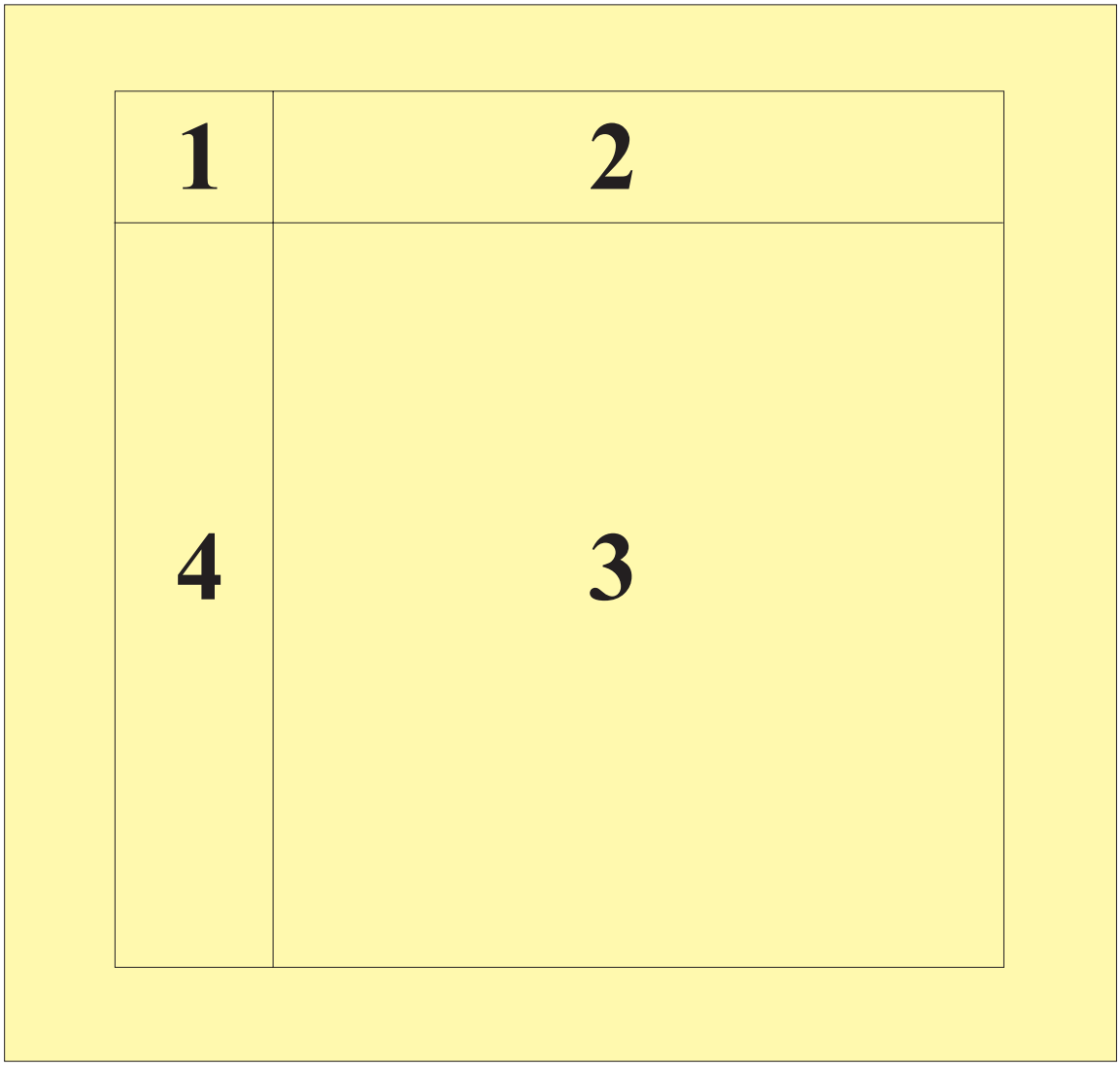

Figure 7.19: Coefficient Allocation Table

Coefficients a and b are not always in one set due to the boundary condition of the image. For example in figure 7.19, area 1 (first pixel of each frame of image),

there are no vertical state variables or horizontal state variables provided from the previous pixel. Therefore, it uses one set of coefficients by itself. In area 2, there are no vertical state variables provided from the upper row of pixels. This is another set of coefficients. Area 3 has its own set of coefficients. Area 4 has another set of coefficients and there are no previous horizontal state variables for area 4. Therefore, there are four different sets of a and b coefficients and state tables. The total number of coefficients for each a and b in the second order 2-D IIR filter is 36. A dual port RAM is designed to store all 72 coefficients and the values in this dual port RAM are retrieved with different counters providing different sets of coefficients to the different blocks of data. Generally speaking, the mechanism of allocating the coefficients a and b is the same with each processor. Only the first processor (first data block processing) uses a different set of coefficients. The first processor uses area 1 coefficients a and b, and area 2 coefficients a and b. The rest of the processors in the processor module array (PMA) are using the same (normal) sets of coefficients (area 3 and 4 in figure 7.19). Note that the processors in the PMA are arranged in a ring type architecture. The coefficient set in the first processor must be changed into the normal coefficient set right after it has processed the first block of data. Therefore, the first processor in a frame will need to change coefficients set when it has been assigned a row that is not the first one in a frame. In addition, another processor may be assigned the first frame in a multi-frame application.

Vertical State Variable Passing Control

The vertical state variables are stored in the FIFO between the two processors by the previous processor. The successor processor will have to wait until the proper vertical state variable is ready, then continue its processing. There is a handshaking protocol between the processors.

Normal Processors The normal processors will start their handshaking with the previous processor right after they receive their own block. The request for a vertical state variable will be raised first. If the state variable is ready in the previous vertical state variable FIFO, an acknowledging signal as well as the vertical state variable will

be sent to the requesting processor and terminate the request. The processor will again send a request to the previous processor before another vertical state variable is needed. The controller monitors the FIFO states and make sure the state variable is available. When the FIFO is to be written to and read from at the same time, the FIFO write always has the priority.

Abnormal Processor In an image frame, the data processing for the first data block (first processor initial condition) is different from the normal data block processing. In figure 7.19, the first data block has no vertical state variable passing from the upper row. Also there are no horizontal state variables passing from the left hand side to the first pixel data. According to [8], the transient is suppressed from the upper row by not allowing a change in the state as the system crosses this boundary. Therefore, two different state tables (similar to table 7.1), which are associated with area 1 and area 2 with different sets of coefficients a and b must be created. Whenever the vertical state variables or horizontal state variables are absent, a zero must be substituted. The same computational primitive architecture is used in all four areas. The processor array is looped in a ring type architecture. There is no handshaking for vertical state variables between the first processor and the previous processor at the initial first block data processing period.

Vertical State Variable FIFO Size As described in the previous paragraph, at the initial phase, when the first block of data is processed in the first processor, there is no handshaking between the first and the previous processor. In other words, there are no vertical state variables to be passed from the previous processor to the first processor at the initial phase. Therefore at the initial phase, the vertical state variables generated from the last processor, which is supposed to input into the first processor, have to be stored in the vertical state variable FIFO. Since FIFO memory occupies lots of wafer area, it is important to determine the proper size of this FIFO.

The previous processor vertical state variable FIFO size depends upon how fast the current processor retrieves the vertical state variables from it. The current processor can retrieve vertical state variables from the FIFO once it obtains its data block.

The time difference from when the vertical state variables are ready in the previous processor FIFO to when the current processor is able to retrieve the vertical state variables from it is affected by the processing speed of the processor. This time difference varies when the number of processors varies. Therefore, it is proper to choose the upper bound for this FIFO size. Setting the vertical state variable FIFO size equals to the data block size is a conservative approach.

Hierarchical Independent Control

To divide a central control problem into a hierarchical control problem, it is possible to divide and conquer a very complicated system control problem into separated small local control problems. Our approach is to divide the control in a hierarchical way by separating the vertical state variable passing control from IM-PMA control and PMA-OM control. The benefit of this hierarchical control design is simple, clear cut, and it will be easy to integrate it later with a higher level hierarchical control. In addition, a modular hierarchical control is more flexible and it makes it more feasible to accommodate different mathematical algorithms.

7.4 Output Module Architecture

The input data block is distributed to the designated processor sequentially according to the token passing protocol. The output data block from the PMA is output to the OM according to the token passing protocol as well. Therefore, the OM can reconstruct the processed output data by combining the blocks of data as appropriate.

Chapter 8

Scalability and Flexibility

8.1 1D-FIR Filter

The main purpose of our research is to adapt different but similar algorithms into the same hardware without changing too many components or parameters. The flexible configuration presented in our research is especially useful in a scenario as follows. A robot is located at a remote area (or a hazardous area), and there is a need for the robot to change its function in real time when a certain situation happens. The partially reconfigurable speed for the FPGA is less than 50 ms. Therefore, with the flexible algorithm mapping design in our research, we believe the reconfigurable time from one algorithm to another, or from one specification to another, is near real time. The robot scenario requirement is met with our design approach.

We use variable counters to implement different filter system designs in our research and to adapt to different block sizes and different filter lengths for each filter system. A special selector is designed in the IM to choose either broadcasting or alternatively distributing the incoming data block into two different FIFOs. The incoming data of the 2D-FIR and the 2D-IIR filter system uses the alternating distribution selection and the 1D-FIR filter system uses the broadcasting one. All three algorithms use the same IM hardware.

The processor core of the 1D-FIR filter is different from that for the 2D-IIR filter, because intermediate data is passed between the 2D-IIR filter processors. The last

processor in the 2D-IIR PMA is connected to the first processor, and the control part of these two processors are different. The 1D-FIR filter is different from the 2D-FIR filter in the pre-image processing and part of the processor core. All these hardware design variations/differences between algorithms are limited to the minimum in our research.

The processors in the PMA for the same algorithm are all the same. Therefore, it is easy to scale up the system processing speed by cloning the same processor. Fault tolerance is considered by changing the connecting point between the processors and bypassing a faulty processor.

We design the system not only be able to accommodate different algorithms but also to be able to change parameters within a specific algorithm. Within the same algorithm, the data processing block size, the filter length, the number of processors, and the filter coefficients can all be changed. The actual configuration of an application depends on the system requirements such as power consumption, filter frequency response, throughput performance, processor utilization and efficiency. Since the scalable/flexible parts are designed in variable counters, FIFO modules, or a configurable processor core, it is reasonable to use configurable components such as a FPGA to implement those scalable/flexible parts and use an ASIC to implement the fixed parts.

Table 8.1 shows how easily the 1D-FIR filter system specification/configuration can be changed by modifying the associated parameters. The first row in table 8.1 shows the variable specifications in the 1D-FIR filter system. The second row shows where the variable parameters are located, and the first column shows all the variable parameter modules. For example, if the number of processors needs to be changed due to a different throughput performance requirement, the last column check marks in table 8.1 indicate that the token line, the acknowledge line, the processor number, the request line, and the input/output ports, are all to be modified. These modifications are located in the PMA.

Specification	Block size		Filter length		Filter coeff.		PE number	
Module location	IM	PMA	IM	PMA	IM	PMA	IM	PMA
Flip-flop counters	√		√					
Block counters	√							
Forward counter	√							
Backward counters	√							
Token line								√
Acknowledge line								√
Processor								√
Request								√
Input ports								√
Output ports								√
Cycle counter				√				
FIFO size		√						
Read write counters		√						
Overlap counter				√				
Register controller				√				
Coefficient controller				√				
Coefficient memory				√		√		

Table 8.1: 1D-FIR Filter System Variable Parameters

8.1.1 Scalability

The scalability of the 1D-FIR filter system depends on whether we can add more processors in the system in order to obtain better throughput performance. Since we use the overlap-save algorithm to implement our 1D-FIR filter, there is an orthogonal characteristic between the two processing blocks. That means there is no dependency relation between two computing blocks. Therefore, there is no data passing between two processors and the processors operate independently and concurrently. If the IM provides the PMA sufficient data blocks while the PMA keeps increasing the number of processors and the data block size, the system throughput performance will be increased until there exists a data flow bottleneck.

Variation of Processor Number

The processors in the PMA are all identical. Therefore, adding or reducing the number of processors in the PMA is just as simple as inserting or deleting processor elements into the PMA and then connecting the necessary ports and control lines.

8.1.2 Flexibility

The flexibility of the 1D-FIR filter system includes the changing of the block size, the filter length, and the value of filter coefficients. The purpose for changing the block size and filter length is to obtain better performance, better processor utilization, and smoother output data. Table 8.1 shows the parameters that need to be changed once its corresponding specification in the system is changed.

Variation of Block Size

The second and third columns display the variable parameters corresponding to the system block size change. All variable parameters are different counters except the PMA FIFO, which is exactly equal to the block size. The first counter in the IM Flip-Flop counter specifies the length of the filter that is used in the system [7], and the sum of these two counters is equal to the block size. If the system filter length is not changing and only the block size is, then the second Flip-Flop counter needs to be modified to match the block size. The two block counters in the IM, with each counting the block size and recording the output data number for the individual IM FIFO, will be changed accordingly. The forward counter in the IM is used to ensure that one block of data is sent to the PMA after the FIFO full situation occurs. The write counter and the read counter on each processor record the precise data number, which is input into or output from the processor FIFO. Therefore, to change the input processing block size for the 1D-FIR filter system is to change all the corresponding counters and the FIFO size in the PMA. Another remedy method for changing the block size without changing the processor FIFO size is to use the write counter, which records the actual FIFO input data number, to indicate a full block data is input into

the FIFO and use this signal to trigger the processor to start its processing. In this way, we can chose a bigger FIFO size rather than modify the processor FIFO size each time we change the block size.

Variation of Filter Length

Changing the system filter length is more complicated than changing the system block size because it includes not only modifying different variable counters but also modifying the controllers in the PMA. The first component that needs to be modified is the Flip-Flop counter in the IM. The first counter of the Flip-Flop counter will be changed according to the filter length that is used in the system. The second counter of the Flip-Flop counter, which counts to the remaining of the block size minus the filter length, needs to be modified so that the sum of these two counters equals the block size. The other components which are all located in the PMA that need to be modified are the cycle counter, the overlap counter, the register controller, the coefficient controller, and the contexts in the coefficient memory.

Changing the filter length will cause the changing of the number of coefficients. Therefore, the first modification in the PMA is the coefficient controller and the contents of the coefficient memory. Secondarily, according to figure 6.1, the total number of one-section computational primitives is changed when the filter length is changed. If we want to keep the original processor architecture for the sake of minimizing modifications, not increasing any number of computational primitives in the PMA, then we have to modify the register controller, which controls the data path according to the different computational primitives and stores the temporary state variables in its registers. Since we decided not to change the computational primitive architecture in the PMA and modify only the register controller, the number of cycles that the register controller uses will be changed. Therefore, the PMA cycle counter needs to be changed accordingly.

The purpose of the overlap counter is to eliminate the overlapped area between each block of data so that the original signal can be reconstructed. The overlapped area is equal to the length of the filter. Therefore, the overlap counter in the PMA is

to be modified when the filter length is changed.

Variation of Filter Coefficient

In order to change the frequency response of the filter, the coefficient values for the filter must be changed. This means by changing the contents in the coefficient memory, the filter frequency response will be changed.

It is clear that, according to the analysis of table 8.1, the design of the 1D-FIR filter system BDPA is not only flexible but also scalable. The design has application specific characteristics and is a high performance computational device. Moreover, its specifications can be changed easily by changing the variable parameters.

8.2 2D-FIR Filter

Although the BDPA we designed is not as flexible as the VLIW technique that is used in most of the multi-media processors, it is much more flexible than the ASIC. The goal for the BDPA is to investigate application specific computing architectures so that a block (cluster) of data can find a data path that flows through a concurrent processor locally and obtain high performance computing for different algorithms. These micro-architectures are designed as flexible/scalable as possible and are formed as special modules (intellectual property IP) in the library. On the other hand, since the architecture is scalable/flexible, the end user designer can form their customized application specific computing system with different system throughput performance, wafer area, power consumption, according to the resource, specification, and requirement that the end user system demands.

8.2.1 Scalability

Variation of Processor Number

The 2D-FIR filter system variable parameters are listed in table 8.2, which is similar to table 8.1. The processor architecture in the 2D-FIR filter system is a

Specification	Block size		Filter length		Filter coeff.		PE number	
Module location	IM	PMA	IM	PMA	IM	PMA	IM	PMA
Flip-Flop counters	√		⋆					
Block counters	√							
Forward counter	√							
Backward counters	√							
Token line		◯						√
Acknowledge line		◯						√
Out token line		◯		√ √				√ √
Out acknowledge line		◯		√ √				√ √
Output control				√ √				
Block processor								√ √
Row processor		◯		√ √				
Request		◯						√
Input ports		◯						√
Output ports		◯						√
Vertical state variable signal		◯		√ √				
Cycle counter				√				
FIFO size		√						
Read write counters		√						
Overlap counter				√				
Register controller				√				
Coefficient controller				√				
Coefficient memory				√		√		

Table 8.2: 2D-FIR Filter System Variable Parameters

hierarchical processor architecture, which is different from the 1D-FIR filter processor architecture. The asynchronous block data controller in the 2D-FIR filter system is also different from the 1D-FIR filter system. There is only one input block data token controller ring that is necessary in the PMA in the 1D-FIR filter system. In the 2D-FIR filter system, it is necessary to have another output token controller ring in the PMA. Therefore, the double check marks in the last column in table 8.2 show the difference between the 1D-FIR filter system and the 2D-FIR filter system when modifying the system processor number. Notice that the row processor number on the block processor does not change while changing the number of the block processors in the PMA in the 2D-FIR filter system. The rest of the modifications for changing the

number of block processors in the 2D-FIR filter system are the same as for the 1D-FIR filter system. The 2D-FIR filter system is easily scaled up/down by adding/reducing block processors in the PMA, modifying the input/output token control lines, signal requesting line, and I/O ports.

8.2.2 Flexibility

Variation of Block Size

The algorithm used in the BDPA FIR filter systems is the overlap-save algorithm and the data blocks in the 2D-FIR filter system are partitioned in a way that each block contains two dimensional sub-image. So, changing the data block size in the 2D-FIR filter system could mean changing only one dimension or changing both dimensions. As it is shown in figure 7.9, the block data (sub-image) is input into the block processor in a way that each row-data of a sub-image is input into a row processor, and the number of row processors on a block processor is equal to the number of rows of a sub-image.

If the data block (sub-image) size variation only happens in the horizontal direction, the system modification is the same as in the 1D-FIR filter system, which is to change the FIFO size and different counters in the IM and PMA. This is a much simpler method to modify the block size in the 2D-FIR filter system.

If the sub-image size variation happens in the vertical direction, then, as shown in table 8.2 third column, all the circled parts need to be modified. That includes row processor number modification, token (input and output) controller modification, block data requesting signal modification, vertical state variable passing line modification, and input/output ports modification.

Variation of Filter Length and Coefficient

An obvious difference of changing the filter length between the 1D-FIR and the 2D-FIR filter system is the modification of the IM. It is not necessary to modify the IM if changing the filter length in the 2D-FIR filter system. The image pre-processing

module partitions the input image frame into overlapped sub-images with the same data block size. Therefore the Flip-Flop counter in the IM always counts the same number and distributes the same size of data block into PMA block processors.

If the filter length in the 2D-FIR filter system is changed, the overlapped area '1' and '2' in figure 7.16 will be changed. This is to say the configuration of row processor, output control, out token ring, vertical state variable signal line, and the output overlap counter need to be modified.

Theoretically speaking, according to the signal flow graph in figure 6.2, changing the filter length will lead to changing the configuration of the computational primitive, which complicates the modification of the 2D-FIR filter system. In order to keep the computing data in the dynamic range of 16-bit arithmetic operations and fix the configuration of the computational primitive, we still use the 4-section computational primitive configuration by changing only the register controller and the coefficient controller. Although the modification of the controllers will set different data/coefficient path to the computational primitive, the complicated reconfiguration of the computational primitive is avoided. The cycle counter is to be changed accordingly and the contents of the coefficient memory is to be modified as well.

8.3 2D-IIR Filter

The 2D-IIR filter system also has the capability of changing the number of processors, the block size, the filter length, and the value of filter coefficients. Table 8.3 shows the variable parameters for the 2D-IIR filter system. The obvious differences between modifying the IIR and the FIR filter system, which make the IIR filter system scalable, is the extra vertical state variable FIFOs and the associated controller. According to the BDPA designed in our research, the 2D-IIR filter system has to pass vertical state variables between the processors while the 2D-FIR filter system does not. The vertical state variable FIFOs store the intermediate vertical state variables, which are passed from the previous row pixel computation to the next row pixel computation during the image processing.

Specification	Block size		Filter length		Filter coeff.		PE number	
Module location	IM	PMA	IM	PMA	IM	PMA	IM	PMA
Flip-Flop counters	√							
Block counters	√							
Forward counter	√							
Backward counters	√							
Token line								√
Acknowledge line								√
Out token line								√
Out acknowledge line								√
Processor				√				√
Request								√
Input ports								√
Output ports								√
Vertical state variable lines				√				⋆
Cycle counter				√				
Vertical FIFO				⋆				
Vertical FIFO size		⋆						
FIFO size		√						
Read write counters		√						
Register controller				√				
Coefficient controller				√				
Coefficient memory				√		√		

Table 8.3: 2D-IIR Filter System Variable Parameters

8.3.1 Scalability

The procedure for adding/reducing the number of processors in the 2D-IIR filter system is similar to that for the 2D-FIR filter system. After adding/reducing the processors, it is necessary to add/eliminate the input/output control lines, that is, the input/output token and acknowledge lines, the input/output ports, and the signal request lines.

The processors in the 2D-FIR filter system use a two-level hierarchical processor architecture. The row processors are included in each individual block processor. The vertical state variable registers, which are to pass the vertical state variable between the row pixels within a data block, are designed within the block processor. Therefore,

the passing of vertical state variable for a 2D-FIR filter system is inside the block processor. There is no need to modify the vertical state variable passing lines or the control lines for passing vertical state variables between processors.

The processor in the 2D-IIR filter system is one-level hierarchical architecture and the vertical state variable passing happens between two processors. Therefore, after adding/reducing the number of processors in the 2D-IIR filter system, it is necessary to modify the vertical state variable passing lines and the control lines.

8.3.2 Flexibility

Block Size Variation

In the 2D-IIR filter system, all the data blocks are the same size and every block contains one row of pixels of input image. Therefore, the IM Flip-Flop counters always count the same block size number to control the IM FIFO's input. The IM block counters ensure exactly one block of data to output from the IM FIFO. The forward counter and the backward counters in the regulator monitor one block of data to be output to the PMA or to be input from the data source. All these counters in the IM are to be modified according changes to the block size. The modification is exactly the same as in the 2D-FIR filter system.

The PMA FIFO size, which contains exactly one block of data, and the PMA read/write counters, which monitor the PMA FIFO input/output data number, are to be modified to the size of the data block. The vertical state variable FIFO size in the last processor is to be modified according to the block size changes as well.

Filter Length and Coefficient Variation

If the filter length is equal to two, which is the example described in this paper, there are nine state space equations to be computed in order to obtain one pixel output, and there are two vertical state variables ($q_{2,1}$ and $q_{2,2}$) per input to be passed between processors. If the filter length is increased up to three, there will be 16 state space equations to be computed in order to obtain one pixel output, and there will be three vertical state variables to be passed between processors.

In order to reduce the control complexity, that is, implementing the hierarchical self-sufficient control concept, the 2D-IIR filter BDPA is designed in a way that each vertical state variable intermediate output is assigned to a specific FIFO and has its own set of controllers. The example illustrated in this dissertation, a second order 2D-IIR filter system has two vertical state variable FIFOs in each processor. Therefore, according to the description above, it is necessary to modify the number of vertical state variable FIFO, the cycle counter, and the vertical state variable passing lines when the filter length is changed in the 2D-IIR filter system. Because different filter lengths in the 2D-IIR filter system generates a different state table (i.e., table 7.1), the following must also be modified: the register controller which controls the data path, the coefficient controller which controls the coefficient path, and the computational primitive registers which store the temporary state variables, and the coefficient memory which stores the proper coefficients.

8.4 Interchangeability between Algorithms

Interchangeability between three algorithms using the same FASIC hardware is the purpose of our research. Once the algorithm is changed for the FASIC, the whole function of the device operates differently. For example, a sensor robot in a hazardous area with its limited transmission bandwidth to communicate with its control station can use time sharing method to transmit both processed voice and video signal. The FASIC real-time algorithm interchangeability in our research can provide sufficient resource to achieve this goal.

The 1D-FIR and the 2D-FIR filter system variable parameters are shown in table 8.1 and 8.2. By comparing both tables, it is easily found that the 2D-FIR filter system is more complicated than the 1D-FIR filter system. As mentioned before, the 1D-FIR filter processor architecture is used as part of the 2D-FIR filter processor core. By adding the necessary extra circuits such as the out token line, the out acknowledge line, the output control, the vertical state variable signal line, and modifying the block processor and row processor two level hierarchical processor core, the 1D-FIR filter system can be changed into the 2D-FIR filter system. All these modification

are located at the PMA. The IM and OM module for both algorithms are the same. To change from the 2D-FIR filter back to the 1D-FIR filter is the reverse procedure.

In order to change from the 2D-FIR filter system into the 2D-IIR filter system, we need to compare table 8.2 and table 8.3. The difference in the 2D-FIR filter system architecture is the output control, the block/row processor core, the vertical state variable signal line, and the overlap counter. The difference in the 2D-IIR filter system architecture is the processor core, the vertical state variable line, and the vertical FIFO. All these modification are happening at the PMA as well. As one can see changing algorithms between the FIR and the IIR filter is more complicated than changing algorithms between the 1D/2D FIR filter. The 2D-FIR filter system architecture occupies more wafer resource than the 1D-FIR filter system. However, we still can obtain tremendous throughput performance and speedup by using few block processors and the duel clock rate architecture in the 2D-FIR filter system, which will be discussed in the performance analysis section.

Chapter 9

Design Methodology

The simulation setup in this paper is not cutting edge technology. There are many hardware/software co-verification commercial off the shelf (COTS) products. Xilinx and MathWorks predict that the use of a high level language (i.e., c++ or Java) to design a hardware system will be implemented in 2002.

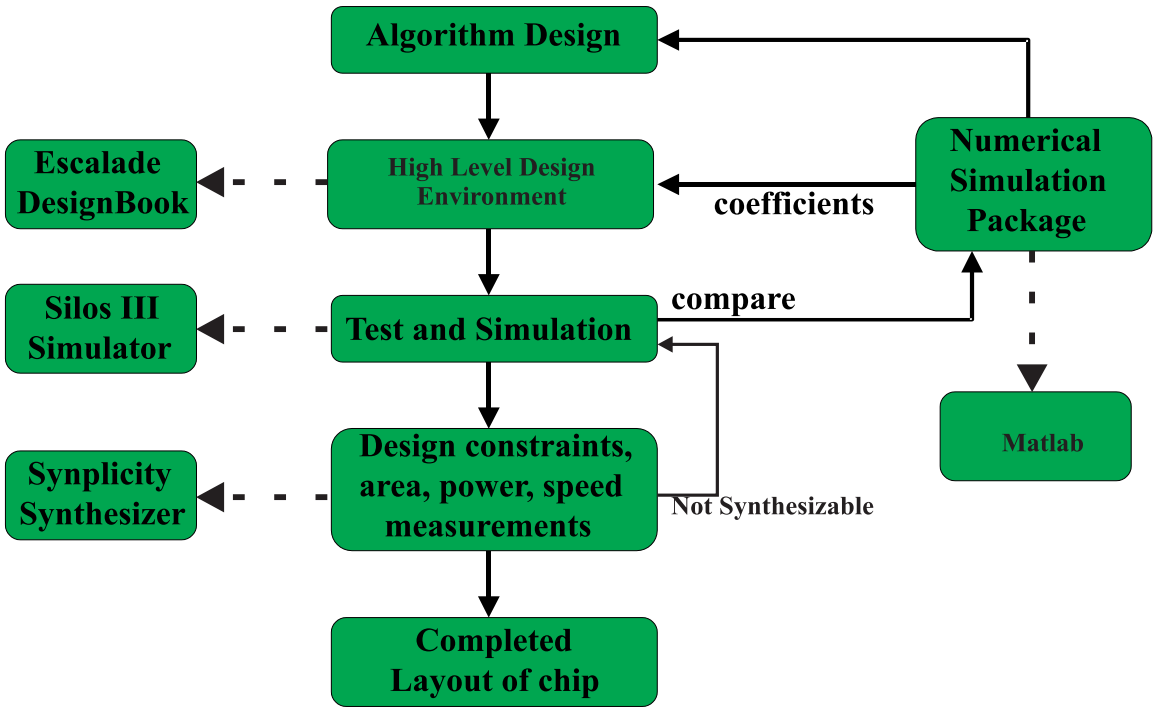

Figure 9.1: Simulation Environment Block Diagram

Although the gap between the digital signal processing system design and hardware prototyping implementation seems to be getting closer and easier, it is still necessary to use traditional/manual methods to design a system in order to obtain a circuit as simple/flexible as it can be during the algorithm mapping/architecture designing stages. The optimized architecture generated from a COTS product is normally not the simplest one due to the constraint of using architectural modules that are available commercially. Our approach, as shown in figure 9.1, begins with simulating the partitioned algorithm with Matlab first. The architecture, as well as the filter coefficients that are generated from Matlab, are then captured by DesignBook/Renoir, which is used to generate the Hardware Description Language (HDL) representation of the system. To make sure the logical design is correct, simulation results from the Silos III simulator are compared with the simulation results from the Matlab simulation. In order to develop the FASIC, iteration of the design between the three algorithms is necessary. Several back and forth design modifications between Synplify, Silos III, and Renoir were typically made in order to obtain a synthesizable system. The PMA architecture designed for the three algorithms in this paper are synthesizable. The netlist generated from Synplify can be further mapped and configured on a FPGA board with the help of the Xilinx tools.

Reviewing the architecture, the data flow, and the flexibility/scalability of the designed system, it is suggested to use a FPGA board to accommodate the PMA module and to use ASIC otherwise. The changing parts in the IM module consists of the Flip-Flop counters, the block counters, the forward counter, and the backward counters. There is only one kind of circuit that needs to be modified. It is easy to offset the counter count number and achieve the IM modification. Therefore, we suggest the IM be fitted in an ASIC circuit.

The processor core architectures for the three different algorithms are different. The PMA modification consists of different kinds of circuits. They are the TPC, the onboard FIFOs, the data/coefficient path controller, and different computational primitive architectures. Therefore, it is reasonable to use a FPGA board to capture the PMA architecture. The FPGA can be dynamically reconfigured and accommodated to these different kinds of circuits in real-time. The data block flow between

the IM, the PMA, and the OM modules are through buses. This data block flow is in large granularity as opposed to a small granularity data computation/memory accessing operation. The system throughput performance will be kept with this mixed ASIC/FPGA design.

Chapter 10

Performance Analysis

10.1 2D-IIR Filter

2D-IIR 64 x 64	Throughput	Utilization
4 processor	0.3554	0.9986

Table 10.1: 2D-IIR Performance Table

Table 10.1 shows the performance of the 2D-IIR filter system using a mono-clock rate. A small image was input into the 2D-IIR filter system with mono-clock rate and the system throughput was 0.3554 pixels per clock cycle. If the system clock speed is 100 Mhz, then the throughput is more than 35 million pixels per second. If the image transmitting rate is to transmit 60 frames per second of a 512 x 512 pixels image, then the transmitting rate is about 16 million pixels per second. Clearly the 2D-IIR filter system with mono-clock rate and four processors is sufficient for this transmitting requirement.

By observing the utilization of the four processors operation, all four of them almost reach their full power. It is predictable that, if we increase the PMA input data flow speed and add more processors to the PMA, we shall obtain a better throughput performance.

10.2 1D-FIR Filter

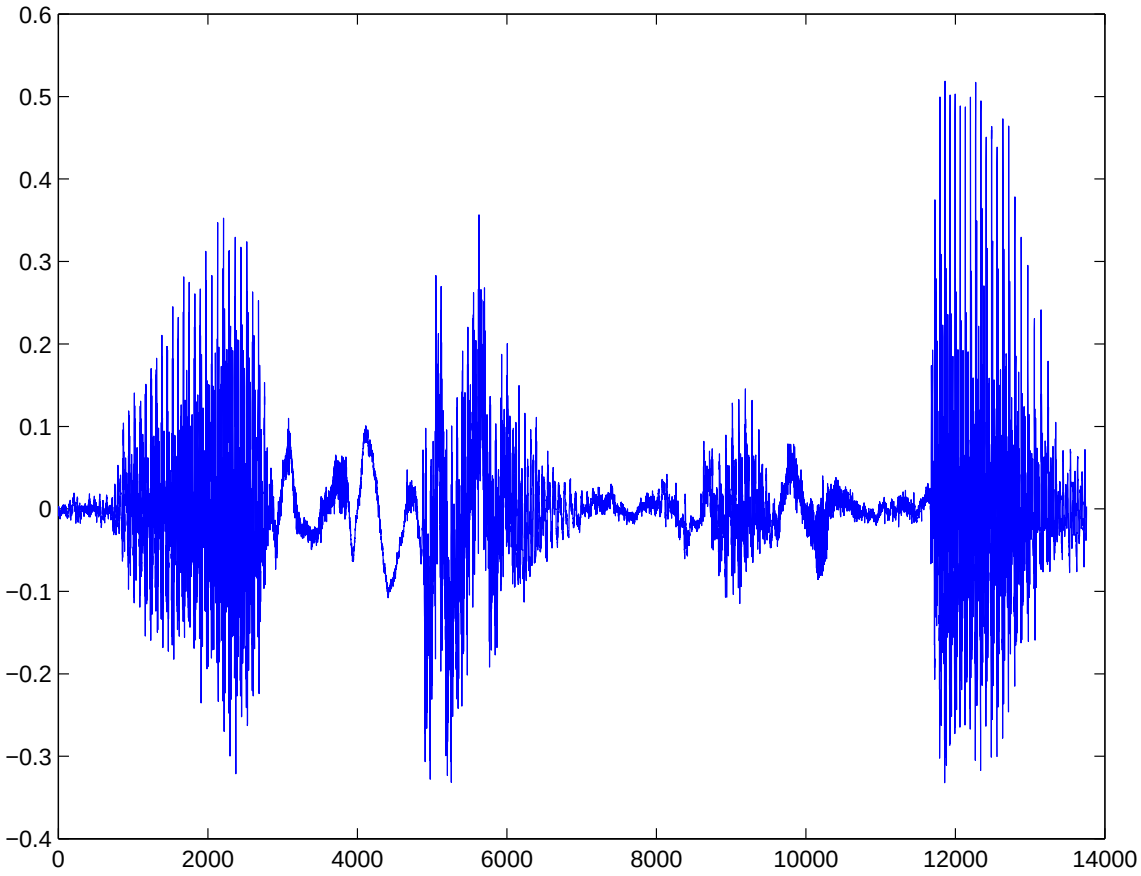

Figure 10.1: 1D Original Signal

10.2.1 Waveform Comparison

We used an $8KHz$ sample rate audio signal (figure 10.1), which was captured by a Sun workstation, combined with a $3.8KHz$ sinusoid data carrier (generated from Matlab) and a white noise for the input data (figure 10.2). This mixed signal represents a received voice signal which has been corrupted by the channel noise. We use this mixed signal as an input signal to the 1D-FIR filter system designed in our research. The system is a mono-clock rate multiprocessor system. The output of the 1D-FIR filter hardware is supposed to recover the original voice signal.

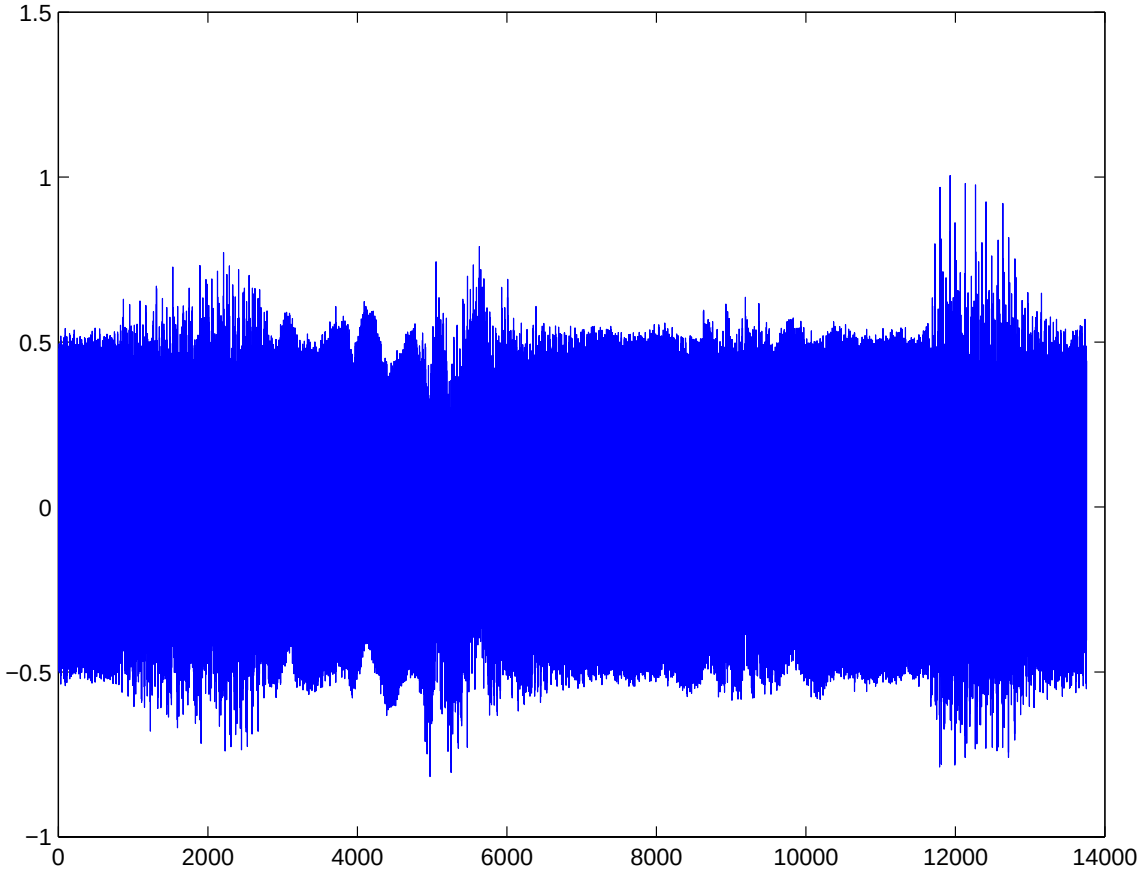

Figure 10.2: 1D Signal with Noise

The mixed signal is scaled to 16 bit signed integers for the hardware architecture simulation. A 63^{rd} order FIR filter was designed for the simulation to filter out the white noise from the mixed signal. The mixed signal was fed into the hardware 1D-FIR filter system and the output waveform was capture and displayed with Matlab. We compared the output result from the hardware implementation with the floating point simulation result from Matlab, which also implemented the overlap-save algorithm. The output from the hardware implementation (figure 10.4) was very close to the output from the Matlab simulation (figure 10.3). The signal to noise ratio was 30dB where the floating point computation result was taken as the signal.

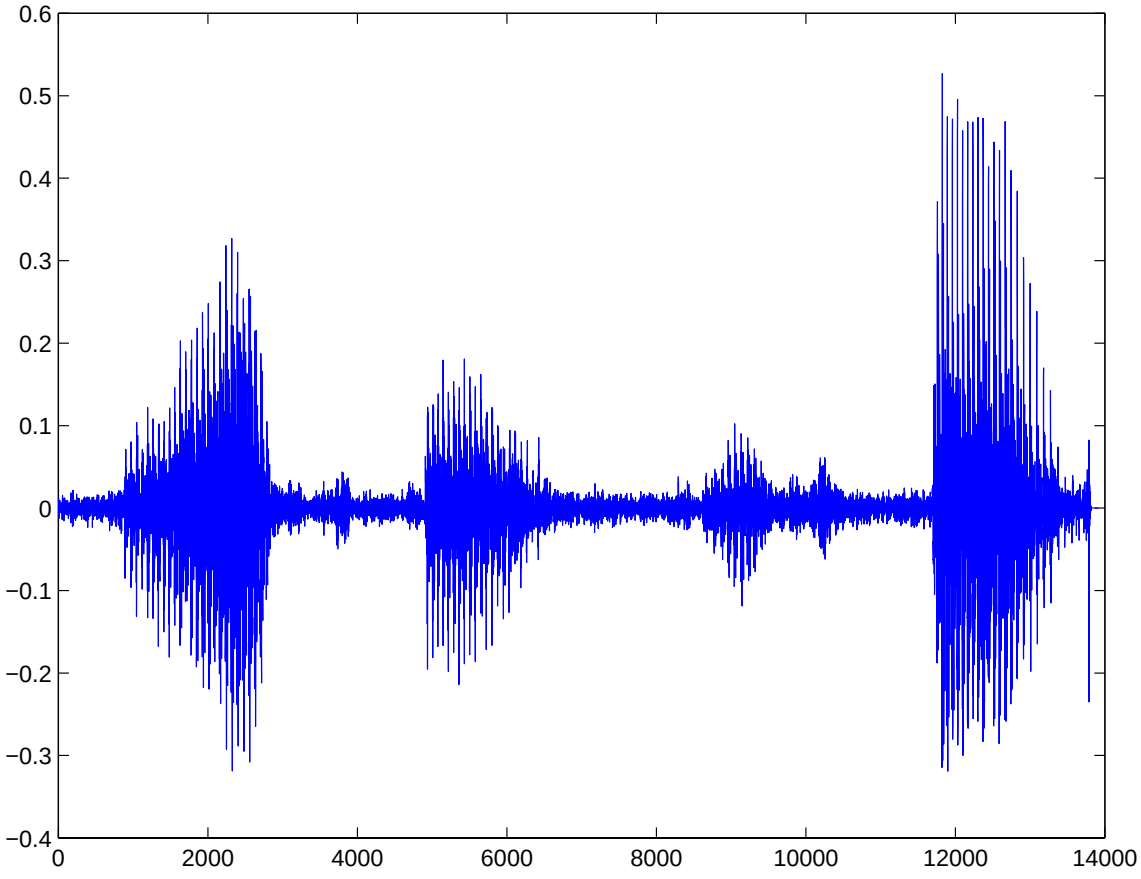

Figure 10.3: Matlab Filtered Output

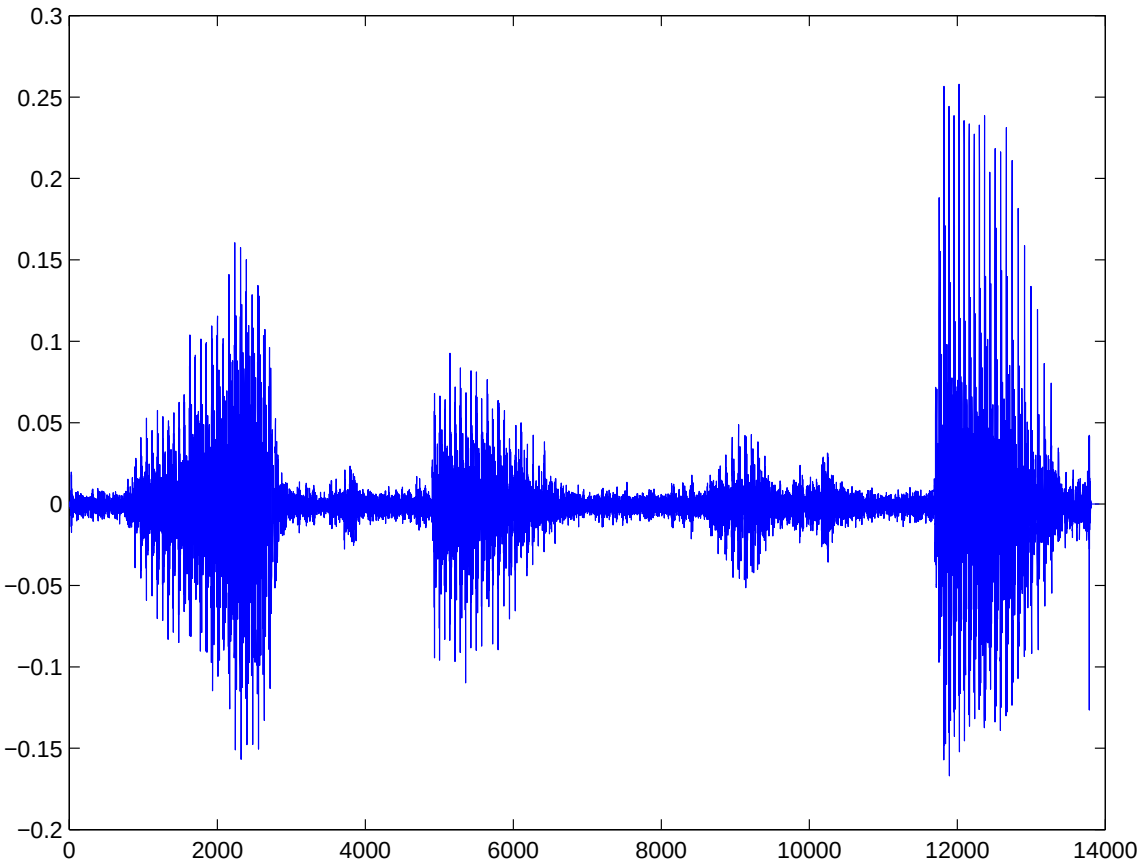

Figure 10.4: 1D- FIR Filtered Output

10.2.2 Throughput Performance

There are $L+1$ multiply/add units in the processor in figure 7.7. The state variable for computing the output value $y(n)$ is propagated from $q_{L+1}(n-1)$ to $q_1(n-1)$. If the processor consists of $L+1$ multiply/add units, then the processor can generate one output value $y(n)$ with one clock cycle. Since the onboard memory is a FIFO, it cannot be written and read at the same time. Therefore, there is no way to retrieve one input data and store back the computed output value into the same FIFO at the same clock cycle. Adding an output FIFO to store the computed result in order to keep the processing speed is not a desirable design. Even when sending the output result directly to the OM, the block data flow bottleneck still exists at the IM FIFO. With this bottleneck, the processor will be waiting for data most of the time and the utilization of the processors is very low.

We used a method to reduce the number of multiply/add units in the processor to avoid both low processor utilization and using a redundant FIFO. In doing so, the number of cycles needed to process each data sample is increased (at least more than two clock cycles). This also means the processing speed for each sample slows down in a processor. However, it is easy to keep up the throughput performance by adding more processors in the PMA and make the data flow processing in a pipeline fashion. This not only provides time to write back the output to the original FIFO but also increases the processor utilization.

	12 proc/4 prim	4 proc/64 prim
Throughput	0.3673 s/c	0.4093 s/c
Utilization	88.92%	39.26%

Table 10.2: 1D-FIR Filter System Performance Table

The performance and the timing information were validated using the Verilog Hardware Description Language (HDL) simulation. Table 10.2 shows the results for two versions of the 1D-FIR system using a mono-clock rate. The second column in the table represents the performance of a system with 12 processors. Each processor has a four multiply/add units and generates one sample output every 16-clock cycles.

The utilization is high because the processors are working most of the times. All the multiply/add units in this system are the same.

The third column represents a version with 4 processors and each processor has 64 multiply/add units. The processor is supposed to generate one sample for each clock cycle if the IM provides sufficient data blocks. There are three different multiply/add units in this version. The multiply/add unit used in both versions is the ***one-section computational primitive***. The second version has better performance, but lower processor utilization. This means the data flow speed can not match the data processing speed and the processors in the PMA are idle most of the time.

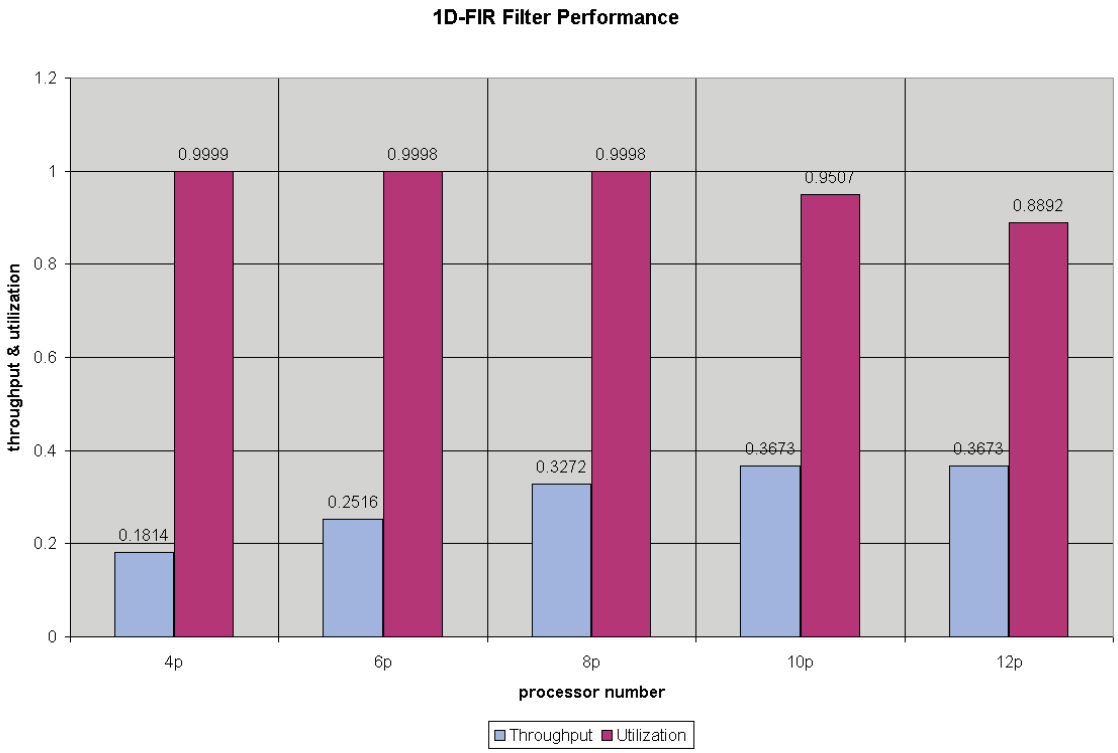

Figure 10.5: 1D-FIR Filter Performance

The system performance of the second column version in table 10.2 is shown in figure 10.5. This 1D-FIR filter system also uses a mono-clock rate system. If the system clock frequency is 100 Mhz, the system throughput can reach more than 36 million samples per second. The small scale linear speedup characteristic is observed

for 4-processors to 10-processors. After 10-processor, the throughput will not be able to increase by adding more processors. The utilization of the processor is going down after adding more than 10 processors. This means there exists a data flow bottleneck at the input/output area at this point. The data flow bottleneck problem can be improved by using a dual clock rate multiprocessor system. This performance improvement was verified by the 2D-FIR filter system simulation. An example of the detail timing information of a signal waveform is shown in the appendix.

10.3 2D-FIR Filter

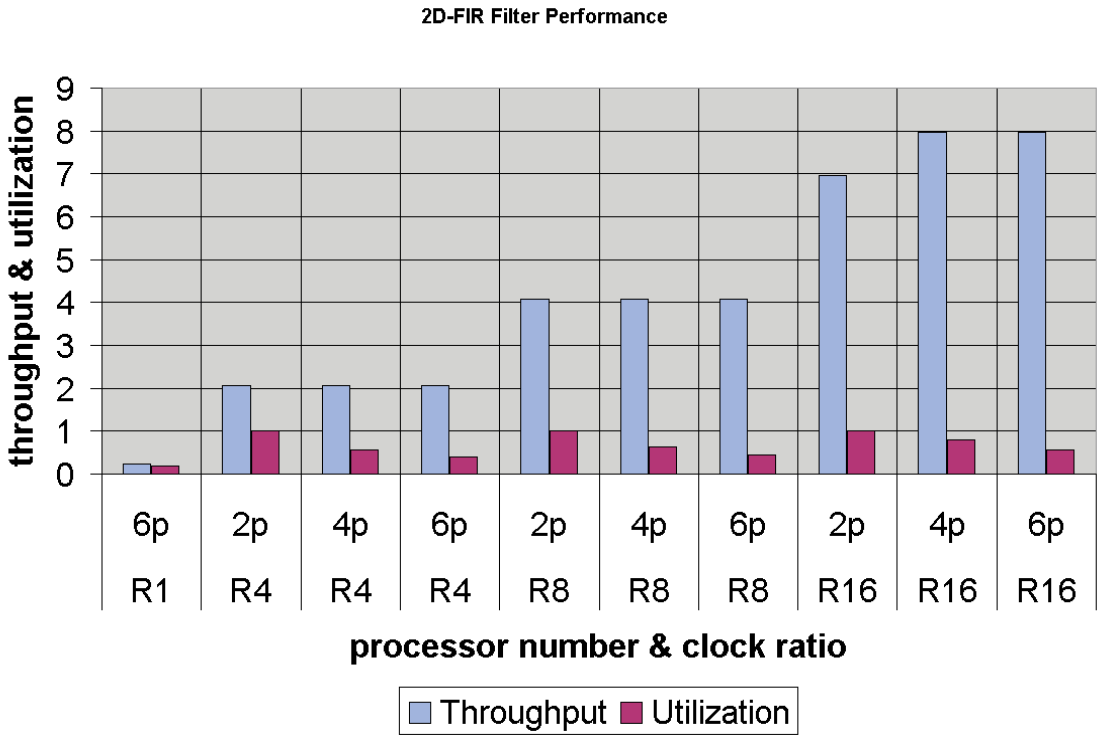

Figure 10.6: 2D-FIR Filter Performance

Figure 10.6 shows a 2D-FIR filter system performance chart. The simulation systems are dual clock rate multiprocessor systems. There are four different clock ratio simulations (R1, R4, R8, and R16), which use different numbers of processors.

R1 stands for a clock ratio equal to one, R4 stands for a clock ratio equal to four, and so on. The two digit linear speedup characteristic can be observed through increasing the clock ratio. For example, the R4 simulation has throughput almost ten times as fast as the R1 simulation and the throughput of the R8 simulation is double that for the R4 simulation. The highest throughput happens at a clock ratio equal to 16 with more than four processors. If the processor clock rate is 100 Mhz, then the throughput obtained here is 800 million pixels per second, which is far more than sufficient to process 1024 x 1024 pixel images and 60 frames per second.

Notice that the two simulations when the clock ratios are four and eight, and the number of processors in the system is four, the utilization of the processors decreases compared to the two processors system's utilization. This means the system clock ratio, which is to drive data blocks into the processors as fast as possible and keep them busy at all times, is still not large enough. Another interesting simulation is when the system has two processors and the clock ratio equals 16. The processors are fully utilized yet the throughput has not reached its linear speedup limit, which is 8 pixels per processor clock cycle. Obviously the clock ratio is big enough for this simulation, driving sufficient data blocks into the processors. However, after adding another two processors in the PMA, the data flow bottleneck appears again.

The purpose of using a dual clock rate system is to increase the input/output data flow speed. By increasing the data flow speed, the processors can obtain enough data blocks to keep busy. The advantage of this technique is that it is not necessary to use a very large memory in the IM or complicated control circuitry to provide the PMA with sufficient data blocks. It is easy for the system to reach its double digit linear speedup high throughput performance by using a smaller number of processors. Of course, the best combination to get rid of the data flow bottleneck problem and to obtain a high throughput performance and high processor utilization is to use both a dual port RAM memory in the IM and the dual clock rate multiprocessor system.

Chapter 11

Conclusion

We have systematically mapped three different algorithms onto the hardware to demonstrate its flexibility and scalability. It is suggested to use an ASIC circuit to implement the fixed modules and a configurable FPGA to implement the parameter variable parts in our system so that we can render a real-time configurable high performance multi-function ASIC.

In order to solve the processing and data flow bottleneck problem, we have developed the hierarchical data flow control, hierarchical processor, and the dual clock rate multiprocessor system. A special four section computational primitive was designed to accommodate the system image processing computation into the 16-bit integer arithmetic operation dynamic range. The flexible ASIC circuit designed in our system has been proven to have high throughput performance and high processor utilization. The dual clock rate multiprocessor system technique can obtain very high throughput with few processors.

Since the FASIC has many variable parameters which can be tailored into different application specifications, a further investigation is necessary to detail the optimization of resources (i.e., block size, clock ratio, and processor number) to accommodate different applications' requirements.

Since it has been proven that three different algorithms can fit into the same hardware, chances are great that other similar algorithms can be accommodated in the architecture of the BDPA. Hopefully, a hardware BDPA library for different

algorithms can be formed in the near future.

Bibliography

[1] Andre' Abrial, Jacky Bouvier, and Marc Renaudin. *A New Contactless Smart Card IC Using An On-Chip Antenna and an Asynchronous Microcontroller.* IEEE journal of Solid-State Circuits, 36(7), July 2001.

[2] Winser E. Alexander, Douglas Reeves, and Clay Gloster Jr. *Parallel Image Processing with the block data parallel architecture.* Proceedings of the IEEE, 84(7):947–968, July 1996.

[3] S. Alexandre, W. Alexander, and D.S. Reeves. *A programmable simulator for analyzing the block data flow architecture.* Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pages 399–400, 1994.

[4] Probir K. Bondyopadhyay. *Moore's Law Governs the Silicon Revolution.* Proceedings of the IEEE, 86(1), January 1998.

[5] Tsuhan Chen. *VLSI Design and Implementation Fuels the Signal-Processing Revolution.* IEEE Signal Processing Magazine, January 1998.

[6] Mohamed Yahia Dabbagh and Winser E. Alexander. *Multiprocessor Implementation of 2-D Denominatior-Separable Digital Filters for Real-Time processing.* IEEE Transactions on Acoustics, Speech, and Signal Processing, 37(6), June 1989.

[7] An-Te Deng and Winser E. Alexander. *Configurable/Scalable Block Data Flow*

Paradigm(CSBDFP). Proceedings of World Multiconference on Systemics, Cybernetics and Informatics, 6(1):57–62, July 2001.

[8] W. W. Edmonson and W. Alexander. *Transient Suppression at the Boundary for 2-D Digital Systems.* IEEE Transactions on Circuits and Systems, 42(11):716–717, November 1995.

[9] Pierpaolo Baglietto et al. *Image Processing on High-Performance RISC System.* Proc. of the IEEE, 84(7):917–929, July 1996.

[10] S. B. Furber. *The Return of Asynchronous Logic.* IEEE International Test Conference, June 1996.

[11] Douglas F. Gray. *Prototype Machine for EUV Chip Technology Unveiled.* IDG News Service, April 2001.

[12] Dan W. Hammerstrom and Daniel P. Lulich. *Image Processing Using One-Dimensional Processor Arrays.* Proceedings of the IEEE, 84(7):1005–1017, July 1996.

[13] S. Howard, Hak-Lim Ko, and W.E. Alexander. *Parallel processing and stability analysis of the Kalman filter.* Conference Proceedings of the 1996 IEEE Fifteenth Annual International Phoenix Conference on Computers and Communications, pages 366–372, 1996.

[14] Margarida F. Jacome and Gustavo de Veciana. *Design Challenges for New Application-Specific Processors.* IEEE Design & Test of Computers, June 2000.

[15] Kurt Keutzer, Sharad Malik, Richard Newton, Jan M. Rabaey, and A. Sangiovanni-Vincentelli. *System-Level Design: Orthogonalization of Concerns and Platform-Based Design.* IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, 19(12), December 2000.

[16] J. H. Kim and W. E. Alexander. *A multiprocessor architecture for two-dimensional digital filters.* IEEE Trans. Comput., 36:876–884, 1987.

[17] Johannes Kneip, Mladen Verekovic, and Peter Pirsch. *An Algorithm-Hardware-System Approach to VLIW Multimedia Processors.* IEEE 0-7803-3780-8/97, 1997.

[18] Christoforos Kozyrakis, Joseph Gebis, Samuel Williams, David Patterson, and Katherine Yelick. *Hardware/Compiler Codevelopment for an Embedded Media Processor.* Proceeding of the IEEE Transactions, 89(11), November 2001.

[19] Wenheng Liu and Viktor K. Prasanna. *Utilizing the Power of HIGH-PERFORMANCE COMPUTING.* IEEE Signal Processing Magazine, September 1998.

[20] Giobanni De Micheli and Rajesh K. Gupta. *Hardware/Software Co-Design.* Proceedings of the IEEE, 85(3), March 1997.

[21] American Institute of Physics. *MIT Researchers Create Rudimentary Atom Laser.* http://www.aip.org/physnews/preview/1997/alaser/index.html, 1997.

[22] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck. Discrete–Time Signal Processing. Prentice Hall, Upper Saddle River, New Jersey 07458, 1999.

[23] A. Peleg and U. Weiser. *MMX technology extension to the Intel architecture.* IEEE Micro, 16(11):42–50, August 1996.

[24] M. Phillip. *A second generation SIMD microprocessor architecture.* in Conf. Rec. Hot Chips X Symp., Palo Alto, CA, August 1998.

[25] B. Ramakrishna Rau and Michael S. Schlansker. *Embedded Computer Architecture and Automation.* IEEE proceeding of Computer Society, April 2001.

[26] R. Rosesser. *A Discrete State Space Model for Linear Image Processing.* IEEE Trans. Automat. Contr., pages 1–10, 1975.

[27] LAWRENCE SPOHN. *Subatomic breakthrough could lead to high-speed computers.* http://www.nando.net/newsroom/ntn/info/110898/info19_18937_noframes .html, 1998.

[28] Marono T. J. Strik, Adwin H. Timmer, Jef L. van Meerbergen, and Gert-Jan van Rootselaar. *Heterogeneous Multiprocessor for the Management or Real-Tine Video and Graphics Streams*. IEEE journal of Solid-State Circuits, 35(11):1722–1731, November 2000.

[29] Wonyong Sung, Sanjit K. Mitra, and Branko Jeren. *Multiprocessor Implementation of Digital Filtering Algorithms Using a Parallel Block Processing Method*. IEEE Transactions on Parallel and Distributed Systems, 3(1):110–120, January 1992.

[30] Michele Taliercio. *SoC in 0.1 μm: A Design and Technology Challenge*. Proceedings of World Multiconference on Systemics, Cybernetics and Informatics, 15(2):528–533, July 2001.

[31] Frank Vahid and Tony Givargis. *platform tuning for embedded system design*. IEEE Computer, Integrated Engineering, pages 112–114, March 2001.

[32] Albert Wang, Earl Killian, Dror Maydan, and Chris Rowen. *Hardware/Sofrware Instruction Set Configurability for System-on-Chip Processors*. ACM, DAC 2001, June 2001.

[33] V.C. Wilburn and W.E. Alexander. *A parallel implementation of the discrete wavelet transform*. Proceedings of the 26th Southeastern Symposium on System Theory, pages 260–264, 1994.

[34] V.C. Wilburn, Hak-Lim Ko, and W.E. Alexander. *An algorithm and architecture for the parallel solution of systems of linear equations*. Conference Proceedings of the 1996 IEEE Fifteenth Annual International Phoenix Conference on Computers and Communications, pages 392–398, 1996.

[35] V.C. Wilburn, S.H. Yoon, and W.E. Alexander. *Real-time occluded object matching using the DWT on a BDFA*. Proceedings of the IEEE-SP International Symposium on Time-Frequency and Time-Scale Analysis, pages 170–173, 1994.

[36] Hongyu Xu and Winser E. Alexander. *A High Performance Architecture for Real-Time Signal Processing and Matrix Operations.* IEEE ISCAS '92. Proceedings., International Symposium on Circuits and Systems, 3:1057–1060, 1992.

[37] Sung H. Yoon, Winser E. Alexander, and Jung H. Kim. *Block Data Flow Implementation of the 2-D Discrete Wavelet Transform.* Proceedings of the IEEE-SP International Symposium on Time-Frequency and Time-Scale Analysis, pages 294–297, 1994.

Appendix A

Simulation Signal Waveforms

The following example is a 2D-FIR filter system using a dual clock rate PMA. There are four processors in the PMA and the clock ratio is 16 to 1, where the processor clock frequency is sixteen times slower than the system clock frequency. The processing image size is 128 x 128, processing block size is 32 x 32, and the filter length is six. We use 4-section computational primitive to do the 16-bit integer arithmetic operation. The example is to explain how the throughput performance is extracted from the simulator signal waveforms, which validates the logic correctness and provides critical timing information.

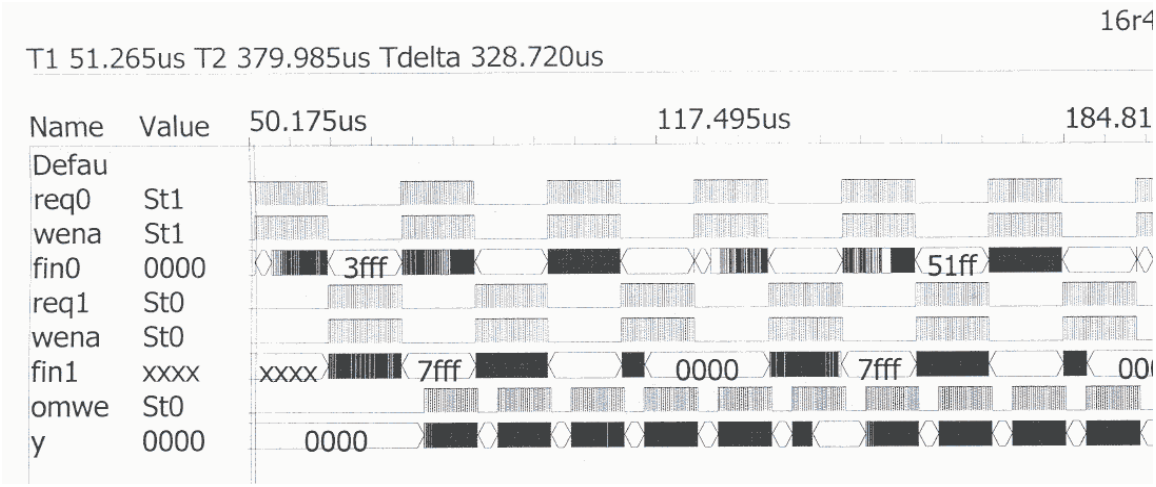

Figure A.1: One Frame Image Processing Waveforms

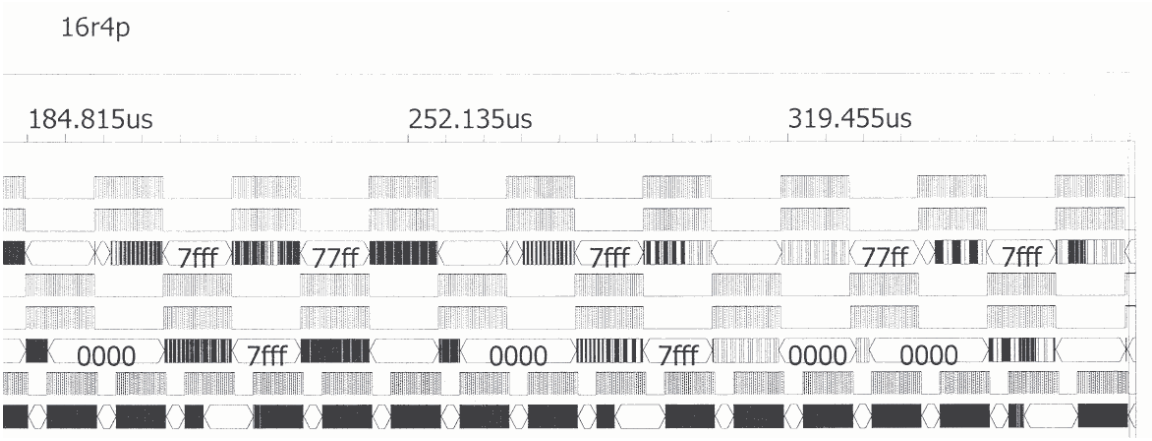

Figure A.2: One Frame Image Processing Waveforms

Figure A.1 and A.2 is the signal waveforms which cover one frame of image processing. The first row of the waveform is the default. The second row waveform, req0, is the request signal from the odd processor group. The third row waveform, wena0, is the write enable signal that generated from the IM. It enables the onboard FIFOs in the PMA to obtain its data block. The forth row waveform, fin0, is the data that input into the onboard FIFO. The following three rows of waveforms are similar to the previous three waveforms, which belongs to the even group of processors. The last two row waveforms are 'omwena', which is the output data write enable signal, and the 'y' is the actual output data.

The first pixel of the image entered the first processor at 51,265 system time and the last processed pixel came out from the PMA at 389,895 system time. The system clock is in 10 system time units. Therefore the total time to process one frame of 128 x 128 image is 328,720 system time units.

Figure A.3 and A.4 is an example which shows the first processor in the PMA idles some time after it finishes its first processing block. The first processor was busy processing its first data block from 51,265 to 88,145. We have four processors in the PMA. So, after the forth processor got its data block, which happened somewhere around 99,885, the first processor started to get its second data block. The idle time for the first processor at this period was 11,760 system time units. We can

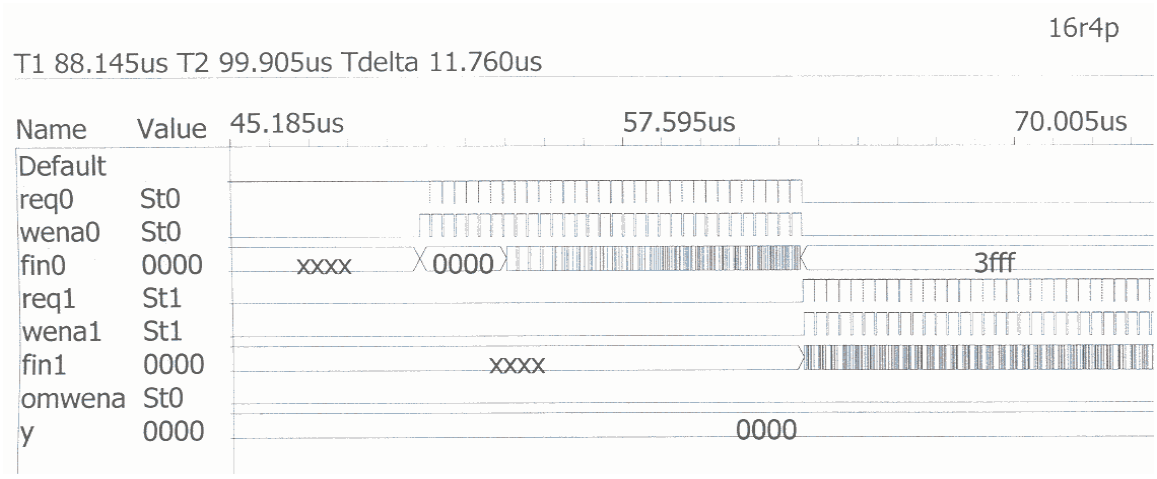

Figure A.3: Example of First Processor Idle Time

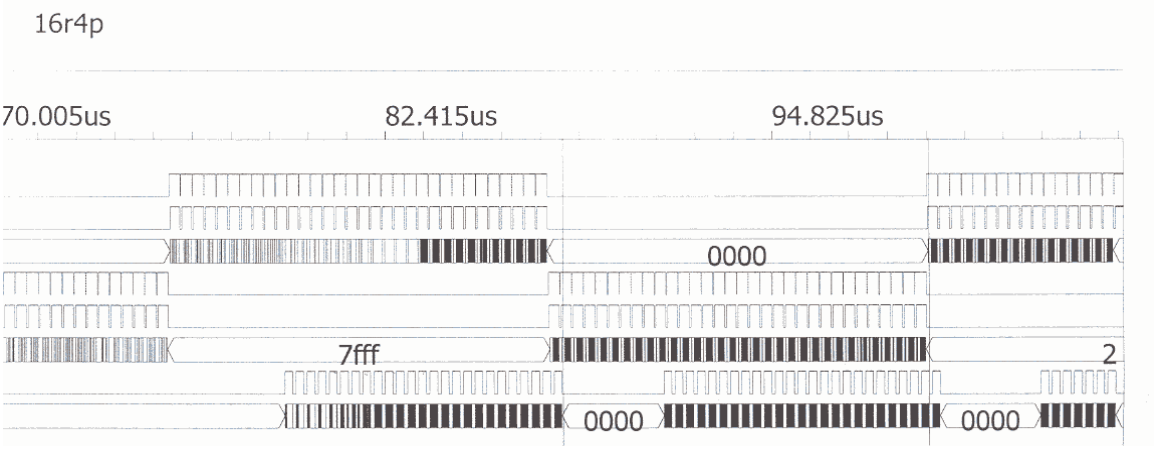

Figure A.4: Example of First Processor Idle Time

accumulate all the idle times for all four processors during processing one frame of image and obtain the utilization of the processors by the following equation.

$$Utilization \ = \ 1 - \frac{total \ idle \ time}{one \ frame \ processing \ time}, \tag{A.1}$$

Where the total idle time is the accumulated total four processor idle time, and the one frame image processing time is the time elapsed for four processors during the one frame image processing time.

The throughput performance is:

$$throughput \ = \ \frac{128 \times 128}{\frac{32872}{16}} \ = \ 7.9747. \tag{A.2}$$

128×128 is the total image pixels that have been processed. 32872 is the total number of system clock cycles that was used and 16 is the clock ratio for the system clock rate and the processor clock rate. If we have a computer system clock frequency that is equal to $1.6GHz$, which is pretty normal nowadays, and the processor clock frequency is equal to $100MHz$, we can have almost 800 million pixels per second throughput performance. The throughput performance is not limited to this number if we scale up the processor number and the data flow speed.

There are thirty-two row-blocks of data in a data block, which is shown in the 'wena0' and 'wnea1' first block waveforms. The output data block contains only twenty-five row-blocks of data in a data block. The seven overlapped row-blocks of data has been eliminated.

Figure A.5 and A.6 is to show that one row-block of data contains 32 pixels. 'T1' was the time the first pixel entered the first row processor. 'T2' was the time when the thirty second pixel finished entering the first processor. The difference time 'Tdelta' is the time elapsed to input 32 pixels into the first row processor.

Figure A.7 and A.8 zooms in to the output row-block and shows that there are 25 pixels in one output row-block. The timing label, 'T1', 'T2', and 'Tdelta' at the left-up corner also shows that this is the last output row-block data. If we further zoom in, we can see the actual value of the last output row-block data in figure A.9 and A.10.

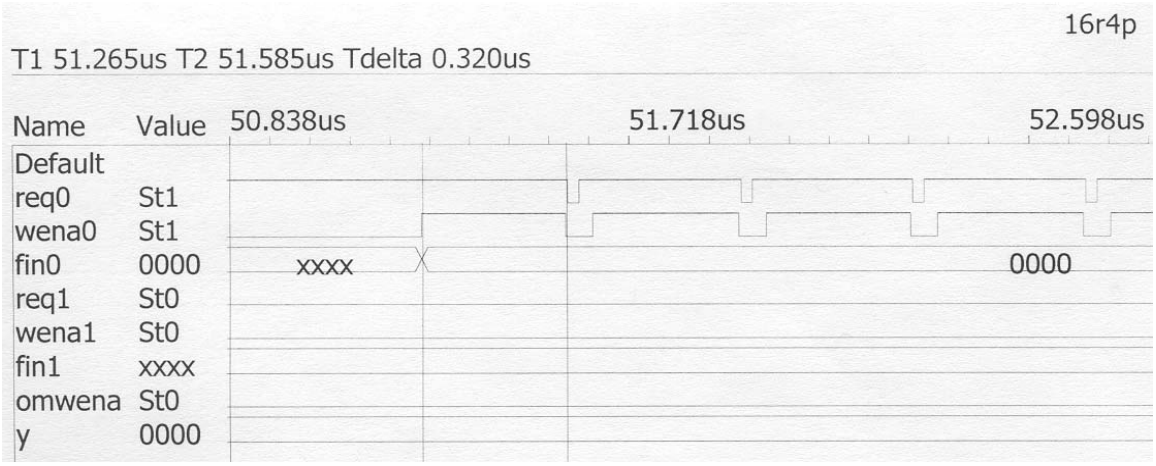

Figure A.5: Row-block Number Waveform

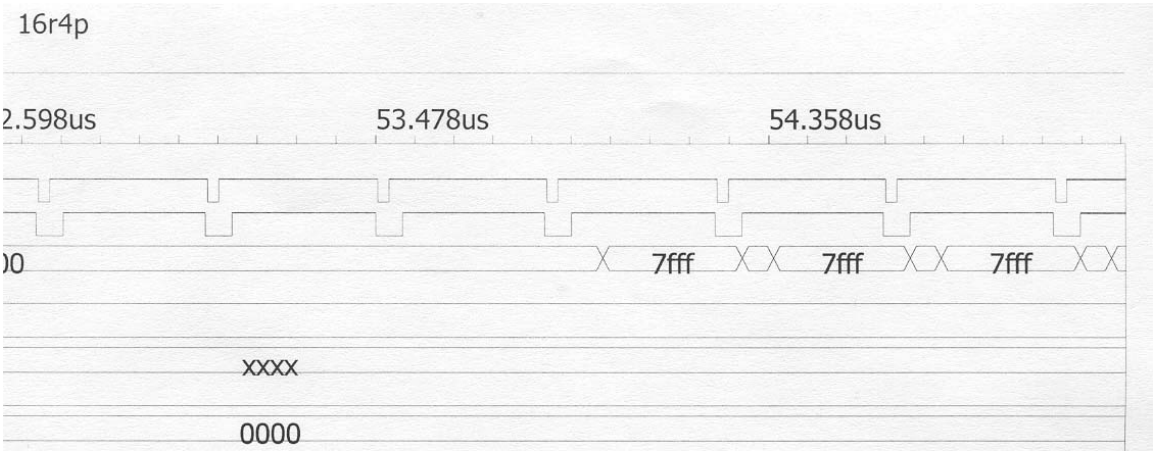

Figure A.6: Row-block Number Waveform

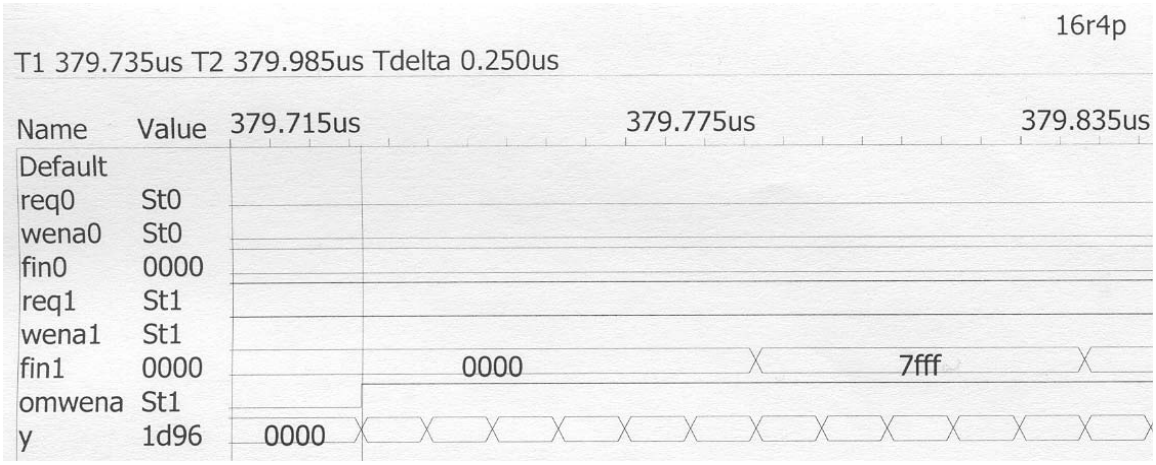

Figure A.7: Output Row-block Number Waveform

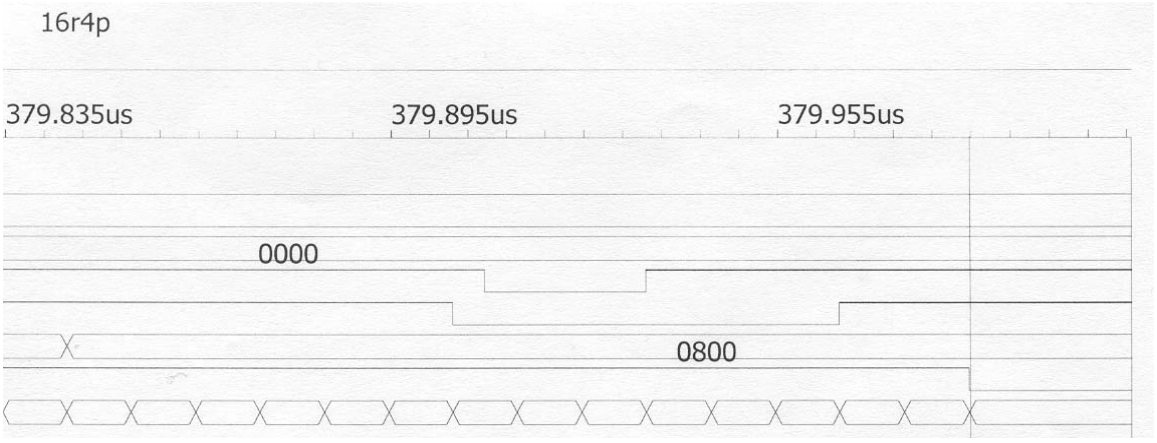

Figure A.8: Output Row-block Number Waveform

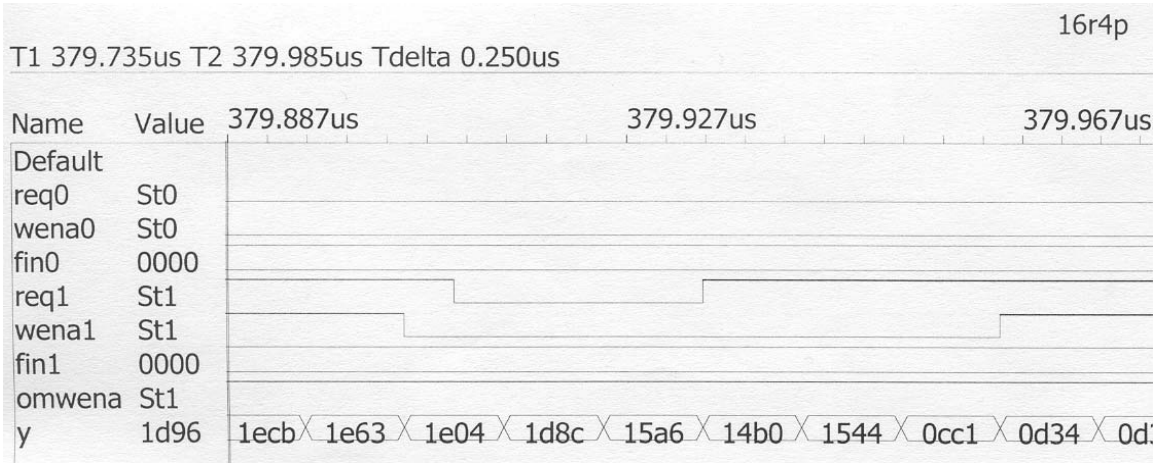

Figure A.9: Output Row-block Value Waveform

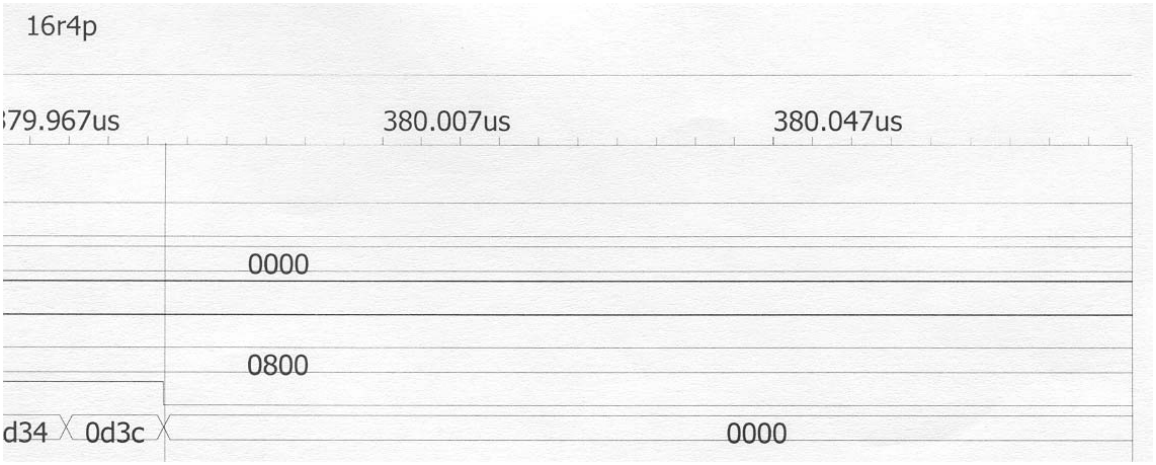

Figure A.10: Output Row-block Value Waveform

Appendix B

Control Pseudo Codes

B.1 2D-FIR Filter System

The bit stream signal is input from the image preprocessing module (IPM) and it is already in data block formats. The bit data block is located in different FIFOs in the IPM and is accessed by the pseudo sequence N_1 and N_2, which is generated by the Flip-Flop counter in the IM. N_1 and N_2 initiate the coordinator in the IPM, which coordinates the proper FIFO in the IPM and transfer the data block into proper FIFOs in the IM.

B.1.1 IM Control

Top Level Control:

1. reset
2. if (two FIFOs in the IM are full with certain block of data)
3. then (enable all the controller in the IM)
4. end if

FIFO (1 or 2) Output Control:

1. Reset

2. If (request signal)
3. then (output data to PMA) // The receiving processor is decided by the token in the PMA.
4. Wait until (output one block of data)
5. then (output data stop)
6. end if

Priority Control:

1. Reset
2. If (PMA request) and (FIFO write)
3. then (freeze the data source)
4. Wait until (PMA request stop)
5. then (release the data source)
6. end if

An example of the priority control Verilog program is shown below.

```
‘timescale 1ns/1ns
module priority(Clk, Rst, req, wena, freeze);
input Clk;
input Rst;
input req;
input wena;
output freeze;
    reg freeze;
    parameter [0:0]
        State0    = 1’d0,
```

```
    State1    = 1'd1;
reg [0:0]
    priorityState, priorityNextState;
reg Nextfreeze;

always @(posedge Clk or negedge Rst) begin
    if (~Rst) begin
        priorityState <= State0;
        freeze <= 1'd1;
    end
    else begin
        priorityState <= priorityNextState;
        freeze <= Nextfreeze;
    end
end

always @(priorityState or req or wena or freeze) begin
    priorityNextState = priorityState;
    Nextfreeze = freeze;
    case (priorityState)
    State0: begin
        if (req & wena) begin
            Nextfreeze = 1'd0;
            priorityNextState = State1;
        end
    end
    State1: begin
        if (~ req) begin
            Nextfreeze = 1'd1;
            priorityNextState = State0;
        end
```

```
        end
        endcase
    end
endmodule
```

Regulator Control:

1. reset
2. if (both FIFOs are full)
3. then (freeze the data source)
4. wait until (PMA request)
5. wait until (one data block is output to PMA)
6. then (release the data source)
7. end if

B.1.2 PMA Control

Token_in Control:

1. reset
2. If (token_in=1)
3. then (ack_out=1, token=1)
4. after one clock cycle
5. then (ack_out=0)
6. if (token=1) and (FIFO empty)
7. then (request =1)

8. wait until (for FIFO is full)
9. then (token_out = 1, request =0, token=0)
10. wait until (ack_in =1)
11. then (token_out = 0)
12. end if
13. end if

Processor Control:

1. if (reset)
2. then (processor lock=0)
3. wait until (FIFO full of input data block)
4. then (processor lock = 1)
5. wait until (FIFO fills with processed result)
6. then (lock = 0)
7. if (token_out =1)
8. then (output data to IM)
9. wait until (FIFO is empty)
10. then (reset the processor)
11. end if
12. end if

FIFO Read/Write/OMread/OMwrite Control

1. reset

2. then (enable FIFO write)
3. if (processor lock = 1)
4. then (enable FIFO read)
5. wait until (all input data being read once)
6. then (stop FIFO read)
7. wait until (FIFO full of processed data)
8. then (stop FIFO write) and (enable FIFO OMread) and (start the eliminating counter)
9. wait until (counter reach set number)
10. then (enable Omwrite)
11. wait until (FIFO is empty)
12. then (disable Omwrite) and (disable Omread) and (enable FIFO write)
13. end if
14. end

Dual Clock Rate Control:

1. reset
2. if (processor start signal appears)
3. then (set the clock rate equals to processor clock frequency)
4. wait until (the FIFO is full of processed data)
5. then (set the clock rate equals to system clock frequency)
6. end if