The Design of a Floating-Point Convolution System

Jean Drolet

B. Sc.A., (Université Laval), 1989

Department of Electrical Engineering

McGill University

Montréal

August, 1992

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Engineering

Abstract

Convolution is the basic operation behind many image processing algorithms. However, it is a computationally intensive operation. Dedicated hardware exists to implement the fixed-point version of this operation. But recent developments such as laser range data processing now require floating-point arithmetic which is often performed by software.

This thesis presents the design of a specialized convolution processor that operates on double precision floating-point data. This convolver is based on an array of systolic cells and may be configured to process both images and unidimensional signals. Support circuitry handles data format conversion as well as data sequencing for the systolic array. In addition, the processor communicates with the memory of a host computer via a DMA (direct memory access) interface to the VMEbus. In this thesis, the design of these auxiliary subsystems is emphasized and their implementation in application specific integrated circuits (ASIC) is presented.

Résumé

La convolution est l'opération à la base des algorithmes de traitement des images. Malheureusement, celle-ci exige un nombre énorme de calculs. Dans le but de réduire le temps de calcul, des processeurs dédiés à arithmétique à point fixe ont été développés. Alors, les progrès récents, notamment dans le traitement des données provenant d'un télémètre a laser, requièrent l'usage d'une arithmétique à point flottant qui doit souvent être exécutée par logiciel.

Ce mémoire présente la conception d'un processeur spécialisé pour la convolution de données en format point flottant double précision. Ce processeur s'articule autour d'une matrice de cellules systoliques et peut être configuré de façon à opérer tant sur des signaux unidimensionnels que des images. Des circuits auxiliaires assument la conversion des données de format point fixe à format point flottant de même que l'alimentation de la matrice systolique avec une séquence de données adéquate. De plus, le processeur communique avec la mémoire d'un ordinateur hôte au moyen d'une interface DMA (accès direct à la mémoire) sur le bus VME. Ce mémoire porte une attention particulière sur la conception des sous-systèmes auxiliaires de même que sur leur implementation à l'aide de circuits intégrés à application spécifique (ASIC).

Acknowledgements

The [illegible] of this thesis would have never been possible without the support of the following people.

First, I want to thank my parents for their love, understanding and encouragements they gave me throughout my experience in graduate studies.

I am [illegible] Alfred Malowany for his patience and his supervision toward [illegible] this long awaited master's thesis. The financial support of the [illegible] and Engineering Research Council of Canada is also gratefully acknowledged.

Thanks are due to J.F. Panisset, J.F. Côté and F. Larochelle who were also involved in this project. They deserve credit for their help and collaboration. I am also thankfull to all undergraduate students who have contributed in one way or another to this convolution system.

Finally, particular thanks to the "veterans" Christian, Kathleen, Marco, Mathieu and Nick for their precious advice and friendship.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Our society is resolutely moving into the information era. In the past decade, the demand for extracting, recognizing, understanding and even conveying the information contained in images has been increasing steadily. There is evidence that this trend will be maintained as we enter the twenty-first century. Indeed, this is a major spur for significant advances in the field of digital image processing.

Image processing consists mainly of transformations such as enhancement, restoration, reconstruction and coding of images. Applications embrace different spheres of human activity. In robot vision, for example, multiple image processing algorithms are used to perform filtering operations, edge detection and other low-level tasks behind motion planning [Nelson and Aloimonos, 1989] and object handling [Tillett, 1989]. Moreover, with common algorithms such as histogram equalization and convolution, automatic visual inspection [Skinner *et al.*, 1990] can be implemented to discriminate between acceptable and non-conforming images of parts in a production chain. Automating the visual inspection task improves speed and reliability since robots are not susceptible to fatigue or boredom which lead to judgement errors.

Perhaps the most impressive application of image processing is computerized tomography (CT). In a medical or industrial environment, CT assists in the reconstruction of density functions (images) derived from the measurements of emanations that have passed through an investigated object or organ [Nowinski, 1990] Most of the time, image reconstruction is achieved by means of the convolution back-projection method. Other biomedical applications include feature extraction for video-endoscopy [Sheblee *et al.*, 1989] and for blood analysis [Parthenis *et al.*, 1990].

In multimedia applications and telecommunications, image processing is often used for video compression. In this case, the principle is to exploit aspects of the human visual system for the coding of a sequence of images [Gall, 1991, Fox, 1991]. However, image reconstruction of a compressed image may also be achieved by means of interpolation [Kung, 1988].

Image processing developments also include algorithms to process and filter images collected from external devices such as satellites, acoustic sensors and scanning tunneling microscopes [Stoll, 1991]. In any case, applications in the fields of remote sensing [Stewart, 1991, Barberi *et al.*, 1991] acoustical imaging and, eventually, the automotive environment [Crisman and Webb, 1991, Kehtarnavaz *et al.*, 1991] depend heavily on the ability of these algorithms to manipulate raw data to obtain as much meaningful information as possible.

As useful as they can be, image processing algorithms are nevertheless computationally intensive, requiring multiple operations per pixel. Considering that a typical image of size 1024×1024 with 256 gray levels requires one Mbytes of computer storage, the throughput required to meet this computational demand is enormous. This is even more the case in real-time applications where a sequence of image frames must be processed at video rate. Uniprocessor computers inspired from the conventional von Neumann architecture [Neumann, 1987] are simply overwhelmed with the large amounts of data that they cannot process efficiently. Therefore, novel computer architectures should be investigated.

Several characteristics of image processing make it suitable for parallel processing. Many algorithms are inherently parallel because an identical operation is applied throughout an image either one pixel at a time (eg. histogram equalization) or one region at a time (eg. convolution/correlation). This implies the notion of data locality which may be exploited for concurrent processing [Lee and Aggarwal, 1990]. In fact, an image may be divided into subimages which can be operated on separately by a set of processing elements working in

parallel. In addition, these algorithms usually apply sequential functions to an image. In this case, a pipeline [Hennessy and Patterson, 1990] architecture may achieve function parallelism to increase throughput significantly.

1.1 Thesis Overview

This thesis presents the design of a specialized parallel system to speed up the computation of the convolution algorithm used in many signal processing applications. This processor operates on double precision floating-point data which circulate in a systolic fashion through the processing elements.

Chapter 2 provides some background information about convolution. Different computer architectures developed in the last decade to implement this processing function are reviewed with emphasis on systolic array developments.

Chapter 3 gives a high-level description of the floating-point convolution processor. The system architecture is presented with a focus on data flow in the systolic cells. Chapter 4 discusses in detail the design of auxiliary units which support operation of the systolic array. These include data converters and a buffering circuit to feed the array of basic processing elements.

In Chapter 5, a short description of the CAE tools that were used during the design process is presented. However, the bulk of this chapter concentrates on the implementation issues of the auxiliary subsystems described in Chapter 4. In particular, simulation results are provided and discussed. Lastly, some board-level considerations are examined. Chapter 6 concludes this presentation with suggestions for future work and improvements to the existing system.

Chapter 2 Background

2.1 Convolution

Discrete convolution is a basic operation widely used in low-level signal and image processing. As opposed to other operations, it can be applied directly to the original time or spatial domain without need to transform the signal into its frequency domain representation. When a discrete signal is passed through a linear time invariant (LTI) system, a modified discrete time sequence is produced at the output. An LTI system [Proakis and Manolakis, 1988] is completely characterized by its transfer function, $h(n)$, namely its response to the unit sample sequence $\delta(n)$ also known as the discrete impulse function. The response $y(n)$ of an LTI system as a function of the input signal $x(n)$ and the unit sample response $h(n)$ is called a convolution sum and is defined as follows:

$$y(n) = \sum_{k=-\infty}^{+\infty} h(k) \times x(n-k) \tag{2.1}$$

Equation 2.1 may be reformulated as follows:

$$y(n) = h(n) * x(n) \tag{2.2}$$

where $*$ denotes the convolution operator.

In reality, however, we are only concerned with causal signals of finite length. Therefore, the convolution formula becomes:

$$y(n) = \sum_{k=0}^{N-1} h(k) \times x(n-k) \tag{2.3}$$

where N is the length of the transfer function $h(n)$ sequence.

In image processing, by extension, convolution is stated as follows:

$$Y(i,j) = \sum_{p=-m/2}^{+m/2} \sum_{q=-n/2}^{+n/2} W(p,q) \times I(i-p, j-q) \tag{2.4}$$

where I is the input image and W is a window of width n and height m. The window, also called kernel, is a 2-D signal equivalent to the 1-D impulse response $h(t)$. By changing the coefficients of the convolution kernel, specific masks can be created to extract features such as edges on an image, to perform linear interpolation or simply to implement various FIR filters. One may see convolution as a neighborhood operation which performs computations on surrounding pixels. As shown in Figure 2.1, convolution is computed by sliding a window over an image in a raster scan pattern. At each pixel location, overlapping pixels are multiplied with the coefficients of the window and then summed. The pixel underlying the center of the window is then replaced with the result of the operation.

Convolution has multiple applications. In low-level image processing, it is applied mainly as a filtering technique. For instance, "smart sensing" is a good practice which aims at using the lowest resolution sufficient for a task so as to reduce the computation time of higher-level operations. As such, Gaussian-like low-pass filters provide an efficient smoothing method for controlling the resolution of an image [Babaud *et al.*, 1986, Deriche, 1990]. By reducing the high frequency content, these low-pass filters may also enhance an image corrupted with high-frequency noise. On the other hand, Marr and Hildreth [Marr and Hildreth, 1980] have shown that the detection of edges and/or lines can be achieved by convolving the image with the Laplacian of a Gaussian ($\nabla^2 G$).

In telecommunication systems where different types of signals (video, speech, facsimile, etc.) need to be transmitted and received, it is sometimes appropriate to decrease the sampling rate before transmission and then perform up-sampling upon reception to restore the original bandwidth of the digital sig-

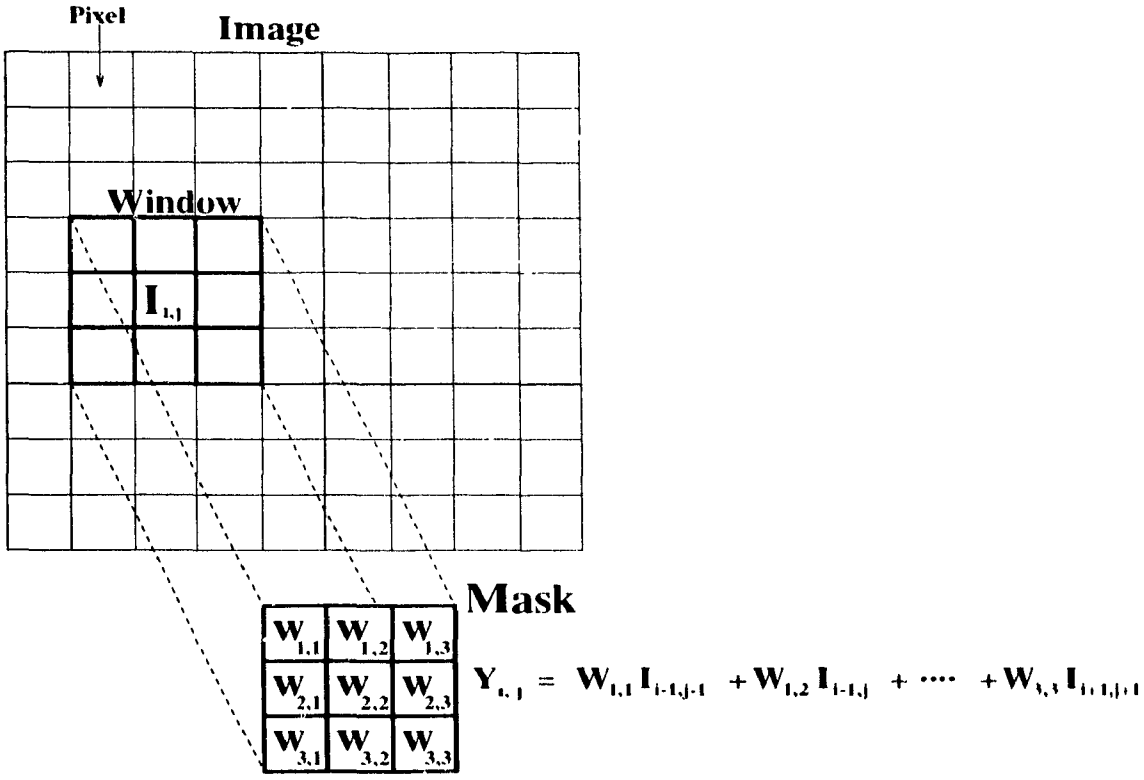


Figure 2.1: 2-D convolution

nal. In this case, the up-sampled signal is convolved with an interpolation filter [Schafer and Rabiner, 1973] to obtain values for the data between samples. In image processing, different interpolation filters are used for anti-aliasing purposes, especially on synthetic images created algorithmically. In another respect, convolution is also extensively used to reconstruct 3-D images by processing data obtained from views of a target object from many different perspectives. In this case, interpolation filters might be used to reconstruct 3-D objects from their 2-D projection in a radar image. Convolution is also applicable to matched filtering [Sklar, 1988], a telecommunication technique which provides the maximum signal-to-noise power ratio for a given transmitted symbol waveform. In a digital matched filter, the received signal is convolved with a filter whose impulse response is the reversed waveform of the transmitted signal.

The problem of convolving an $N \times N$ image with an $M \times M$ kernel has an order of computation of $O(N^2M^2)$. Unfortunately, the usefulness of convolution is

hampered by a tremendous number of computations that grow quadratically with both the image and kernel sizes. This taxes not only the arithmetic capabilities of a general-purpose machine performing the computation, but also the bandwidth of its memory subsystem since computation of a single result requires accessing its M^2-1 neighbors and their corresponding kernel coefficients. Thus low-level image processing has to be performed off-line if computers whose architecture does not match the computational requirements of convolution are employed.

2.2 Architectures for Convolution

Advances in very large scale integration (VLSI) technology over the last decade have made possible the realization of various parallel processors which were formerly impractical to build because of extravagant size and cost. The following sections present an overview of those parallel architectures that are particularly suitable for convolution.

2.2.1 Systolic Architecture

The concept of a systolic array was first introduced in the late seventies by Kung and Leiserson [Kung and Leiserson, 1978] as an attempt to achieve more efficient computing from silicon by balancing computation with I/O bandwidth. In his excellent paper [Kung, 1982] on the basic principle of systolic architectures, Kung writes:

> "In a systolic system, data flows from the computer memory in a rhythmic fashion, passing through many processing elements before it returns to memory, much as blood circulates to and from the heart."

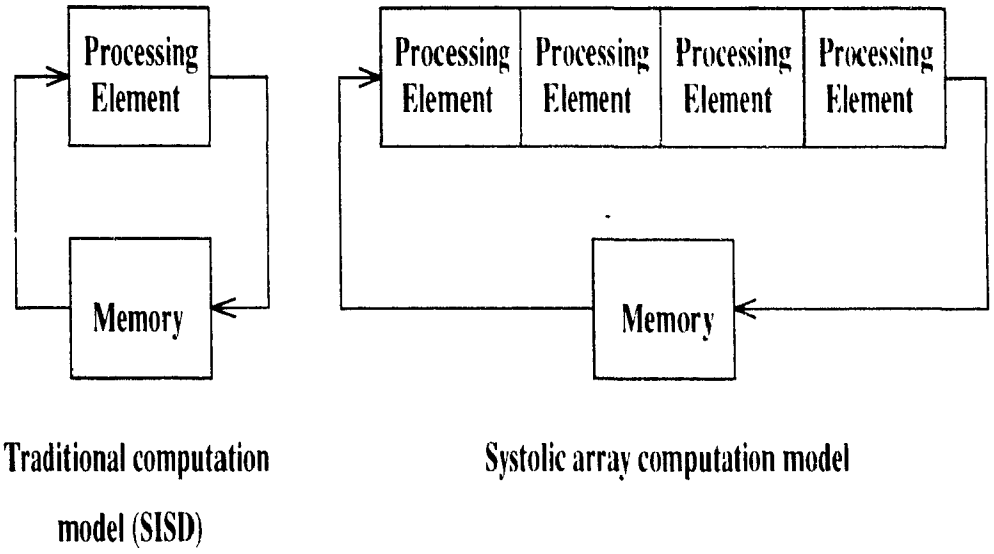


Figure 2.2: Comparison between traditional and systolic architectures

By performing multiple computations for each memory access, the systolic architecture can speed up execution of compute-bound problems such as convolution without increasing I/O requirements. Figure 2.2 illustrates this concept

A typical systolic array exhibits the following architectural characteristics.

- **Modularity:** The structure is made of simple, similar building blocks also called cells. They connect to each other through regular, well defined interfaces. As such, the array may be extended indefinitely From a VLSI standpoint, modularity and regularity are two attributes that make systolic arrays very attractive [Mead and Conway, 1980]

- **Synchronicity:** A global clock synchronizes the operation of the cells. However, one of the limiting factors in building large systolic arrays is the difficulty to achieve proper and reliable synchronization due to clock skews. Different clock distribution schemes have been proposed to overcome this problem [Dikaiakos and Steiglitz, 1991, Fisher and Kung, 1984].

- **Locality:** The cells exhibit local connections to restrict circulation of the data to immediate neighbors. Only the boundary cells of the array may perform I/O to and from memory. Contrary to other parallel architectures which suffer

from interprocessor communication overhead when the number of processing elements increases, a systolic array avoids irregular or long distance data communication which makes it easily expandable

- **Concurrency:** The processing power of the systolic architecture comes from pipelining the stages involved in the computation. At the array-level, each cell processes the information in such a way that the output it generates is used as an input to a neighboring processing element. In order to permit even higher concurrency and throughput, another level of pipelining may be introduced, if possible, to allow the operations inside the cells to be pipelined as well [Kung *et al.*, 1981, Kung and Webb, 1984].

Various schemes have been proposed to map different image processing algorithms and matrix operations onto systolic topologies [Kung, 1982, Kwan and Samuel, 1990, Moreno and Lang, 1990]. Since systolic arrays are frequently implemented as special-purpose or dedicated devices acting in conjunction with a host CPU, they offer high performance at a cost which might not always be justified. For this reason, efforts have been deployed to develop arrays with programmable processing elements that can be used for multiple compute-bound problems.

The Warp machine [Annaratone, 1987] for instance, is a systolic array developed at Carnegie-Mellon University for many applications including signal and image processing and autonomous navigation for a robot vehicle called Navlab [Crisman and Webb, 1991]. The heart of the Warp computer is a short linear array consisting of ten cells. Each cell is a 10-MFLOPS processor with a local program and data memory which can be programmed using a high-level language. The Warp array is attached to an external host consisting of MC68020 processors for transferring data to and from the array. A VMEbus-based workstation provides the environment for running applications programs.

The first commercial general-purpose systolic computer, Matrix-1, from Saxpy Computer Corporation [Foulser, 1987] was introduced in 1987. It achieves 1000 MFLOPS by means of an array of up to 32 computational zones. Each zone has a pipelined 32-bit adder and a multiplier with the same characteristics. Like in a single-instruction multiple-data (SIMD) architecture, each processor receives the same instructions at a given clock cycle. Matrix-1 can function in systolic mode in which data are transferred linearly across the zones or in block mode in which all zones operate independently.

Other non-programmable forms of systolic hardware have been developed [Nash and Petrozolin, 1985, Kandle, 1987] to address specific problems. For instance, Lopresti [Lopresti, 1987] describes an original application where a systolic array is used for comparing nucleic acid sequences.

Despite the introduction of a limited number of systolic machines, this paradigm has not yet delivered its promised fruits as few machines have found their way to the marketplace. Manohar and Baudet [Manohar and Baudet, 1990] identify two major limitations of current systolic designs. The first relates to the fact that systolic algorithms often require a processor array whose size depends on the size of the problem to be solved. The second drawback is due to the limited bandwidth between the host computer and the array. As a matter of fact, some systolic architectures require a bandwidth proportional to the number of processing elements in the array. The authors propose some solutions to make systolic solutions more practical.

2.2.2 Other Parallel Architectures

Over the last decade, novel computer architectures for parallel processing have been introduced. In fact, their wide variety is now forcing us to question Flynn's taxonomy [Flynn, 1966] of computers based on instruction and data streams. Flynn

essentially proposed four categories of computers.

1. **SISD** (single instruction stream, single data stream) which corresponds to the uniprocessor
2. **SIMD** (single instruction stream, multiple data streams).
3. **MISD** (multiple instruction streams, single data stream). This is a rather theoretical category as it involves multiple processors applying different instructions to a single datum.
4. **MIMD** (multiple instruction streams, multiple data streams).

This is a rather coarse model which does not provide for hybrid architectures such as pipelined vector computers and systolic arrays. While maintaining the essential ideas of Flynn's taxonomy, Duncan [Duncan, 1990] classifies computer architectures into three categories: Synchronous, MIMD and MIMD paradigm.

Synchronous computers include SIMD and systolic architectures as well as vector computers such as the Cray X-MP [Robbins and Robbins, 1989][1]. Processors of this category perform concurrent operations in lockstep since they are synchronized with either central control units, vector unit controllers or global clocks. SIMD architectures with an array topology are particularly suited for image processing since an identical set of operations is applied throughout an image either pixel by pixel or region by region. Therefore an image may be divided into N subimages which can be processed concurrently by N processing elements. For instance, the access constrained memory array architecture (ACMAA) [Balsara and Irwin, 1991] is a SIMD architecture consisting of a linear array of N processors and an $N \times N$ array of memory modules. Each processor has two buses, one to access a row and one to access a column of memory. With an ACMAA of size N, it is possible to convolve an image of size $N \times N$ with a kernel of size $M \times M$ in $O(M^2N)$ time

[1]The Cray X-MP could arguably be termed MIMD. However, vector processing remains the fundamental characteristic of this computer.

compared to $O(M^2N^2)$ with a SISD machine. Another popular SIMD is the Connection Machine from Thinking Machines Corporation [Hillis, 1985] whose massive parallelism makes it pertinent to low-level vision problems. The model CM-2 can be configured with between 16384 and 65536 1-bit processors. Although big and expensive, the CM-2 is a key element of the DARPA Strategic Computing Vision program [Weems *et al*, 1991] which has implemented vision algorithms for this revolutionary architecture.

The second category, MIMD, involves architectures that are inherently more flexible since processors may be individually programmed. MIMDs may be further divided into shared-memory and distributed-memory processors. Shared-memory computers accomplish interprocessor coordination by having a shared, global memory addressable by each processor while distributed-memory computers achieve coordination by sending messages to each other. An example of shared-memory MIMD computer is the Sequent Symmetry Multiprocessor [Lovett and Thakkar, 1988]. One of the major problems with this architecture turns out to be cache coherency. Special mechanisms such as hardware "snooping" are therefore used to determine when shared memory has been updated. On the other hand, the T414 transputer chip [Whitby-Strevens, 1985] provides hardware support for concurrency and communication, both of which are essential to any distributed-memory MIMD machine. With its 32-bit RISC processor, the transputer may therefore be used as a powerful building block from which new parallel devices may be built. One of the principal challenges of distributed-memory architectures consists in designing a scalable computer which would achieve linear speedup as the number of processors increases. This is especially difficult to achieve because of fast-growing communication overhead. Incidentally, an hypercube architecture [Freer, 1987] is an attempt to reduce this overhead. A hypercube of dimension n connects together $N = 2^n$ nodes in such a way that only one additional communication channel must be added to each node in order to double the number of processors.

The last category, MIMD paradigm, includes architectures based on MIMD principles but having a fundamentally distinctive working concept. This group includes MIMD/SIMD hybrids, dataflow and wavefront architectures. Basically, dataflow computers use the flow of data to initiate the execution of an operation. Consequently, an operation or instruction may execute as soon as all of its operands become available. However, these machines require a powerful supervising system which involves additional hardware and/or software. Datawave [Schmidt and Caesar, 1991] is a good example of a processor which falls into this category. This 4-GOPS (giga operations per second) processor consists of 16 mesh-connected cells characterized by a systolic array topology and built-in dataflow control. By integrating 16 cells in a 1.2-million-transistor chip, Datawave is currently one of the rare multiprocessors to fit within a single chip. Interestingly, it can perform real-time image compression/decompression based on the Joint Photographic Experts Group (JPEG) standard which is vital for many multimedia applications.

Chapter 3 A Floating-Point Convolution Processor

Recent advances in computer vision such as laser range data processing [Malowany and Malowany, 1988] have led to the use of larger convolution kernel sizes that accommodate floating-point arithmetic. While yielding a larger dynamic range and a higher accuracy by minimizing round-off noise, floating-point arithmetic taxes even more the computational capabilities of any general-purpose machine. For instance, a convolution with a 9×9 kernel on a typical 512×512 image necessitates over 42 million floating-point operations. Typical workstations such as SPARCstations currently available at the McGill Research Center for Intelligent Machines (McRCIM) yield a computational capability limited to a few MFLOPS. When performed on these machines, the aforementioned convolution requires in the order of 10 seconds of CPU time to complete assuming that the floating-point unit is kept continuously busy. However, this figure is highly optimistic since the maximum MFLOPS rate cannot be sustained because of the traditional memory bandwidth bottleneck.

The pursuit of ever faster processors with real-time processing capabilities has lead engineers to develop high-performance architectures for convolution. So far, most dedicated systems have been limited to fixed-point arithmetic and small kernel sizes. For instance, a system previously designed at McGill University for robotic applications used 8-bit integer coefficients and a 3×9 window convolution size [Haule, 1990]. With multiple passes, this system would perform a 9×9 convolution on a 512×512 image in about one second. Others have used digital signal processors (DSPs) to implement both fixed- and floating-point convolutions. For example, the popular TMS320C30 from Texas Instruments yields a 33 MFLOPS performance [Papamichalis and Simar, 1986, Lan *et al.*, 1987] and therefore could theoretically complete a convolution in about 1.25 second provided that

the pipeline is kept continuously busy.

In order to further decrease the time spent for convolution, it was decided to develop a systolic floating-point convolution processor which could achieve the aforementioned convolution in under one second. In addition, the processor would be implemented as an "intelligent" peripheral which would easily integrate into the research environment, in this case, a multiprocessor VMEbus-based system called the Sensor Computing Environment [McRCIM, 1990]. The SCE includes a number of single-board computers and peripheral boards such as a laser rangefinder and a variable-scan camera which communicate through the VMEbus backplane. They run under VxWorks (WindRiver Software), a real-time "flavor" of the widely used UNIX operating system.

The following sections present the architecture of the systolic floating-point convolution processor which has resulted from this effort.

3.1 System Architecture

As shown in Figure 3.1, the core of the system is the systolic array of custom VLSI processors. Each processor implements the basic multiply and accumulate operation in IEEE double precision format. The array is configured in 9 rows of 9 custom chips and allows a 9 by 9 kernel to be applied in a single pass. The array can also be configured for 1-D data, in which case an FIR filter with 81 coefficients can be implemented.

One of the more challenging problems in designing around array processors is to move data to and from the array efficiently while keeping the processors as busy as possible. Unlike general-purpose computers with load and store instructions, dedicated systolic processors need additional circuitry to supply the data sequence to the array and to store it after processing. Since most of the image processing research is done on workstations based on the VMEbus, the system includes a DMA

engine built from an embedded Motorola 68020 microprocessor and a VTC VIC-068 VMEbus interface controller. The DMA engine is responsible for transferring images (or signals) from a host computer memory and for writing the convolved image back to host memory.

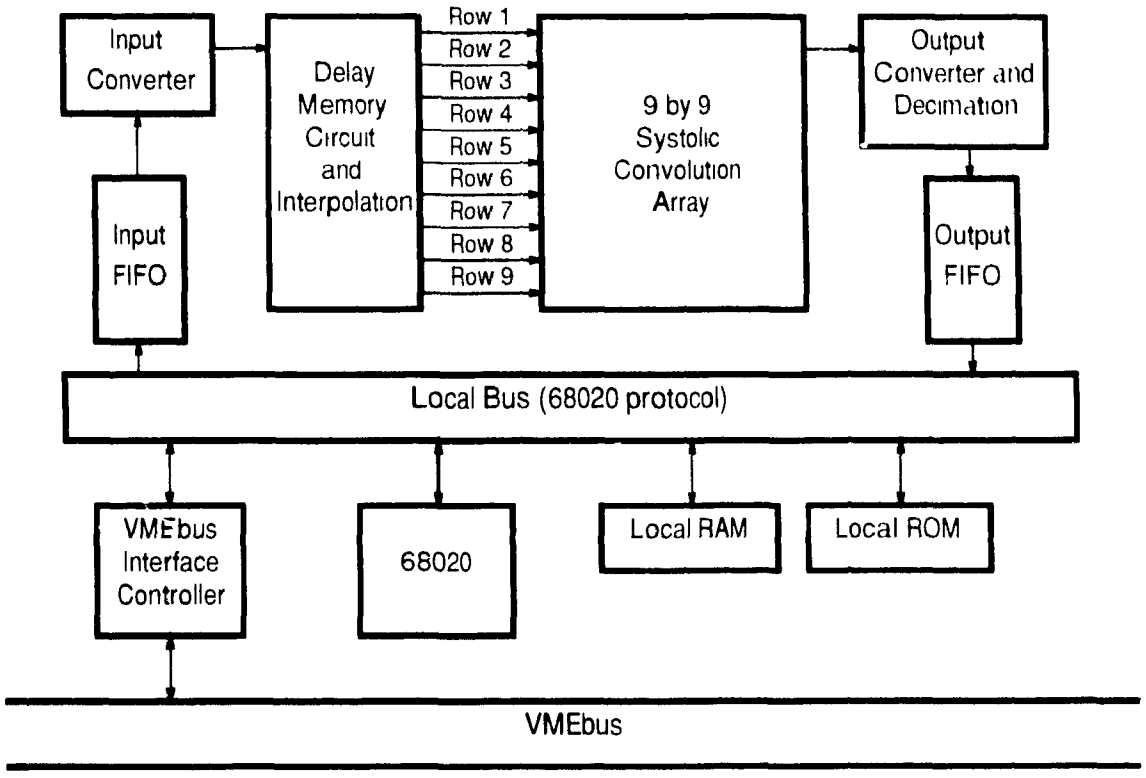


Figure 3.1: Architecture of the convolution processor

The DMA engine reads the source image with 4K transfers into the input FIFO (first-in, first-out) buffer. As depicted in Figure 3.1, an input converter reads data from the FIFO and converts it into double precision floating-point numbers suitable for the systolic array processors. This converter is required since image-processing data often originates in integer format. For example, most frame grabbers generate 8-bit data; the laser rangefinder used at McRCIM generates 16-bit values. The input converter can also pass along data already in floating-point format such as the results of intermediary computations where the full precision of floating-point is desired.

A delay memory circuit (DMC) then takes care of feeding the lines of the image

Feature	Description
Architecture	Systolic
Signal types	1-D or 2-D
Nb of processors	81
Kernel configuration	9 × 9 (2-D) 81 × 1 (1-D)
Arithmetic	IEEE 754 Double Precision F.P. (64 bits)
Bus	VME
Input data	8/16-bit integers or 64-bit F.P.
Output data	8-bit integers or 64-bit F.P.
Interpolation	Up-sampling (2x or 4x)
DMA engine	MC68020 with VTC VIC-068
Estimated performance	126 MFLOPS with 12.5 MHz clock

Table 3.1: Features of the convolution system

to the convolution array in the proper sequence. Each line is sent to the array 9 times, once for each row in the array. In addition, the DMC handles the border effects by extending the source image with a border of zero-valued samples. It may also be used to raise the sampling rate of the data being processed by inserting zero-valued samples into the original data stream. When combined with the proper coefficients in the systolic array, different filters can be implemented to perform interpolation.

Data coming out of the systolic array is processed by the output converter which maps the floating-point numbers back into integer format if required. This proves to be useful when the resulting image is to be displayed on a monitor (CRT). The output converter may also be bypassed if high accuracy is desired, especially when performing multiple passes on the image. The output converter writes to the output FIFO buffer. The DMA engine ensures that no data are lost by

transferring convolved samples back to host memory whenever the FIFO buffer becomes half-full.

Table 3.1 presents some of the characteristics of the convolution system. Further details are given in the following sections.

3.2 The Systolic Array

The systolic array consists of 81 processing elements also called cells, which communicate with two of their neighbors. On the board each cell is physically represented by a 40-pin chip of which 34 pins are used. The current version of the cell implements approximately 49 000 transistors. [Côté, 1990, Larochelle *et al.*, 1989] The chip has been designed in the McGill VLSI laboratory in compliance with the Northern Telecom CMOS3 DLM technology [Can, 1989] The CMOS3 DLM is a 3-micron P-well CMOS process with single-level polysilicon and double-level metal. Circuit fabrication is available through the Canadian Microelectronics Corporation (CMC) which operates an integrated circuit implementation service offered to all major universities across Canada. At the time of this writing, the chip had been revised and re-submitted to the CMC for fabrication.

3.2.1 Data Flow in the Systolic Cell

The systolic cell executes the following operation:

$$Yout = (Xin * C) + Yin \tag{3.1}$$

An incoming value (Xin) is multiplied with a coefficient (C) loaded into the cell prior to convolution. The result is then added to the partial sum (Yin) coming from the previous processor. Each operand (64-bit floating-point number) is serially loaded

by groups of four bits to accommodate the pin packaging constraints. The familiar multiply and accumulate operation is easily divided into simpler operations which can be implemented with three pipeline stages. By allowing the arithmetic operations inside the cells to be pipelined, a significant increase in throughput is realized since the system cycle time becomes equal to the time of a single pipeline stage in the cell rather than the whole cell cycle time. The three processing stages are shown in Figure 3.2 and execute the multiplication, the addition and the normalization. Each stage requires 16 clock cycles to complete (this is called a pipeline cycle) and thus matches the rate at which the operands are loaded into the cell (16 cycles × 4 bits/cycle = 64 bits). The pipeline also implements some shift registers to delay the data flow for proper coordination of the operands.

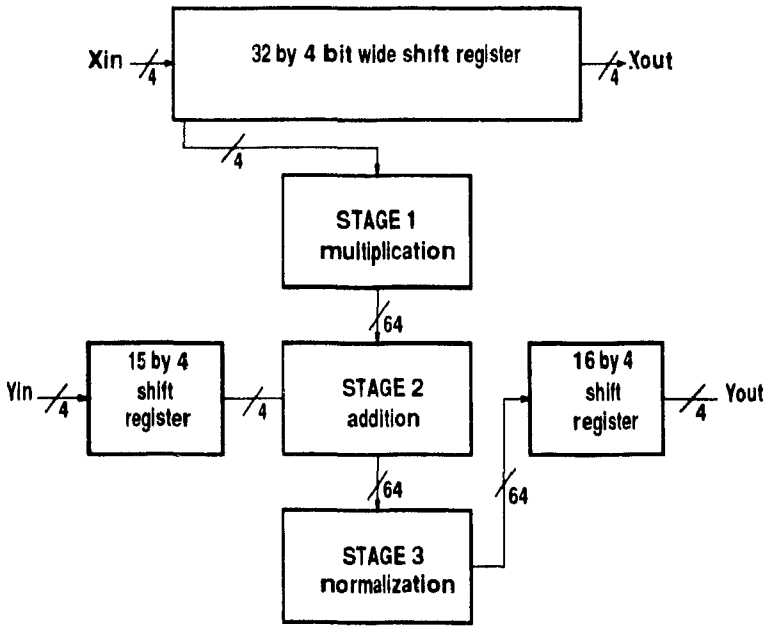


Figure 3.2: Systolic cell architecture

3.2.2 Data Flow in the Systolic Array

A systolic cell has the ability to communicate synchronously with other similar cells. This property allows the realization of various arrays of systolic cells working concurrently. Therefore, it is possible to introduce pipelining at the array

level which will further increase throughput. At this level, each cell becomes a pipeline stage which circulates a pixel intensity value and a partial result to the next processor.

In this convolver architecture the interconnection between the cells can be configured to operate on 1-D or 2-D data. In the 1-D configuration, the systolic cells are simply cascaded to form a convolution window of size 81×1. In the 2-D configuration, the cells are interconnected to create a 9×9 window. The mode of operation can easily be changed with multiplexers as shown in Figure 3.3. Due to space constraints, a smaller array of size 3×3 is illustrated but the concept is similar for the 9×9 array.

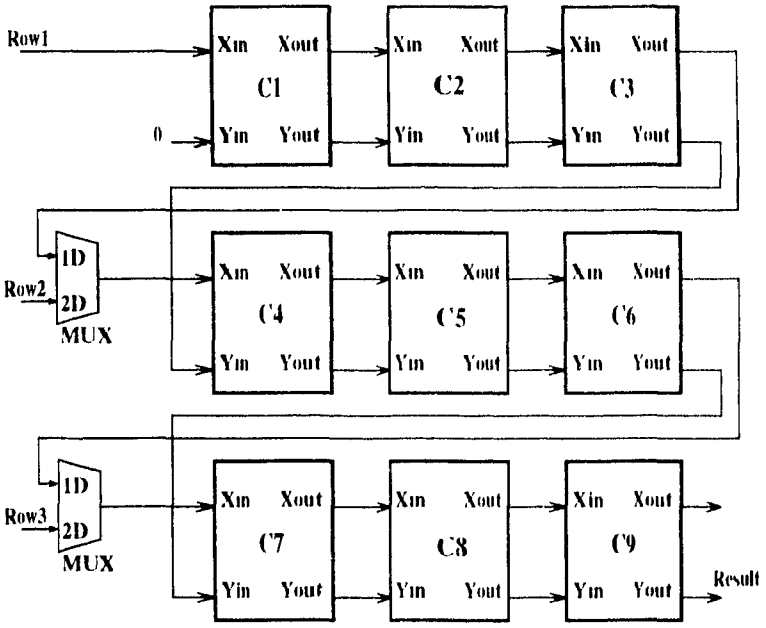


Figure 3.3: Systolic array for convolution

When configured to operate on 1-D data, a sample entering the array will be circulated systolically through the 81 processors. The convolved sample may therefore be collected at the output of the last cell after k cycles where k is the number of pipeline stages inside a cell times the number of cells. In this case, $k = 243$ since there are 81 cells implementing 3 arithmetic stages each. To further illustrate how data flow through the array, Figure 3.4 shows the contents of two adjacent systolic cells pre-loaded with coefficients C0 and C1 at four successive

cycles, t0, t1, t2 and t3. It is assumed that these snapshots show the content of the registers at the end of a pipeline cycle and that blank registers hold garbage. From this picture, it should be apparent that, when the pipeline is full, a convolved sample is produced every cycle.

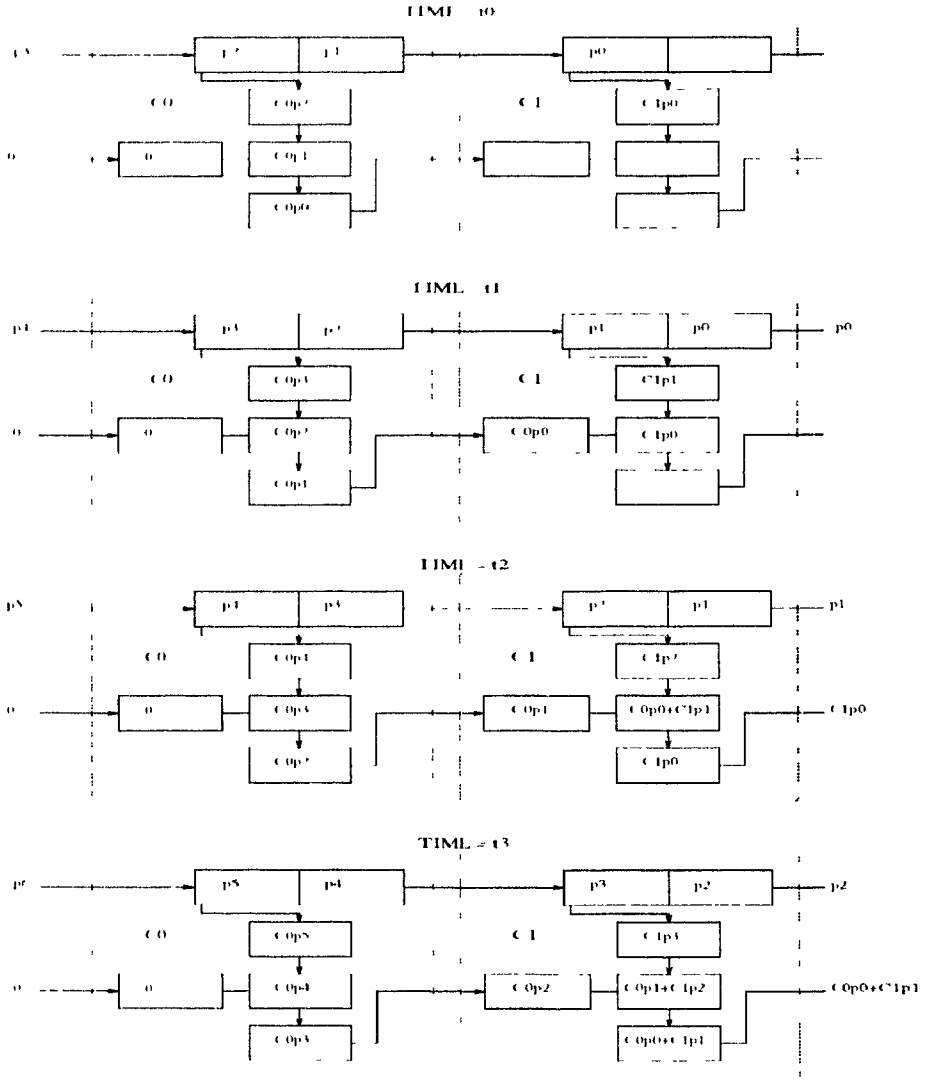


Figure 3.4: Data flow in the systolic array

The systolic array may also be configured to operate on two-dimensional data usually represented as images. In this mode, the pixels move through an array of 9 rows of 9 cells each (i.e. 81 cells), so that the pipeline delay ($k = 243$) is exactly the same as in the 1-D mode. However, the data flow differs since the convolution window will overlap 9 rows of an image at any time. It can be easily inferred from

Figure 3.3 that, at each cycle, 9 pixels enter the array.

Regarding a raster-scanned image as a 1-D array of pixels, then, at time t, the following 9 pixels enter the array:

- pixel p enters row 1
- pixel $p + w - d$ enters row 2
- pixel $p + 2(w - d)$ enters row 3
- pixel $p + (n - 1)(w - d)$ enters row n

where w corresponds to the width of the image and d is the delay incurred in traversing one row of the array. Thus, the delay is equal to the number of pipeline stages in one row. In this design, $d = 27$ since there are 9 cells of 3 stages each per row. As the convolution window scans the rows toward the bottom of the image, it can be seen that the pixels will be sent 9 times to the array except for those near the top and bottom borders.

3.3 The Delay Memory Circuit

In this convolver architecture, pixels have to be read only once from host memory even though they are used in 81 multiplications. This data re-use capability greatly decreases traffic on the VMEbus and further enables the convolution board to use the bus sporadically so that other master devices may take control of the VMEbus during a convolution operation. However, this bandwidth reduction is realized at the expense of hardware complexity on the board. In particular, a delay memory circuit (DMC) stores multiple image rows so that pixels need only be fetched from memory once although they are sent to the systolic array 9 times.

An image is usually acquired by a raster scan method which produces a 1-D array of pixels stored in host memory. The DMA engine transfers the image from

host memory to the board in one sequential stream of pixels. Starting from the upper left corner, each row is transferred one after the other down to the lower right corner of the image. When operating in 2-D mode, the DMC must re-organize the incoming data sequence so as to create 9 streams, one for each row of the array. Conceptually, all streams are similar but delayed from each other by a constant factor. The "stream delay", expressed in pixels, is equal to the width of the image minus the pipeline delay. In order to accommodate this particularity, the DMC implements 8 circular RAM buffers through which the incoming pixel sequence circulates. Rows of the systolic array receive data from their respective circular buffer except row 9 at the bottom which is fed directly with the incoming pixel stream. The internal working of the DMC will be presented in chapter 4.

When operating in 1-D mode, the data stream sent to the array by the DMC will be the same as the sequence in which the signal is transferred to the board. Therefore, the task of the DMC comes down to taking care of the border effects at the beginning and at the end of the data stream.

3.3.1 Border effects

Special attention should be devoted to the border effects. Figuratively, this problem occurs only along the borders because the sliding window lies partly outside the image. As a result, there is insufficient data to convolve any border pixel. A choice must be made on how to compute the intensity value of those virtual pixels. Many solutions exist to overcome this problem but none is completely satisfactory [Levine, 1985]. Linear interpolation and image mirroring along the axes yield good results but involve computing overhead difficult to implement in hardware.

The solution retained here is simple and consists in enlarging the image along the borders with zero-valued pixels. Since the center of the sliding window is initially positioned on the upper left corner of the image, it can be seen that a frame

of four zero-valued pixels has to be inserted along the four sides of the image to fill the empty slots under the window. This way, the convolved image will be of the same size as the source image and no image shift will occur. Image shift should be avoided especially if the same image is to be convolved more than once because a cumulative shift would possibly cause partial loss of information

In the case where the zero-valued pixel solution would yield unsatisfactory results, it is still possible to extend the image with interpolated or mirrored values in software before transferring the image to the convolver. As a result, extra processing would have to be done on the host computer both before and after the convolution for inserting and deleting the border. The new size of the image would also have to be given to the convolution processor at initialization.

3.3.2 Up-sampling

The DMC may also be used for increasing the sampling rate of an image or a signal for interpolation. This is easily achieved by inserting one or three zero-valued samples between each sample of the original data depending on the up-sampling mode (2x or 4x). The new sequence is then pushed to the array whose coefficients perform interpolation on the up-sampled image. This feature proves to be useful since many applications of digital image and signal processing necessitate a change in the sampling rate of a digital signal.

3.4 The Converters

The systolic processor can only operate on double precision floating-point data. Yet, most source data is only available in integer formats. For this reason, an input converter transforms the incoming integer data stream into a floating-point data stream which is sent directly to the delay memory circuit. Similarly, at the output,

a floating-point to integer converter restores the data for displaying results on a monitor (CRT).

3.4.1 Overview of the IEEE Floating-Point Standard

In the past, manufacturers used different proprietary formats to store the numbers and did not always return the correctly rounded results of common operations [Ferguson, 1991]. As the use of floating-point arithmetic increased, the need for a standard representation became necessary. In 1985, an IEEE working group presented the IEEE 754 standard whose goal is to improve software and hardware portability. The standard describes:

- The floating-point format (single and double precision).
- The combination (rounding) of floating-point through common operations such as addition, multiplication and division.
- The behavior under error conditions (division by zero, overflow...).

Since the on-board converters comply with this standard, an overview of the double precision floating-point representation is presented. For complete details about the IEEE 754, the reader is referred to [IEE, 1985].

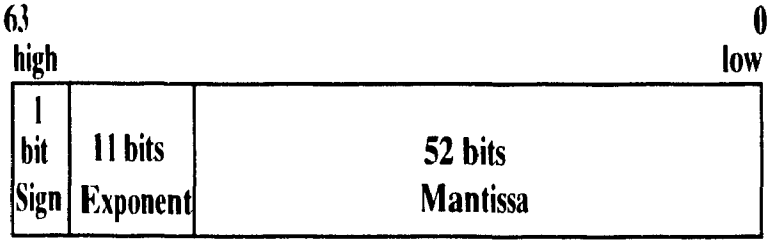


Figure 3.5: Double precision floating-point representation

As Figure 3.5 shows, a double precision number is 64 bits long; one bit for the sign (0 = positive, 1 = negative), 11 bits for the exponent and 52 bits for the

mantissa. In order to ensure a unique internal representation for each FP number, the exponent is adjusted so that the mantissa has an implied 1 before its binary point. This *normalization* means that the 1 in front of the binary point need not be stored since it is always present. So, although there are 52 bits in the mantissa, a 53-bit precision is provided.

The exponent value is represented using the *excess 1023 notation* which gives values in the range -1023 to 1024. In other words, the decimal value (v) of a number is:

$$v = (-1)^s * (1.mant) * 2^{(exp-1023)} \qquad (3.2)$$

where s is the sign, $mant$ and exp are the decimal equivalent of the mantissa and the exponent respectively. In the excess 1023 notation, an exponent with the maximum value represents infinity (∞) only if the mantissa is zero otherwise it is a NaN (not a number). An exponent with the minimum value represents a zero if the mantissa is null, otherwise it indicates an underflow.

3.4.2 Input Converter

The input converter [Drolet *et al.*, 1990] is the first processing stage of the pipeline. It reads the integer data from the input FIFO buffer and requires 16 clock cycles to complete a conversion. Thus, at the end of each pipeline cycle a new floating-point number is ready to be sent to the delay memory circuit. The current implementation of the converter operates in three modes. In the first two modes, 8-bit and 16-bit integers can be converted to 64-bit floating-point numbers. Yet, these conversions only yield positive numbers since it is assumed that the integers represent pixel intensity values which are positive. In the *transparent* mode, the input converter expects floating-point samples which it passes on to the delay memory circuit without conversion.

3.4.3 Output Converter

The data coming out of the systolic array may be processed by the output converter which maps the floating-point results back into integer format. Due to the dynamic range disparity between the two formats, it is impossible to make a one-to-one mapping. For this reason, an interval of floating-point values must be mapped to a single integer by means of a look-up table which supports both linear and non-linear mappings. During conversion, a binary search into the look-up table will iteratively find the integer which maps to a floating-point interval. An appropriate implementation of the algorithm will allow completion of one conversion every pipeline cycle. The converted pixels are stored temporarily into an output FIFO buffer which requests DMA transfers when half full.

In any case when multiple convolutions are to be performed on the same image, the output converter may be bypassed to allow storage of floating-point intermediate results. This mode of operation prevents errors due to repeated conversions between numeric formats which would otherwise result. However, it also generates more memory traffic due to its increased storage requirements.

3.5 The VMEbus Interface

A system such as the Sensor Computing Environment (SCE) of the McGill computer vision laboratory consists of a set of subsystems that need to be interfaced to each other. For instance, the SCE includes a number of general-purpose computers as well as a laser rangefinder, a variable-scan camera and eventually, the convolution processor. Obviously, there is a need for communication between the I/O devices and CPUs. But the devices should also be able to access memory if the information is to be shared among the subsystems.

The SCE relies on the VMEbus to establish a shared communication link between

the subsystems. This standard bus organization offers low cost and versatility that allows a breed of new devices to be easily added to the system. The VMEbus specification manual [Mot, 1982] gives the characteristics of modules and protocols which define the interaction between the bus and devices interfaced to it. Figure 3.6 illustrates how the devices connect to the VMEbus. The signals which make up the bus are divided into four categories. *The data transfer bus (DTB)* contains 32 address lines, 32 data lines as well as their associated control signals. *The priority interrupt (PI)* lines allow devices to interrupt normal bus activity and can be prioritized into a maximum of 7 levels. *The DTB arbitration (DTBA)* consists of signals which enable different bus masters to take control of the bus in turn. Finally, *the utility (UTIL)* lines provide for system initialization and failure detection.

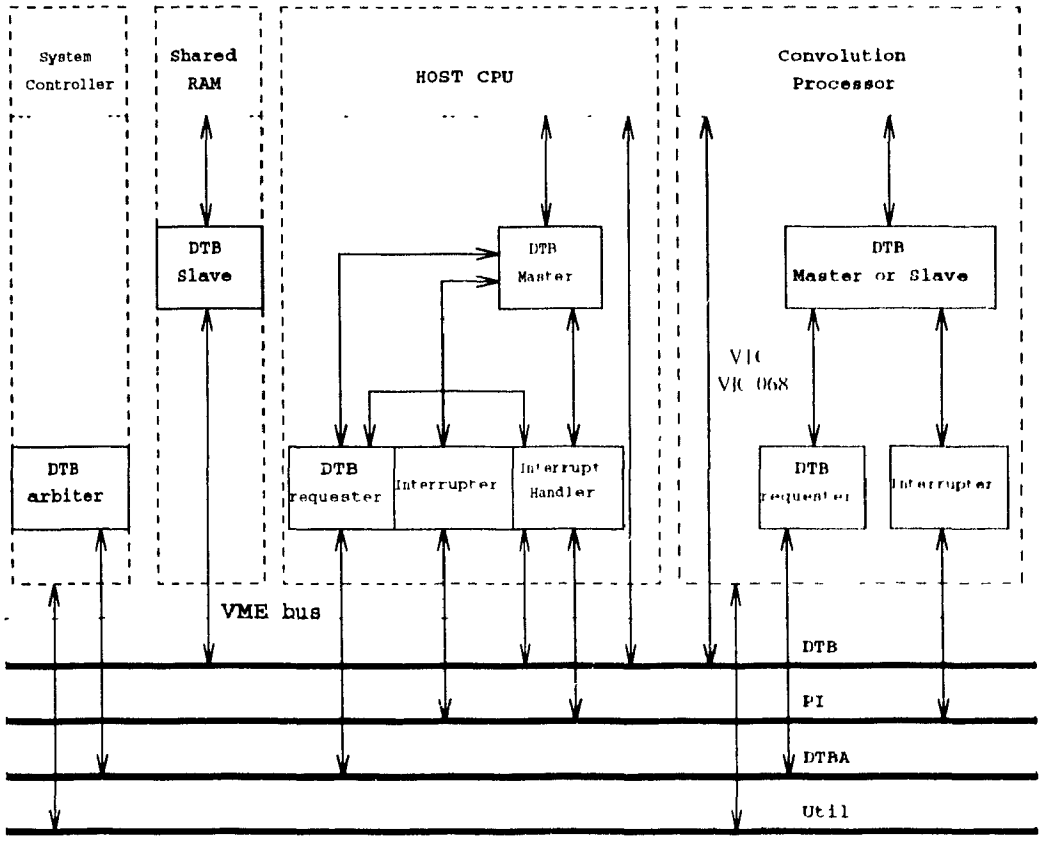


Figure 3.6: Subsystems and their interface to the VMEbus

In order to take advantage of the features available with the VMEbus, it was decided to design an "intelligent" interface which would link the convolution board

to the VMEbus. The advantages of an intelligent interface are numerous:

- Frees the host CPU during convolution operation.
- Added flexibility through software.
- Partial control of the convolution processor.
- Multiple channel DMA controller (fully programmable).

An intelligent interface unburdens the host CPU. This is highly desirable in a multitasking environment especially when time-critical operations are pending. Moreover, a programmable interface adds more flexibility and provides for future extensions. In addition to taking care of the I/O operations, it properly initializes the processor and handles the pipeline fill and flush delays, respectively at the beginning and at the end of a convolution operation. The next section presents an overview of the interface.

3.5.1 The DMA Engine

The main task of the interface is to move information between the convolution processor and memory. Most of these I/O operations involve block transfers which are best handled with direct memory accesses. The solution that was adopted, as shown in Figures 3.1 and 3.6, is the use of an embedded Motorola 68020 microprocessor and a VTC VIC-068 VMEbus interface controller. The VIC, whose operation is set up by the 68020, implements the necessary functional modules to drive directly both the VMEbus lines and the 68020 local bus. This translates to the following capabilities:

- Bus master: The VIC can request control of the 68020 local bus and the VMEbus to perform DMA transfers.

- Bus slave: The host computer may access on-board resources of the convolution processor.
- VMEbus interrupter: The VIC generates an interrupt to indicate the end of a convolution operation.
- Interrupt controller for the local CPU: The VIC prioritizes on-board interrupt requests and forces the local CPU to initiate DMA transfers.
- Inter-processor communication: The host CPU and the local CPU may communicate through special registers mapped to the VMEbus address space.

Upon power up, the 68020 processor begins executing the code stored in the local ROM. Initialization of the VIC registers follows and a program is downloaded into the local 32K RAM. At this point, the 68020 starts running the program and awaits further instructions from the host CPU. A host command might request a new set of coefficients to be downloaded and a new set of values for the look-up table of the output converter might also be desired. It is even possible to reconfigure the on-board programmable gate arrays by downloading the appropriate configuration files from the host memory. But most importantly, the host, through the inter-processor communication registers, can initiate a convolution operation. It does this by sending a message which contains the starting address of the source image or signal, its length, the address where the results are to be stored, the input and output data formats and the up-sampling mode.

As convolution is being executed, the 68020 enters a loop which sets up the DMA registers of the VIC. Although there are no segments in the 68020 architecture, the VIC transfers blocks of only 256 bytes and therefore must have the content of its DMA registers incremented repeatedly for larger block moves. The VIC maximizes bus throughput by using the burst mode and by taking advantage of the full width of the data bus.

FIFO buffers (see Figure 3.1) permit asynchronous communication between the host memory and the convolver. Thus, the DMA can use the bus sporadically at its highest rate while the convolver works at a much lower but constant rate. Yet, on average, both devices operate at the same speed. For instance, when using 8-bit integers both at the input and output (assuming a 12.5 MHz clock), an average of 1.56 Mbytes/second have to be transferred over the VMEbus. On the other hand, if 64-bit floating-point numbers are desired both at the input and output, then a 12.5 Mbytes/second bandwidth is required. For further details on the DMA engine, the reader is referred to [Panisset *et al.*, 1990].

Chapter 4 The Design of Auxiliary Subsystems

Although the floating-point systolic array represents the core of the convolution system, a few auxiliary units provide indispensable help in converting data to the appropriate format and in feeding the array with the proper data sequence. This chapter addresses the design of these subsystems namely, an input converter, a delay memory circuit and an output converter. The architecture of each subsystem is presented in greater detail. All along the design phase, different alternatives have been examined and resourceful solutions have been adopted to meet some changing requirements.

4.1 Input Converter

The convolution processor only works with double precision floating-point data. However, most data acquisition systems sample and quantize source information into an 8- or 16-bit integer format. For this reason, an input converter has been designed to convert data from integer to double precision floating-point format. In addition, a transparent configuration allows data already in floating-point format to pass through without format conversion. The pipeline architecture enables the converter to complete a conversion every pipeline cycle (16 clock pulses) so as to match the processing rate of the systolic convolution cells.

The block diagram of Figure 4.1 illustrates the configuration which achieves the format conversion. A controller assumes the proper coordination of three datapath blocks which perform multiplexing, conversion and serialization of the data passing through. The controller is also responsible for reading the input FIFOs in the proper sequence and for generating two signals which affect the behavior of

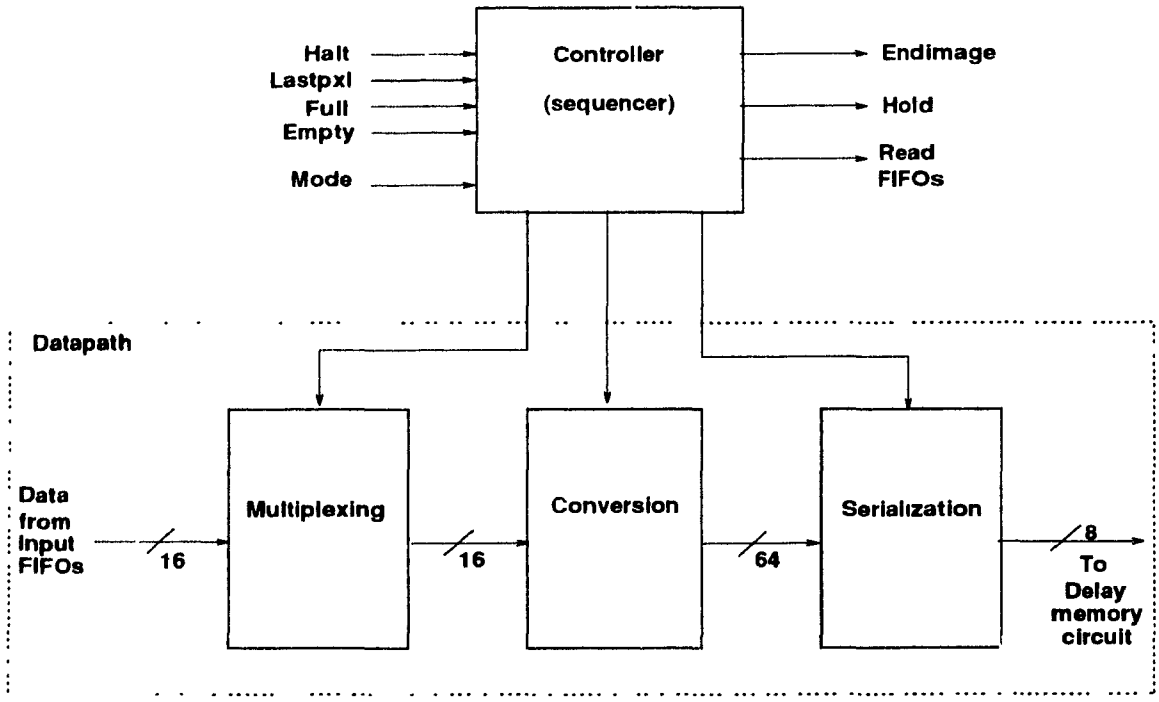


Figure 4.1: Input converter (Conversion configuration)

other subsystems, namely:

- *ENDIMAGE* . This signal is sent to the delay memory circuit and indicates that the last sample or pixel has been converted. This signal is active only if the input signal LASTPXL is already high and both the input FIFO buffer and the pipeline of the converter are empty.

- *HOLD* : This signal is active when the input FIFO buffer is empty or when the output FIFO buffer is full. In these cases, the system pipeline will be temporarily halted until the DMA controller services the FIFO buffers. In this way, no garbage will enter the pipeline and no results will be lost because of a full FIFO buffer.

The operating mode is set with the input signal *MODE*. When low, input data are assumed to be 8-bit integers. When high, input data should be 16-bit integers. In both modes, integers are converted to the floating-point format and fed to the delay memory circuit in chunks of 8 bits starting with the least significant byte of the

floating-point number. The input signal *HALT* is generated by the delay memory circuit and tells the converter to stop its operation during the next pipeline cycle. Typically, this will occur only when the delay memory circuit inserts zero-valued pixels in the data stream.

4.1.1 FIFO Organization and Multiplexing

The input buffer, as shown in Figure 4.2, consists of four parallel 8-bit FIFOs having a storage capacity of 2K bytes each. Its structure allows 32-bit data transfers which take advantage of the full bandwidth available on the VMEbus. Built around a Motorola processor, the DMA engine complies with the *Big Endian* [1] convention for ordering bytes within a word. In accordance with this model, FIFOs 0 and 2 are each mapped to an even address and are therefore connected respectively to bits 31:24 and 15:8 of the input data bus. Similarly, FIFOs 1 and 3 are mapped to odd addresses and are therefore connected to bits 23:16 and 7:0. Naturally, these four address locations are consecutive, FIFO 0 having the lowest address and FIFO 3 having the highest.

In mode 1, 16-bit integers are converted. Thus, at the beginning of each pipeline cycle a 16-bit integer (a word) is read alternately from FIFOs 0 and 1 and FIFOs 2 and 3. In Figure 4.2, it can be seen that multiplexer A forwards bits 15:8 (SEL0 = 1) of the output data bus to the input converter while multiplexer B forwards the 8 least significant bits (SEL1 = 0). In mode 0, 8-bit integers are converted Unlike the previous mode, the FIFOs have to be read individually since each holds an 8-bit sample value to be converted. Now, multiplexer A routes zero-valued bits to the upper half input of the converter (SEL0 = 0). Multiplexer B selects in turn the upper and lower 8 bits of the output data bus so that the bits corresponding to the FIFO being read can be forwarded to the lower half input of the converter.

[1] In Big Endian addressing, the address of a word is the address of the most significant byte within that word.

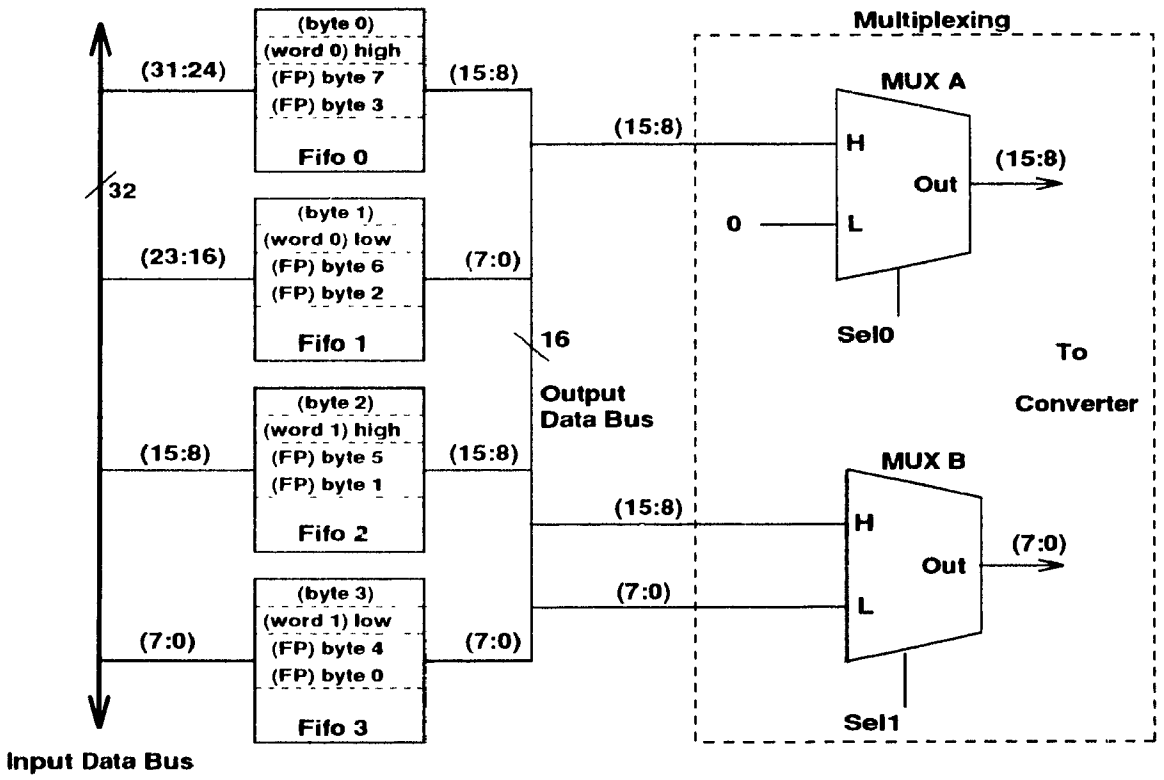


Figure 4.2: Input FIFO organization and multiplexing stage

4.1.2 The Conversion

The schematic of the converter block is included in sheet 5/6 of Appendix A for reference. This block is composed of a 16-bit parallel load shift register, a 4-bit counter and some control logic. A conversion starts on cycle 0 when an integer is latched into the shift register. The binary point is always positioned to the right of the most significant bit (q15) of the register. This means that at the end of the conversion, the remaining 15 bits will map to bits 37 to 51 of the mantissa (these are the most significant bits of the mantissa). Consequently, bits 0 to 36 of the mantissa are always set to zero. Note also that the converter assumes positive values and always sets the sign bit to 0.

Initially, on clock pulse 0, the 16-bit register is loaded with a new integer. At the same time, the 4-bit counter is preset with value 14. From Table 4.1, it can be seen that this value matches the lower four bits of the excess-1023 exponent (see

Section 3.4.1) because the corresponding decimal exponent value at this instant is 15 (recall that the binary point is positioned to the right of the most significant bit of the shift register). Then, for each subsequent clock cycle, the number is shifted left by one and the counter (exponent) is decremented until a 1 appears in the most significant bit (q15) of the register. At this point, the number is *normalized* and both the register and the counter hold their content until the end of the pipeline cycle. The lower four bits of the exponent are given by the output of the counter while bits 4 to 9 are represented by the carry of the counter. As for bit 10, it is found by a logic combination of the carry and q15.

In the last two lines of Table 4.1, it is shown that, if the register needs to be shifted 15 times, then the decimal exponent is necessarily 0, which means that the number being converted is either 1 or 0. If it is 0, at clock cycle 14, both q14 and q15 are low and the counter is stopped (its content is 0000). On clock cycle 15, the remaining bits of the exponent are set to zero because the carry and q15 are low. On the other hand, if the number is 1, then on clock cycle 14, q14 is 1 and the counter will be decremented one more time. Therefore, on pulse 15 the content of the counter changes to 1111, and since signal P0 is asserted (P0 is always active on pulse 15), the carry and the exponent bits 4 to 9 are set high.

4.1.3 The Pipelined Output

The purpose of the pipeline output is to serialize the converted number in such a way that it can be forwarded to the delay memory circuit over the next pipeline cycle. The circuit (see schematic in sheet 6/6 of Appendix A) supplies 8 bits at a time every other clock cycle starting with the lowest significant byte on clock cycle 0. At pulse 0, four registers load the 32 most significant bits of the floating-point number (the lower half bits are always zero and they do not need to be latched). Then, signal CESUP8 disables the registers for the next 6 clock pulses so that no parallel shift occurs during that period. So, on clock cycles 0, 2, 4 and 6 , only

Decimal exponent	Excess-1023 notation exponent (binary)			clock cycle number
	10	9 8 7 6 5 4	3 2 1 0	
15	1	0 0 0 0 0 0	1 1 1 0	0
14	1	0 0 0 0 0 0	1 1 0 1	1
13	1	0 0 0 0 0 0	1 1 0 0	2
12	1	0 0 0 0 0 0	1 0 1 1	3
11	1	0 0 0 0 0 0	1 0 1 0	4
10	1	0 0 0 0 0 0	1 0 0 1	5
9	1	0 0 0 0 0 0	1 0 0 0	6
8	1	0 0 0 0 0 0	0 1 1 1	7
7	1	0 0 0 0 0 0	0 1 1 0	8
6	1	0 0 0 0 0 0	0 1 0 1	9
5	1	0 0 0 0 0 0	0 1 0 0	10
4	1	0 0 0 0 0 0	0 0 1 1	11
3	1	0 0 0 0 0 0	0 0 1 0	12
2	1	0 0 0 0 0 0	0 0 0 1	13
1	1	0 0 0 0 0 0	0 0 0 0	14
0	0	1 1 1 1 1 1	1 1 1 1	15
0	0	0 0 0 0 0 0	0 0 0 0	15

Table 4.1: Possible exponent values

zeros are pushed to the delay memory circuit. On the seventh rise of the clock, the registers are allowed to shift in parallel. Then, on clock cycles 8, 10, 12 and 14, the 32 highest bits are routed to the delay memory circuit 8 bits at a time.

4.1.4 The Transparent Configuration

Sometimes the pixels of an image or the samples of a signal are already available in the double precision floating-point format. In this case, no conversion is required, however the 8 bytes which make up a floating-point value have to be reordered because the systolic cells process the lower bits of their operands first. Figure 4.2 shows how different format values are stored in the FIFO buffers. Since a floating-point (FP) number is 64-bit long, it occupies two rows of the FIFOs where bytes

7, 6, 5 and 4 are first stored in FIFOs 0, 1, 2 and 3 respectively, and the remaining bytes fill up a second row. As a result, the lower half portion of the number is not readily available for reading. Thus, the upper 32 bits must be read out first and temporarily stored into registers. A simple circuit presented in Appendix A (config 64 -> 64) has been designed to reorder and serialize the data before sending the data to the delay memory circuit.

The behavior of this circuit is best shown with the timing diagram of Figure 4.4 showing two cycles. On the first pipeline cycle, the number is read from the FIFO buffers in four chunks of 16 bits and then stored in parallel shift registers where the bytes are reorganized. On the next pipeline cycle, the bytes are shifted out (serialized) from the registers and sent to the delay memory circuit every other clock cycle.

For example, let us assume that the 64-bit hexadecimal number *FEDC BA98 7654 3210* is stored in the FIFOs as shown in Figure 4.3. First, word BA98 is read out of FIFOs 2 and 3 and latched into a 16-bit register on clock cycle 0. The remaining bytes of the number will be latched on cycles 4, 8 and 12. Once latched, the bytes circulate through a set of 8-bit registers (see schematics in Appendix A) until the end of the first pipeline cycle. Next, a multiplexer selects node *LEAST* from which the lower half bytes will be shifted out during clock cycles 14, 0, 2 and 4. The upper half follows when node *MOST* is selected during cycles 6, 8, 10 and 12. In this way, the bytes of the floating-point number appear at node *FPOUT* alternately, starting with byte *10* hex and ending with byte *FE* hex.

4.2 Delay Memory Circuit

Conceptually, the convolution operation can be visualized as a window which moves over a signal or an image in a regular scan pattern. At each step, new pixels enter the window, others leave it and computations are performed on the

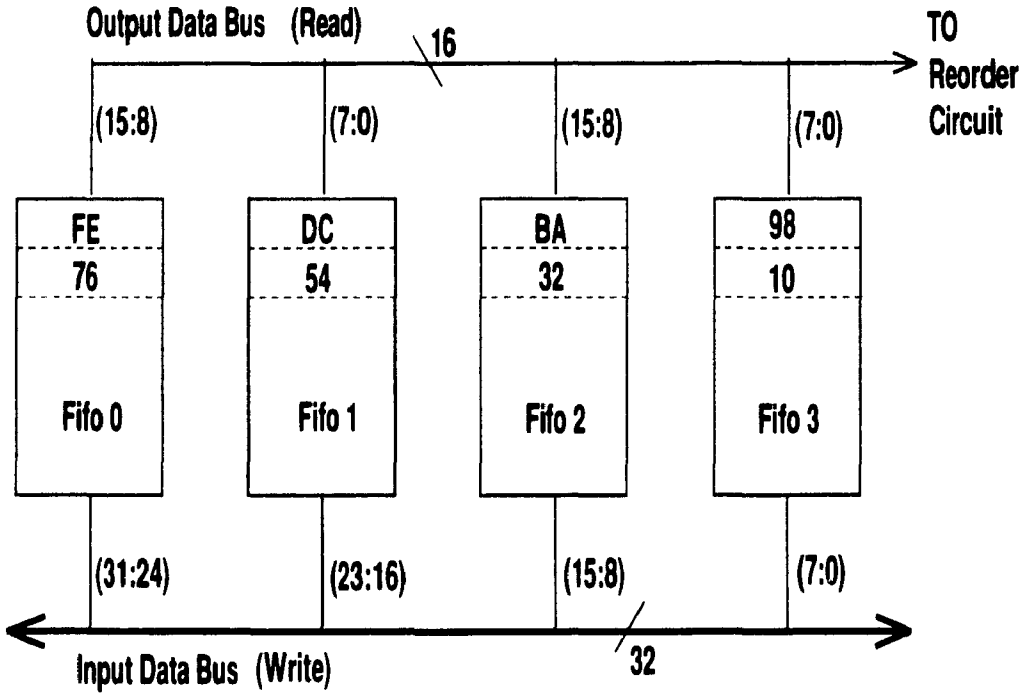


Figure 4.3: Number FEDC BA98 7654 3210 as stored in FIFO buffers

overlapped pixels. In the systolic design, the array is analogous to the sliding window. The delay memory circuit (DMC) provides "virtual mobility" to the systolic array by shifting data through it. The following section describes the design of this auxiliary circuit.

4.2.1 The Specifications

The primary task of the DMC consists in feeding the proper sequence of data to the systolic array so as to keep the processors as busy as possible. However, additional features provide an extension of its scope:

- *1-D or 2-D operation.* The DMC handles both unidimensional signals and images. A control line specifies the mode of operation.
- *High data transfer rate.* In order to facilitate system integration, the DMC design matches the architecture of the systolic array. Therefore, a 64-bit pixel

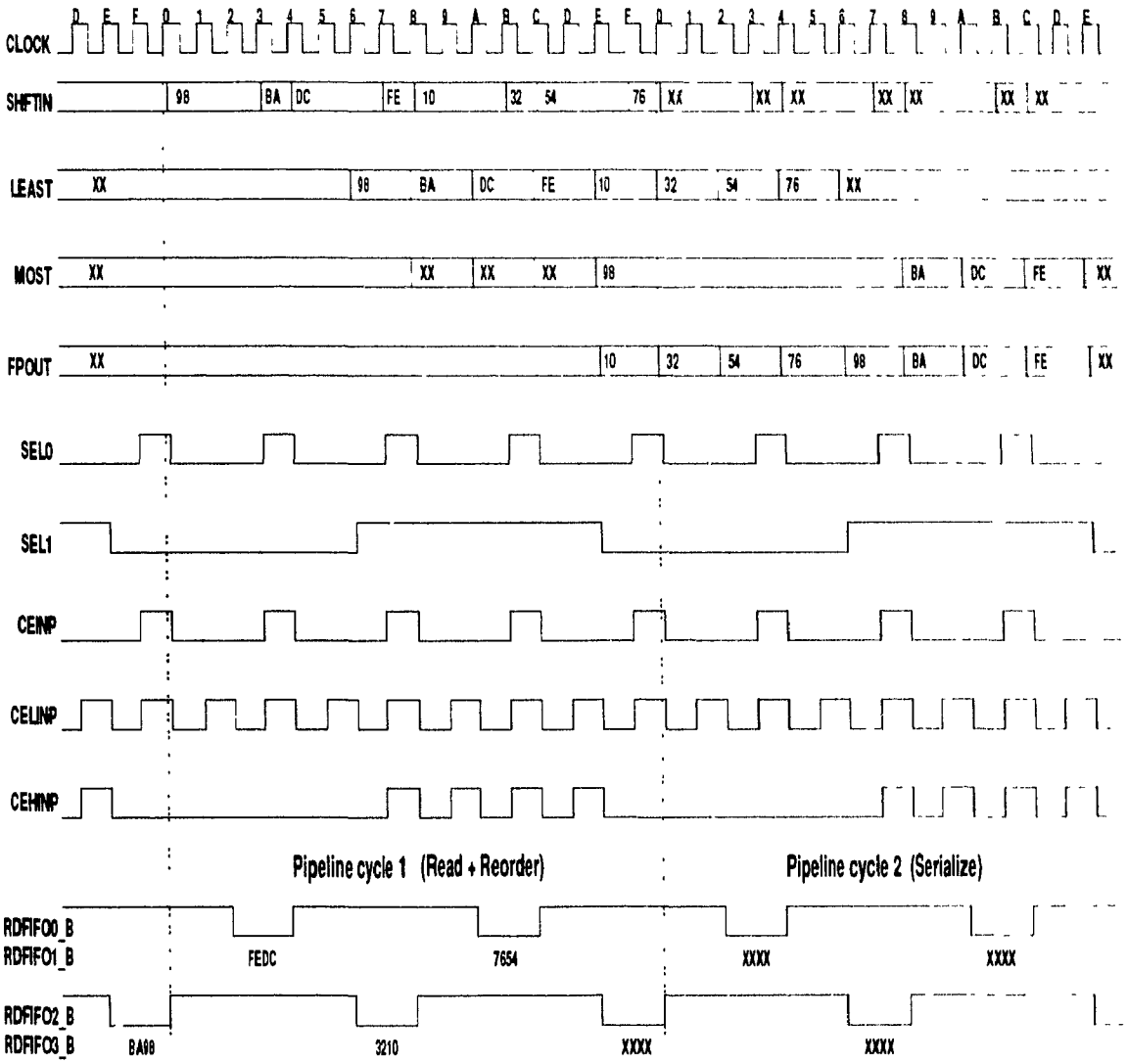


Figure 4.4: Timing diagram of input converter (Transparent configuration)

is serially transferred to the array in one pipeline cycle (16 clock pulses). The DMC keeps up with the demand for new inputs, and thus during each cycle, 9 pixels are sent to the array in parallel.

- *Low VMEbus utilization.* Although this statement seems somewhat contradictory to the previous one, it is possible to conciliate the two by implementing an efficient buffering strategy. The data re-use capability greatly decreases traffic on the VMEbus because a pixel is read only once from host memory even if it is used 9 times.

- *Variable image/signal size.* The convolution processor should not be restricted to operate on a fixed image size. Thus, a broad range of image/signal sizes may be handled by the DMC.

- *Border effects.* The DMC implements a simple strategy to tackle the border effect problem inherent to the convolution of finite signals.

- *Up-sampling.* Depending on the mode of operation, the DMC may alter the original sequence of data to insert 0, 1 or 3 zero-valued pixels between the samples to perform 1, 2 or 4 times up-sampling, respectively. The resultant signal can be interpolated with an appropriate set of coefficients in the systolic array.

4.2.2 Functional Description

Both signals and images require pre-processing in the DMC before being sent to the systolic array. The first alteration consists in adding a frame of 4 zero samples at the beginning and at the end of a signal. Likewise, images are extended along their borders with a frame of 4 zero-valued pixels. This is a simple yet efficient method to overcome the border effect problem since no extra computations are involved. Moreover, the zero-valued frame eliminates image shifting because it enables the convolution window to be centered on every sample of the original image or signal including those along the borders. The second alteration occurs only when operating in up-sampling mode. In this case, the DMC doubles or quadruples the sampling rate of the signal by inserting 1 or 3 zero-valued samples for each sample read from memory. For example, Figure 4.5 shows an original signal and its altered version. Figure 4.6 illustrates an image which has been enlarged with a frame of zero-valued pixels and up-sampled (2x) so as to double both its original width and height. Indeed, with these alterations the image width grows from w to $(2w + 8)$ pixels.

Before processing	P0, P1, P2, P3, ..., Pn
After processing	0, 0, 0, 0, P0, 0, P1, 0, P2, 0, P3, 0, ..., Pn, 0, 0, 0, 0, 0

Figure 4.5: Up-sampled signal (2x) with its border

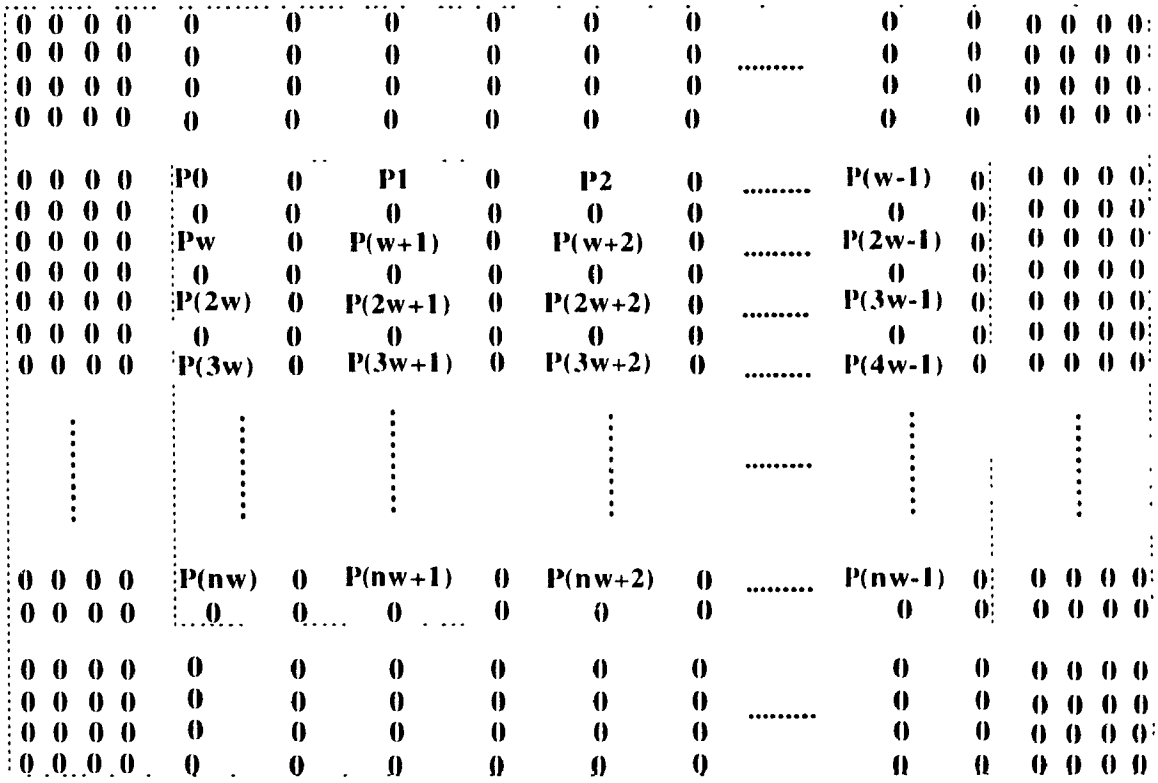


Figure 4.6: Up-sampled image (2x) with its border

The critical task of the DMC is to create the data streams which will feed the systolic array. When operating in 1-D mode, the incoming data stream is sent directly to row 1 of the array. The 8 consecutive rows take their inputs from the rightmost cell of the respective preceding row. On the other hand, the 2-D operation requires the creation of a stream for each row of the array. In this case, proper synchronization is required between the streams. The task is therefore more arduous than the earlier case.

Consider an image of size w. In section 3.2.2, it was shown that a pixel p entering row 1 incurs a delay in traversing the multiple pipeline stages of the systolic processors. Hence, pixel $p + w$ which is directly beneath pixel p cannot enter the array simultaneously. Before it can enter row 2 of the array, this pixel has to wait until p has reached the end of row 1. At this point, the partial sum of row 1 is available and $p + w$ is allowed to enter row 2 simultaneously with this partial sum. In other words, for a given delay d, if p enters the array at time t (in pipeline cycles), then $p + w$ will enter it at $t + d$. As for pixel $p + (n - 1) * w$, it will only enter row n after $(n - 1) * d$ pipeline cycles. As the convolution operation proceeds, each of the 9 rows of the array is fed with a new pixel every pipeline cycle until the last convolved pixel is collected at the output. Consequently, the 9 rows may be considered as the entry point for 9 similar streams of data but delayed from each other by a constant factor. From this, one can infer that a pixel will be sent 9 times to the array; once for each row, starting with row 9 and ending with row 1.

The delay issue as presented above is viewed from a "time" perspective. For practical matters, a "pixel" perspective seems more appropriate to the development of an efficient DMC. The "pixel" perspective aims at finding which pixels enter the array simultaneously. In section 3.2.2 it was found that at time t, pixel $p + (n - 1)(w - d)$ enters row n of the array, where d is the row delay factor and w is the width of the image. From this expression it can be seen that the distance between the pixels to be fed to consecutive rows is constant for a given image width and it is equal to $w - d$.

4.2.3 Alternate Approaches

A software implementation is not only the simplest strategy, but also proves to be the most versatile because it can accommodate different kernel and image sizes. Pre-processing may be performed on the image in memory prior to its transfer to the convolution processor. Initially, nine equidistant pointers are assigned within the image and each of them is associated with a row of the array. At each cycle, the pointers are incremented and a new set of pixels are sent to the array. Unfortunately, this method requires each pixel to be read 9 times from memory. As a result, a large percentage of the VMEbus bandwidth is devoted to the convolution board to the point where other devices may be locked out of the bus for extended periods of time. In addition, the host computer has to translate image data into floating- point format before it gets transferred, otherwise an input converter would be needed for each stream. Consequently the software implementation was rejected since it is not congruent with the idea of an efficient and autonomous convolution processor.

Hardware solutions for the efficient storage and access of parallel data streams have been developed for a wide range of applications. An example of this can be found in [Godon *et al.*, 1990]. In this paper, the authors describe an architecture for the de-interleaving of radar pulse data into separate sequences. Each sequence originates from a single radar device and its content is routed to a particular circular RAM buffer managed by a global memory controller. The features exhibited in the paper have influenced the design of the DMC which presents a problem of similar nature.

Although limited in regard to image size, hardware alternatives contribute to an efficient use of the VMEbus by reading each pixel only once from memory. However, this bandwidth reduction is achieved at the expense of hardware complexity on the board. This is clear since the raster scanned image now reaches the DMC as one data stream. Hence, the DMC has to re-organize it into 9 delayed streams before feeding the systolic array. Several methods exist to rearrange an incoming

data stream. Three of them were investigated for the design of the DMC.

The first method, illustrated in Figure 4.7, uses 8 shift registers through which data circulate Their output connects to a row of the array and to the input of the next buffer. However, no buffer is necessary for row 9 which gets its input directly from the incoming data stream. In accordance with the "time" and "pixel" perspectives that were developed above, a pixel will be first sent to row 9 of the array. But, at the same time, it is copied to the buffer of row 8 where it is stored temporarily for future re-use. Inside the buffer, a pixel is shifted $w-d$ times at which point it reaches the output. In this way, each pixel of the image will eventually circulate through the 8 buffers and be sent 9 times to the array. This method requires a very simple control buffer strategy for shifting data. Unfortunately, large shift registers such as the ones required for this application are not widely used and thus are expensive.

A second method, which became the fist design version of the DMC [Drolet *et al.*, 1991], attempted to go around the above problem by using 8 custom FIFO buffers. In this method, each buffer has a storage capacity equivalent to the image width and is implemented with a RAM addressed in a circular queue fashion. Two pointers, a read and a write pointer, are used to control the 8 buffers which share a common address bus. The distance between the pointers is fixed, with the write pointer lagging the read pointer by an amount equal to the row delay factor. Action takes place in two clock cycles. On the first clock cycles, the read pointer is applied on the address bus, then a set of 8 pixels are fetched from the RAMs and sent to rows 1 to 8 along with the incoming pixel on row 9. On the next cycle, the write pointer drives the bus and pixels are shifted up into the next buffer. Both pointers are then incremented before the next cycle executes. Conceptually, this scheme works well. The major drawback is that a different address drives the memories every clock cycle. When working at high clock rates, it presupposes very fast access memories.

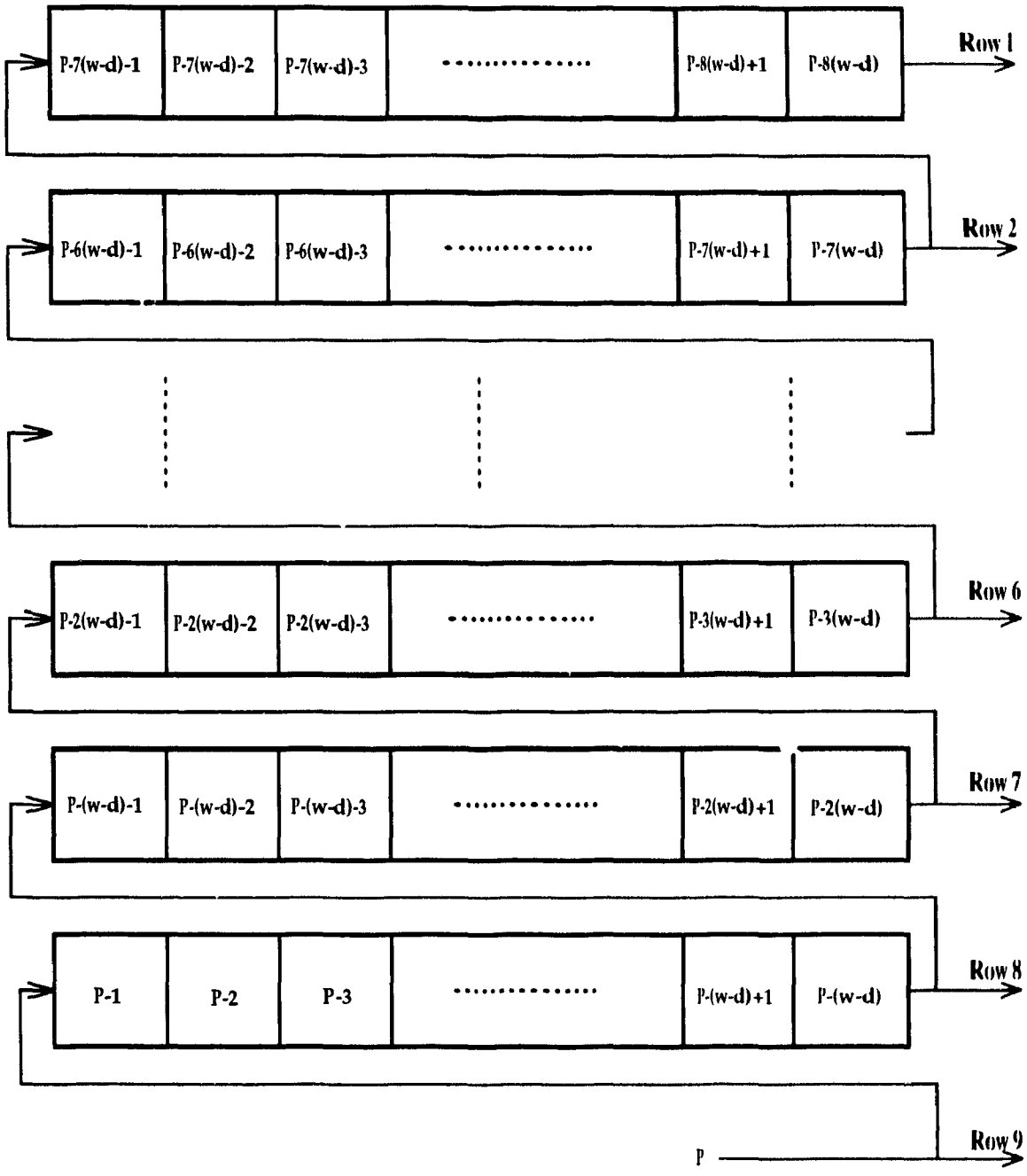


Figure 4.7: Data stream re-organization with shift registers

The latest design version of the DMC takes the best of the two previous methods. Similarly to the two-pointer approach, it makes use of 8 RAM circular buffers but, in light of the shift register approach, their length is limited to $w - d$ pixels. This method behaves exactly like the shift register method except that the data shift is emulated with a single pointer common to all the buffers. The pointer is used to access memories for two consecutive transactions. On the first clock cycle, a read operation is performed. On the next cycle, a write operation (using the same address) shifts up the pixels fed to row n into the buffer associated with row n 1. Although this procedure suffers from the same drawback as the two-pointer technique, it naturally leads to a smaller, more efficient DMC. An in-depth architecture description follows.

4.2.4 The Architecture

The DMC architecture is divided into two blocks as shown in Figure 4.8. The incoming data stream circulates through circular RAM buffers in the datapath block. A controller block, based on three interlocked state machines and a pointer, guides the flow of data in the datapath block. The operation of the DMC is set with the following input lines:

- *DIM :* When DIM = 0, the DMC assumes a 2-D signal (image). When DIM = 1, a 1-D signal is expected.
- *WIDTH :* Specifies the width of the image when operating in 2-D mode. Five lines yield a possibility of 32 different image sizes ranging from 32 to 1024 pixels. The following formula describes the relation between the image width and the WIDTH lines:

$$Image width = (WIDTH * 32) + 32 \qquad (4.1)$$

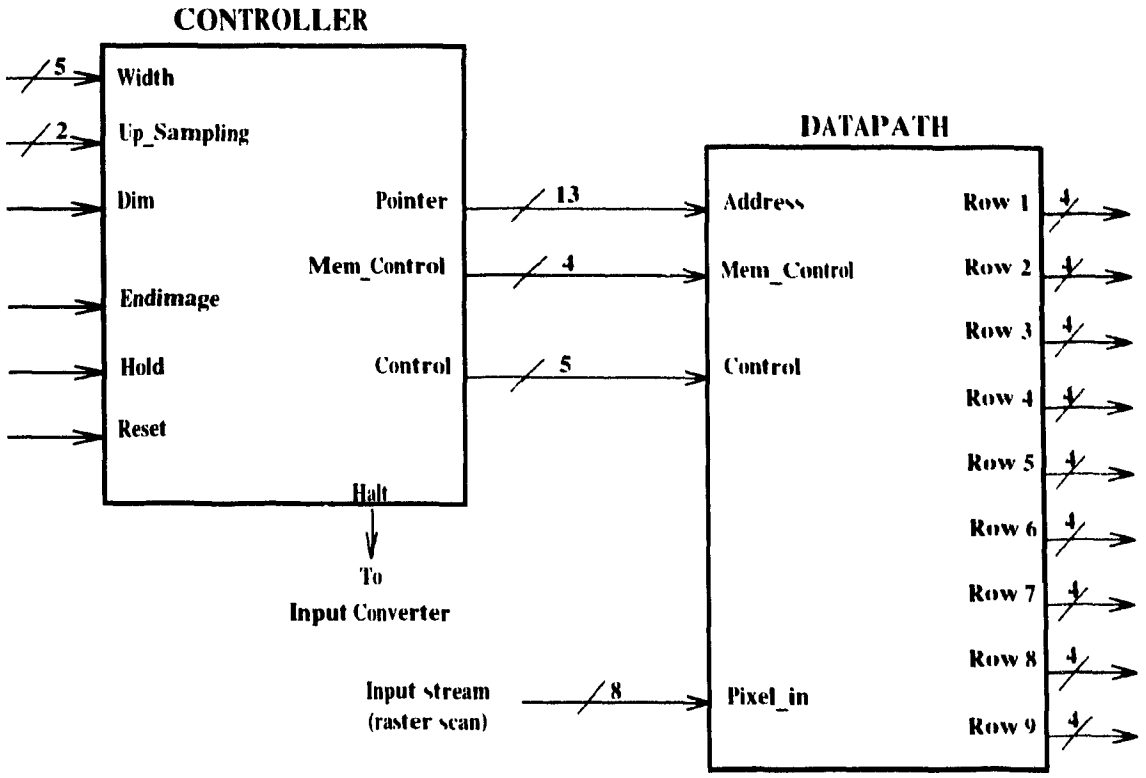


Figure 4.8: Architecture of DMC

Although the maximum width of the image is restricted to 1024 pixels, there is no limitations either for an image height or for the width of a 1-D signal.

- *UP-SAMPLING :* These two lines set the up-sampling mode according to the following table:

S1	*S0*	*Up-sampling mode*
0	0	1x
0	1	2x
1	0	1x
1	1	4x

The following signals provide dynamic communication with other units of the convolution processor:

- *HOLD :* When asserted, the DMC is temporarily stopped.

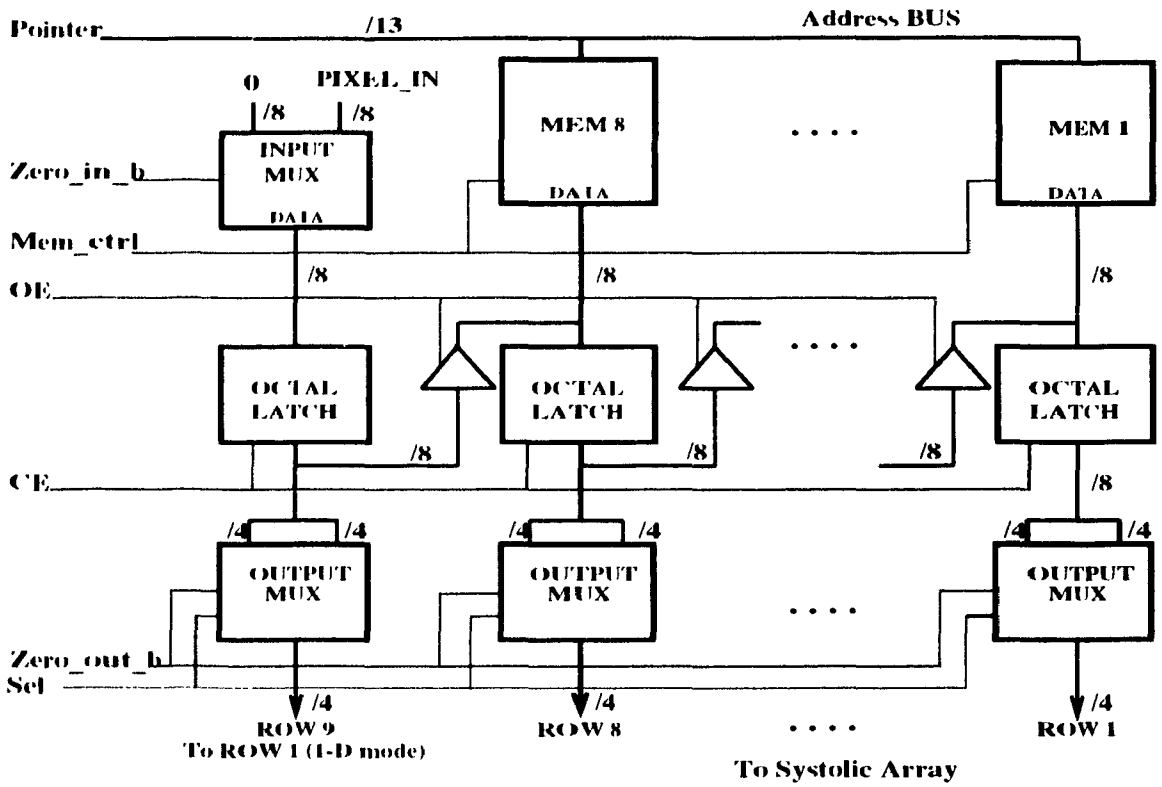


Figure 4.9: Datapath architecture of DMC

- *HALT* : Whenever a zero-valued pixel is to be inserted in the data stream, this signal disables the input converter during the next pipeline cycle.
- *ENDIMAGE* : Indicates that the last sample will be sent to the DMC over the next pipeline cycle.

Datapath Block

Figure 4.9 illustrates a regular datapath architecture which can be extended indefinitely to accommodate larger systolic arrays. Except for row 9 which implements two multiplexers, all rows are similar with one output multiplexer, an octal latch and an 8K RAM buffer. The size of the memory is the primary limiting factor for image width. In this case, an 8K memory buffer can delay 64-bit floating-point pixels by as much as 1024 pipeline cycles before their re-use in the next row.

Similarly to the other subsystems on the board, the DMC transfers floating-point data in one pipeline cycle. Pixels enter the DMC on even clock cycles in chunks of 8 bits starting with the least significant byte in cycle 0 and ending with the most significant byte in cycle 14. However, at the outputs, pixels are sent to the array in chunks of 4 bits. The 16 clock pulses of a pipeline cycle are therefore needed to supply a complete 64-bit pixel.

Multiple control lines guide the behavior of the datapath block.

- *MEM_CTRL* : four lines, $R/\overline{W}$, $\overline{OE}$, $CE2$ and $\overline{CE1}$, control the data transfers to and from the memory buffers. For 1-D operation, $\overline{CE1}$ is set to 0 so as to put memory buffers into power-down mode.
- *ZERO_IN_B* : controls the input multiplexer. When asserted (active low), zero-valued pixels enter the DMC.
- *ZERO_OUT_B* : controls the output multiplexers. When asserted (active low), zero-valued pixels leave the DMC.
- *SEL* : selects one of the two inputs of the output multiplexers. On even clock cycles, the four least significant bits are selected. On odd cycles, the four most significant bits are selected.
- *CE* : enables the octal latches so that they latch on even clock cycles.
- *OE* : enables the tri-state buffers on odd clock cycles so that the previously latched values can be transferred to memory buffers of the next row.

Controller Block

The controller consists of two major blocks as depicted in Figure 4.10. A third block, the pipeline cycle counter, simply generates the 16-pulse sequence (p0,p1,p2 etc.) which is used for synchronization at each pipeline cycle. The pointer block is

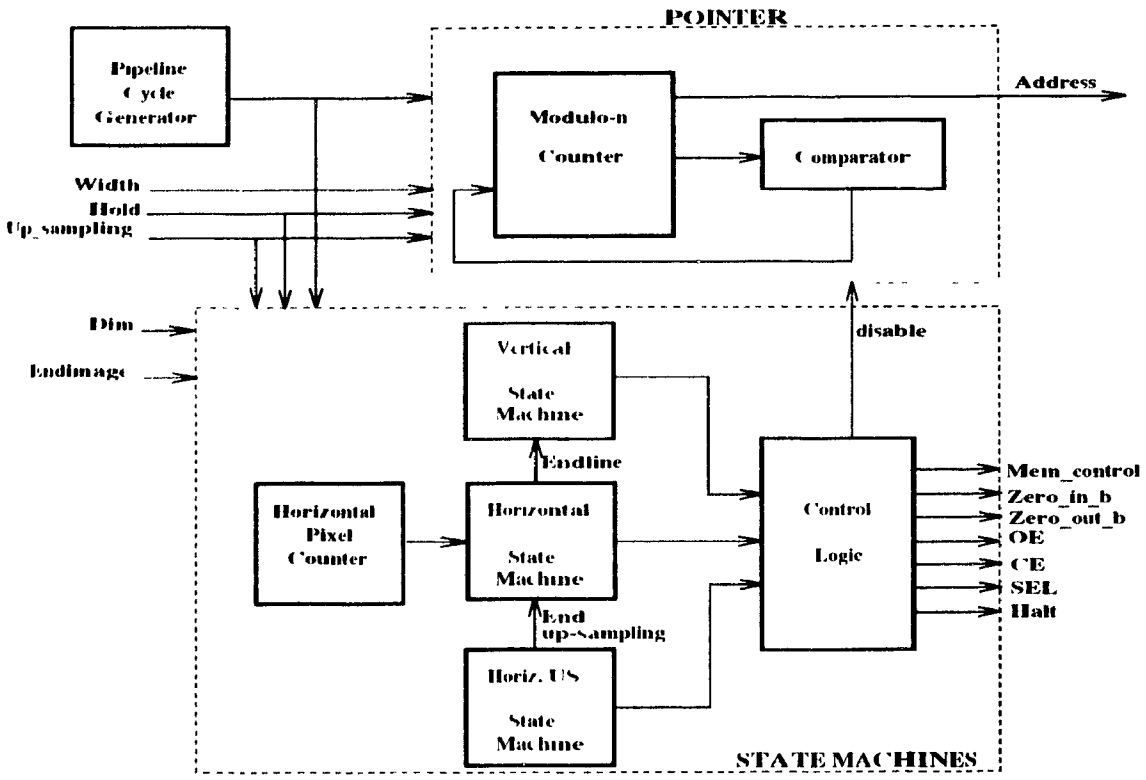


Figure 4.10: Controller architecture of DMC

a 13-bit counter which generates the addresses to access the memory. The counter is programmable "modulo-n" where n is equivalent to the length of the memory buffers $(w-d)$ Finally, three interlocked state machines generate the control signals for the datapath and the address pointer.

Due to the memory buffers, a restricted number of image widths can be accommodated. It seems that this limitation is even more severe when operating in 2x or 4x up-sampling modes because, in these cases, the image width is doubled or quadrupled. However, it would be possible to operate on the same set of image widths as in normal mode (1x) if the storage of the up-sampling zeros could be avoided. In fact, there is no need to store the zeros since their position in the data stream is known in advance. In this design, the output multiplexer of each row can be controlled through the ZERO_OUT_B signal to force zero-valued pixels to be sent to the array and thus avoid unnecessary storage in the buffers. Since the ZERO_OUT_B signal controls the 9 output multiplexers, it must be ensured that

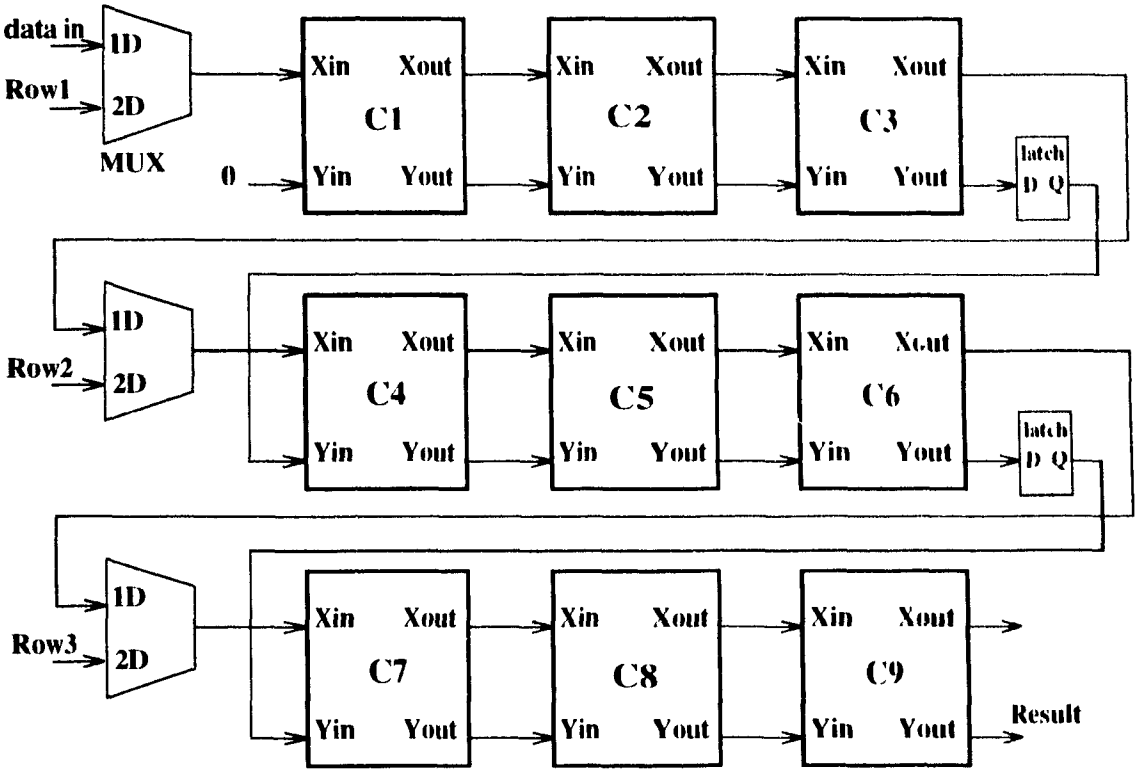


Figure 4.11: Modified systolic array for operation with DMC

all streams will always output zero-valued pixels synchronously. It can be seen that this will only occur if the distance $w - d$ between image pointers is *even*. In this design, w is the extended image width (the up-sampled image width plus the borders) whose value is always even. Unfortunately, the delay factor d which is the delay incurred in traversing the array is an odd value (27). The value $w - d$ will be even only if the delay is also even. For this reason, a set of eight 4-bit registers will be inserted at the partial sum output of the top eight rows of the array so as to obtain a delay of 28 pipeline cycles. Figure 4.11 illustrates this simple modification for a 3×3 systolic array.

Three nested state machines manage the sequence of operations in the DMC. At the bottom level, the *horizontal up-sampling (HUS)* state machine (see Figure 4.12), generates signal ZERO_OUT_B which acts as a strobe for the output multiplexer by forcing a zero at the outputs when activated. When operating in normal mode (1x), the state machine remains in state U1 since no zeros need to be inserted in

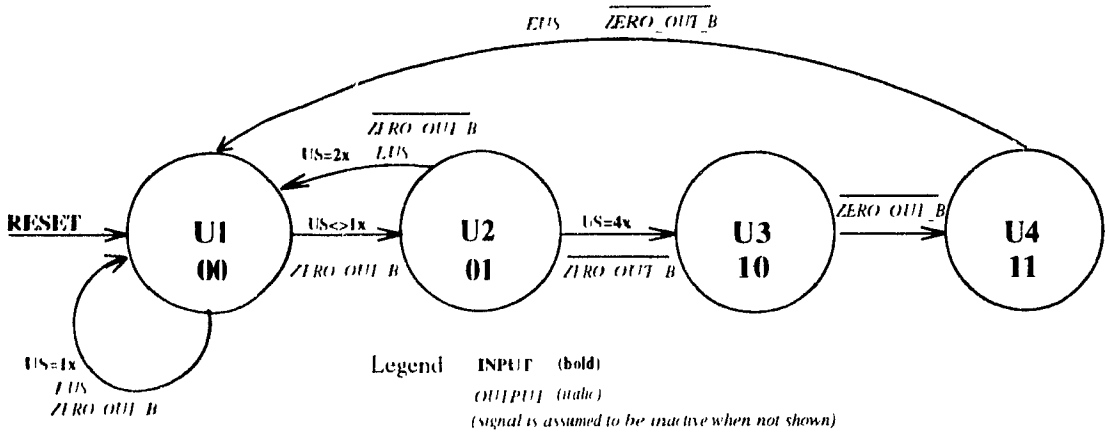


Figure 4.12: Horizontal up-sampling (HUS) state machine

the streams. However in 2x and 4x modes, the state machine changes state every pipeline cycle. In state U1, a normal pixel is read from the input converter. In states U2, U3 and U4, ZERO_OUT_B is asserted (active low) and one or three zeros are inserted in the data streams. The end of a cycle is marked by the assertion of signal EUS (End Up-Sampling) which enables the horizontal state machine to change state if necessary.

The *horizontal* state machine (see Figure 4.13) is responsible for the insertion of the left and right borders of each image line by asserting signal SIH (Select-In-Horizontal). [2] A new line starts with state H1. At this point, at least one zero will be inserted in the incoming data stream, depending on the mode of operation. In 1x mode, 4 zeros will be inserted by going through the sequence H1, H2, H3 and H4. In 2x mode, only 2 zeros will be inserted in sequence H1 and H2 because the HUS state machine already inserts a zero for each pixel in the stream. Similarly, in 4x mode, only one zero is inserted because 3 others will be automatically inserted at the output by the HUS state machine. Once the left border is completed, the state changes to H5 where input pixels enter the DMC. For each pixel inserted, the horizontal counter (see Figure 4.10) is incremented until it reaches the end of the image line. At this point, the right border is ready for insertion and before entering state H1, a flag is toggled. This flag indicates which border (left or right) is being

[2] Signal ZERO_IN_B is the result of an AND operation between SIH and SIV

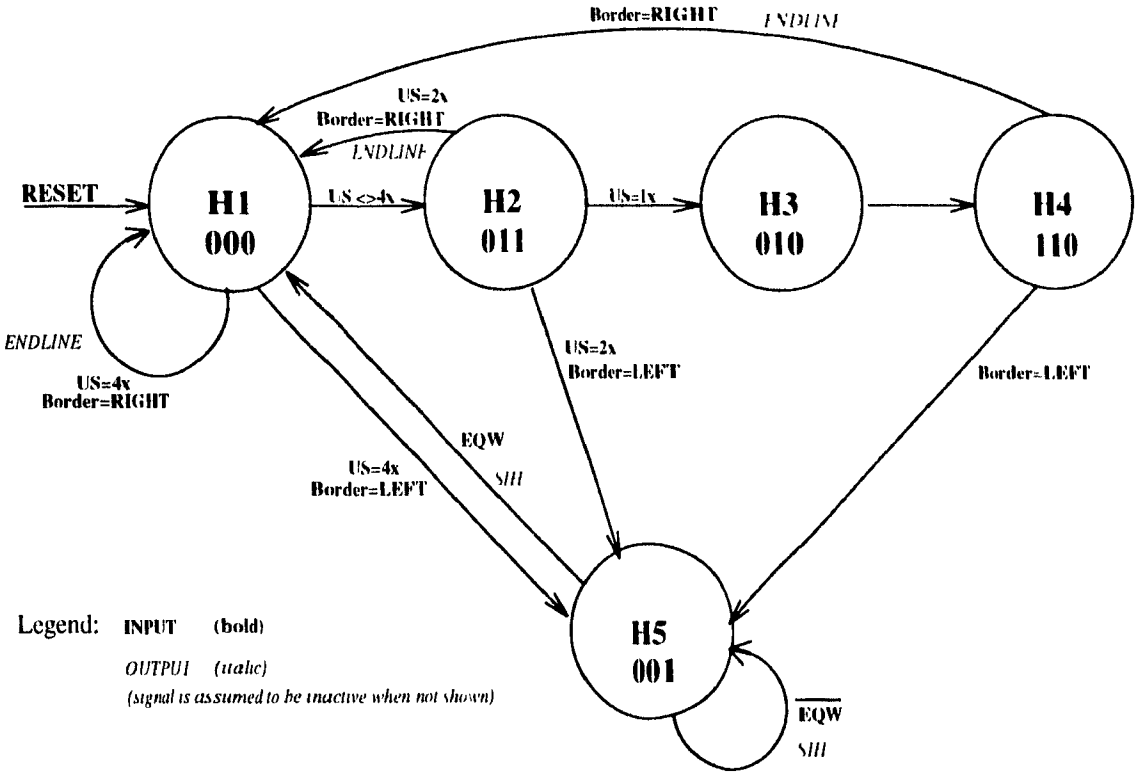


Figure 4.13: Horizontal state machine

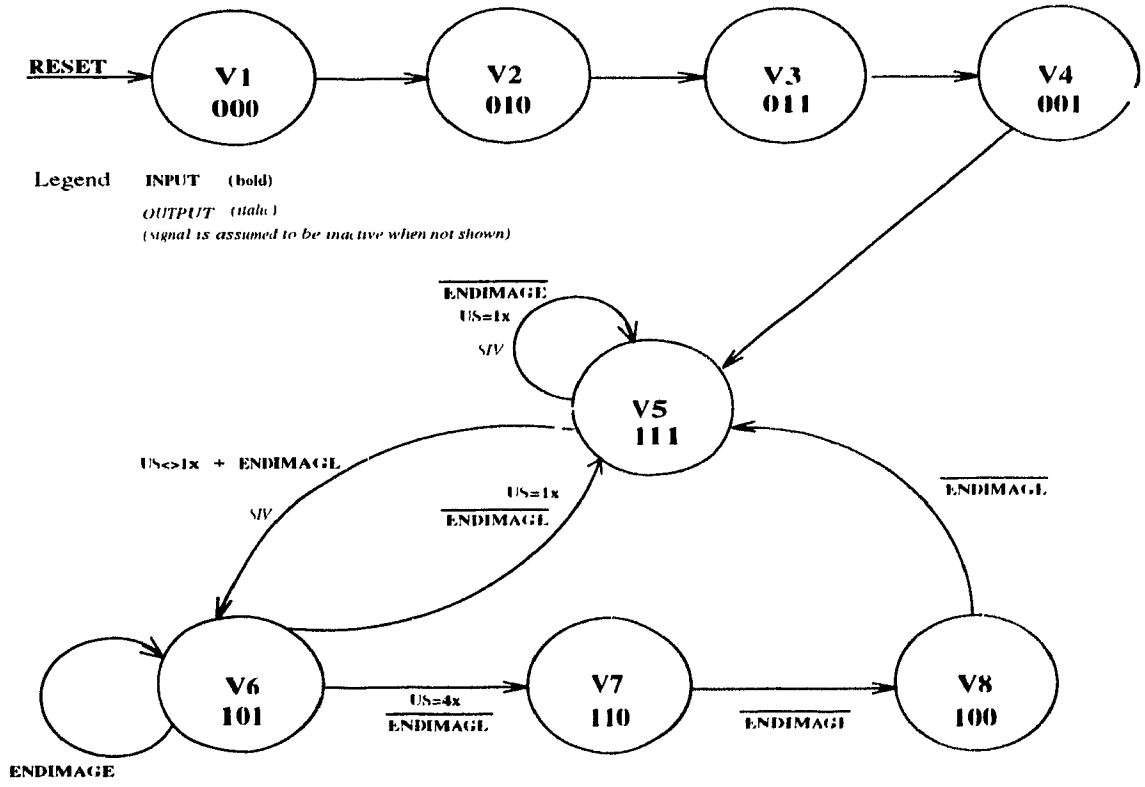


Figure 4.14. Vertical state machine

inserted. At the end of the line, signal ENDLINE is activated and a new cycle may begin. Signal ENDLINE also enables the vertical state machine to change state if necessary.

The *vertical* state machine (see Figure 4.14) is at the top of the nested structure and can only change state if both signals EUS and ENDLINE are set to one. It controls the insertion of the top and bottom borders as well as the insertion of up-sampling rows in the incoming data stream. It should be noted that these all-zero lines are actually injected at the input multiplexer and therefore circulate through the buffers. At the beginning, states V1, V2, V3 and V4 are traversed and 4 null lines are injected into the DMC to create the top border. When V5 is entered, signal SIV (Select-In-Vertical) is activated and normal image lines may be sent to the DMC. In 1x mode, state V5 remains activated until the end of the image (external signal ENDIMAGE) is reached. In 2x or 4x modes, 1 or 3 lines, respectively, have to be inserted for each input line. This is achieved by cycling through V5 and V6 (2x mode) or through V5, V6, V7 and V8 (4x mode). When ENDIMAGE is asserted, state V6 is entered and, at this point, only all-zero lines can be injected into the DMC.

For more details concerning the DMC, the reader is referred to Appendix B and Appendix C which include detailed schematics and PAL files.

4.3 Output Converter

The floating-point results coming out of the systolic array will be forwarded to the last processing stage of the pipeline, namely, the output converter. This converter performs three operations. First, it collects the data at the output of the array. Second, it converts data to integer format and third, it writes the converted values into the output FIFOs. Thereafter, these values may be sent directly to an image buffer for display on a screen.

The output converter may also operate in a *transparent* mode. In this case, no conversion takes place to allow the storage of full precision results. Indeed, this is the favored mode of operation when performing multiple passes on an image to avoid round-off errors.

4.3.1 The Pipelined Input

No conversion can be performed until a full floating-point value has been passed to the output converter. Since 16 clock cycles are needed to send a single value, a pipeline stage has been designed to store the 24 most significant bits of the floating-point value that are needed for the conversion process. As illustrated in Figure 4.15, the stage consists of six 4-bit shift registers and a 24-bit register which latches the floating-point result at the rising edge of the first clock pulse (P0) of each pipeline cycle. The full content of this register is then forwarded to the conversion stage.

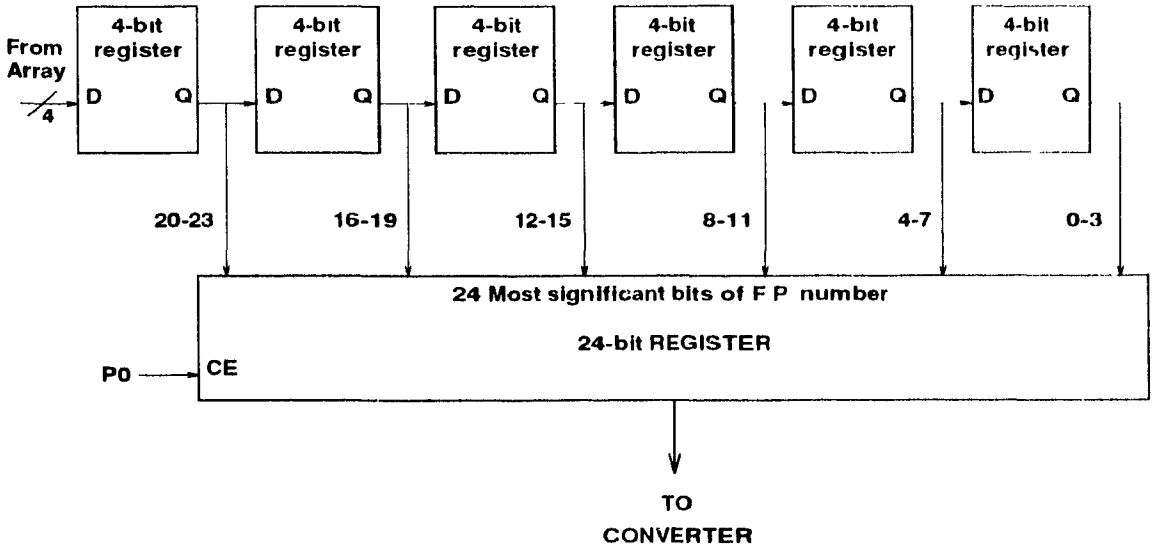


Figure 4.15: Pipelined input to output converter

4.3.2 The Conversion

The conversion of floating-point results into integer format is achieved by means of a binary search in a look-up table. A mapping is performed from a large set of 2^{24} possible values to a smaller set of 2^8 integers. Clearly, this mapping results in a non negligible loss of precision. However, if the range of the floating-point results is known or if the values are distributed in clusters, an optimal precision can be maintained by taking advantage of the non-linear mapping capability of the look-up table. This is accomplished by partitioning the input dynamic range into non-uniform intervals which are loaded in ascending order into the look-up table prior to a convolution operation.

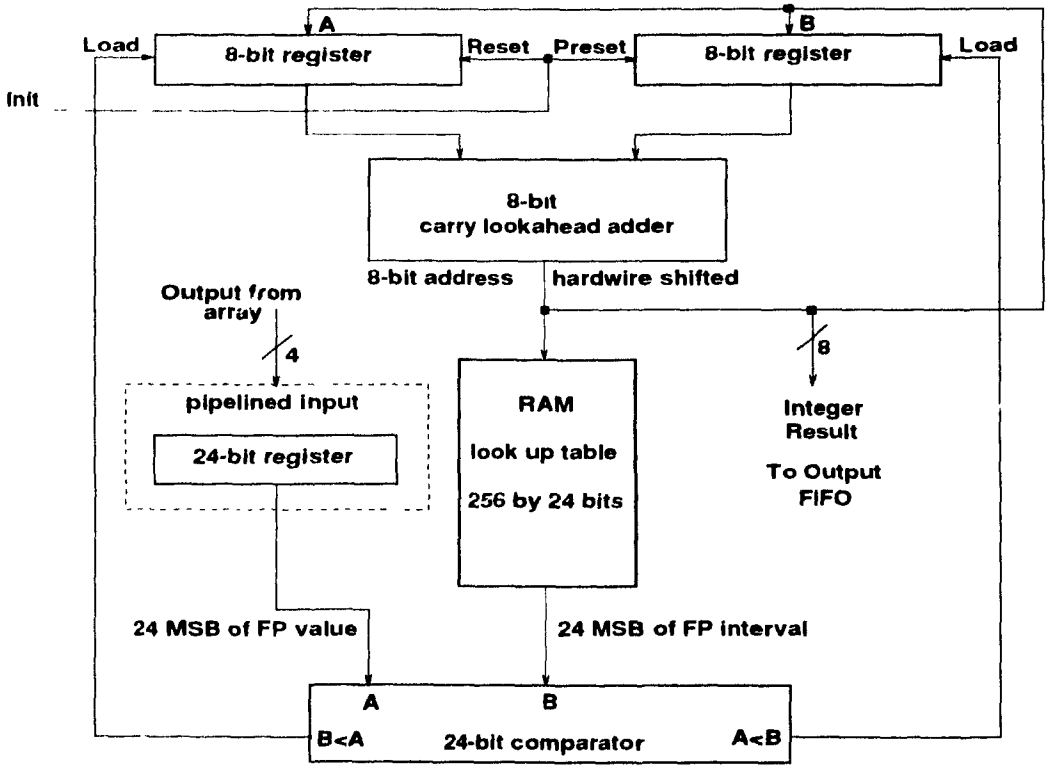


Figure 4.16: Output converter architecture

Figure 4.16 shows a block diagram of the converter. On the first clock pulse of the cycle, the 8-bit register A is reset and the 8-bit register B is preset with the value 255. The content of these two registers are added in an 8-bit carry lookahead adder. The sum is then divided by two using a hardwired shift. This resulting index is

the address to be referenced in the RAM table.

The first index points to the middle of the look-up table. The 24-bit floating-point interval stored in this address is then compared with the number to be converted, which has already been stored in the 24-bit register of the pipelined input. If the interval in the table is larger than the convolved number then register B is loaded with the address of the interval, otherwise register A is loaded with the address. A new iteration begins with the computation of a new index. The operation is repeated until registers A and B hold the same value which means that the best interval has been found. The result of the conversion corresponds to the index (address) pointing in the look-up table.

The principle behind the binary search is that the area of the search is divided by two at each iteration. It can be easily shown that this algorithm will always converge towards a solution within n iterations:

$$n = log_2(b) \tag{4.2}$$

In the above equation, b is the number of elements in the table where the search is being performed. In our case, only 8 iterations are needed since the look-up table stores 256 intervals. Therefore, a conversion can be completed within a pipeline cycle and allows an iteration to last two clock cycles. The 8-bit integer is then ready to be written into one of the output FIFOs.

4.3.3 The Transparent Configuration

The transparent mode of operation provides an independent path which bypasses both the pipelined input and the converter. Since the systolic array pushes the results 4 bits at a time beginning with the lowest significant bits, the output converter has to shuffle the incoming data so as to make them compatible with the Motorola Big Endian byte organization. In this case, the results cannot be written

in the FIFOs until the most significant bits are output from the array. Therefore, the lowest bits have to be temporarily stored in registers before they can be written into the FIFOs. This problem is very similar to the one already explained in section 4.1.4 since both the input and output FIFO buffers are organized similarly.

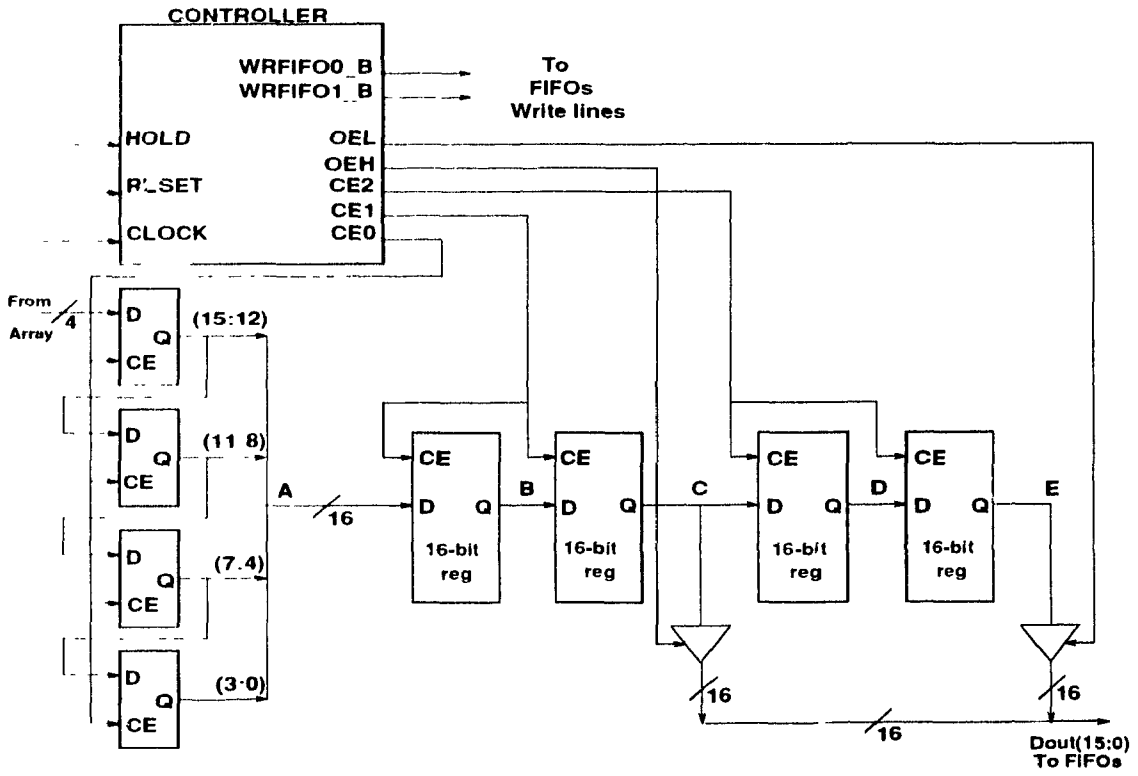


Figure 4.17: Output converter (Transparent configuration)

This simple circuit is illustrated in Figure 4.17. A controller generates control signals such as write lines for the FIFOs and latch enable lines for the multiple registers in the datapath. The operation of the circuit is best explained with the help of the timing diagram of Figure 4.18. Assume that the next floating-point sample to leave the array has a hexadecimal value of FEDC BA98 7654 3210. The sample comes out in chunks of 4 bits starting with 0 and ending with F. These 4-bit values are first shifted through the four 4-bit input registers where they are serialized into 16-bit values which, in turn, will be shifted through the four 16-bit registers over the rest of the pipeline cycle. At the beginning of the next pipeline cycle, these 4 registers hold the floating-point sample. Node C holds value BA98 and node E holds value 3210. On clock pulse 0, BA98 is written to FIFOs 2 and 3

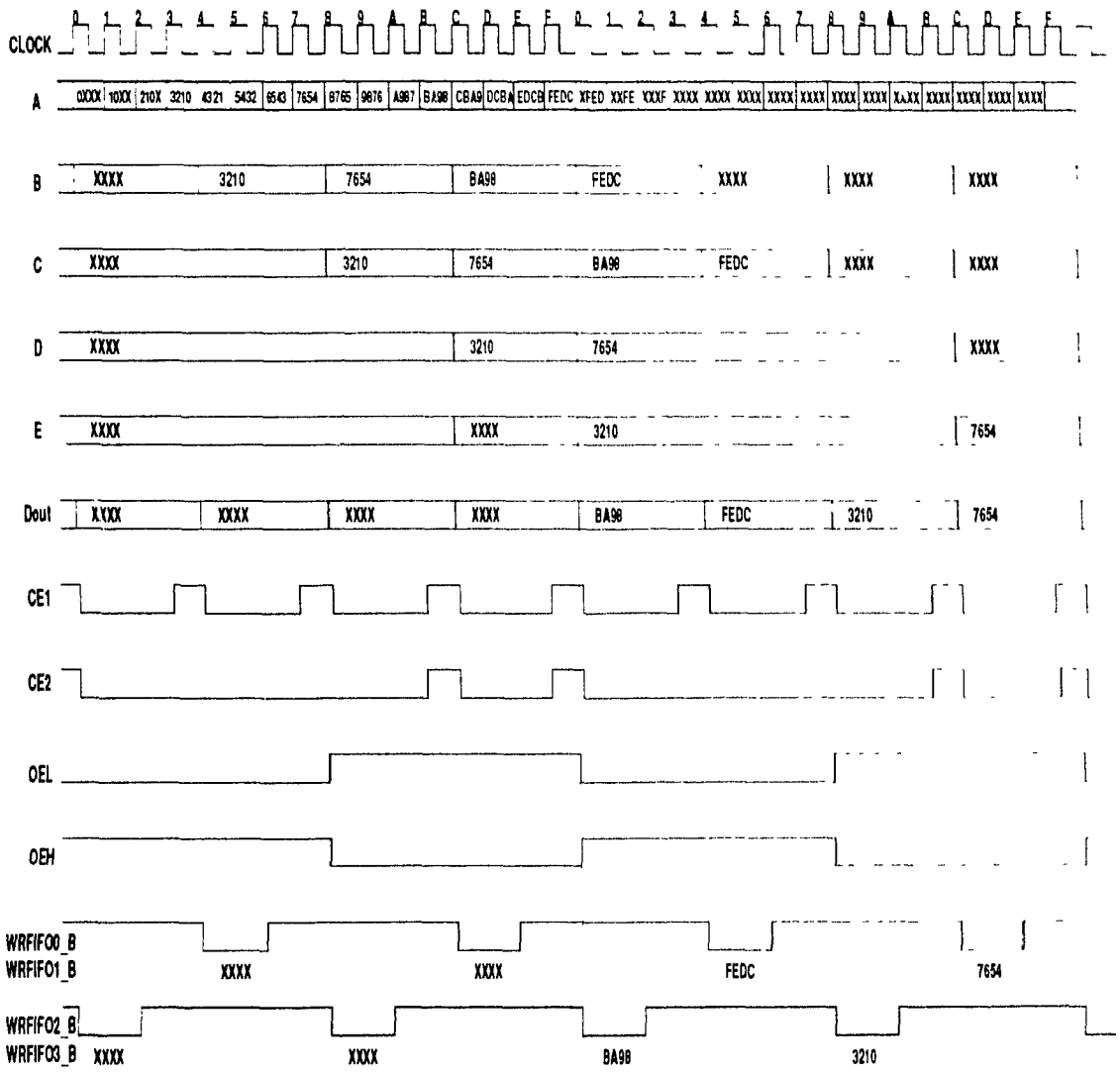


Figure 4.18: Timing diagram of output converter (Transparent configuration)

then on pulse 4, FEDC appears at node C and can be written immediately to FIFOs 0 and 1. On pulse 8, value 3210 at node E is stored in FIFOs 2 and 3. Finally, on pulse 12, value 7654 is shifted to node E and can be stored in FIFOs 0 and 1. In this way, the upper half (32 bits) of the floating-point sample is always written before its lower half as required by the Motorola byte order model. Only one pipeline cycle delay is incurred before the complete floating-point sample is stored in the FIFO buffer.

Chapter 5 Implementation and Results

This chapter describes the implementation issues of the auxiliary subsystems presented in the previous chapter. The first three sections are devoted to the description of the CAE tools that were used during the design process, namely the VHDL design environment, the Mentor Graphics IDEA environment and the Xilinx FPGA development system. Following this, two sections focus on the implementation of the individual subsystems and provide some simulation results. Lastly, system considerations are explored with particular attention being paid to the board-level testing

In this project, a structured approach to design was preferred. In the early phases, behavioral simulations using both the C programming language and the Very High Speed Integrated Circuit Hardware Description Language (VHDL) have been performed. These high-level simulations helped discover potential architectural conflicts and evaluate design alternatives. As a matter of fact, the focus was set on the input/output specifications and verification of the algorithms rather than on registers, adders, combinational logic and their interconnections.

The subsequent steps consisted in the gate-level design of the different subsystems. For this matter, the Mentor Graphics' CAD tool offers a complete environment which supports both schematic capture and simulation. The schematic was first drawn and checked for electrical errors. Then, the design information was extracted for complete functional and timing simulations.

Application specific integrated circuits (ASIC) represent a natural choice for implementing the support circuitry of this convolution system as they help reduce power consumption and system size while increasing overall performance. Among ASIC solutions, the Xilinx XC3000 family of field programmable gate arrays

(FPGAs) offers an interesting alternative for implementing the converters and the controller of the delay memory circuit. The Xilinx development system translates designs into programmable gate arrays and provides the interface with the Mentor Graphics environment for full timing simulation.

A description of the aforementioned computer-aided engineering (CAE) tools follows.

5.1 Intermetrics VHDL Design Environment

The Intermetrics VHDL Design Environment (VDE) [Int, 1990] is a software package which supports the VHDL language as defined by the IEEE 1076-1987 Standard [IEE, 1988]. Version 3.0 is currently available at McGill University and runs under UNIX on SUN 3/60 workstations. The VDE consists of four components: the VDE interface, the analyzer, the simulator and the design database. Figure 5.1 depicts their relationship.

The VDE interface as its name implies, provides the user with a convenient means of accessing the resources of the VDE. More specifically, it allows users to interact with the design database, to create new design libraries, to invoke the analyzer or the simulator and to see the waveforms of a circuit previously simulated. The user interface consists of windows, icons and pull-down menus.

The analyzer checks hardware descriptions for syntactic and semantic errors. It also translates VHDL source text to the intermediate VHDL attributed notation (IVAN) form and stores it in the working library of the design database. These descriptions are stored in a structured library which holds the relationships and attributes relating to the design units.

Simulation involves the use of three programs: the Model Generator, the Build and the Sim programs. The Model Generator reads a unit in IVAN form and

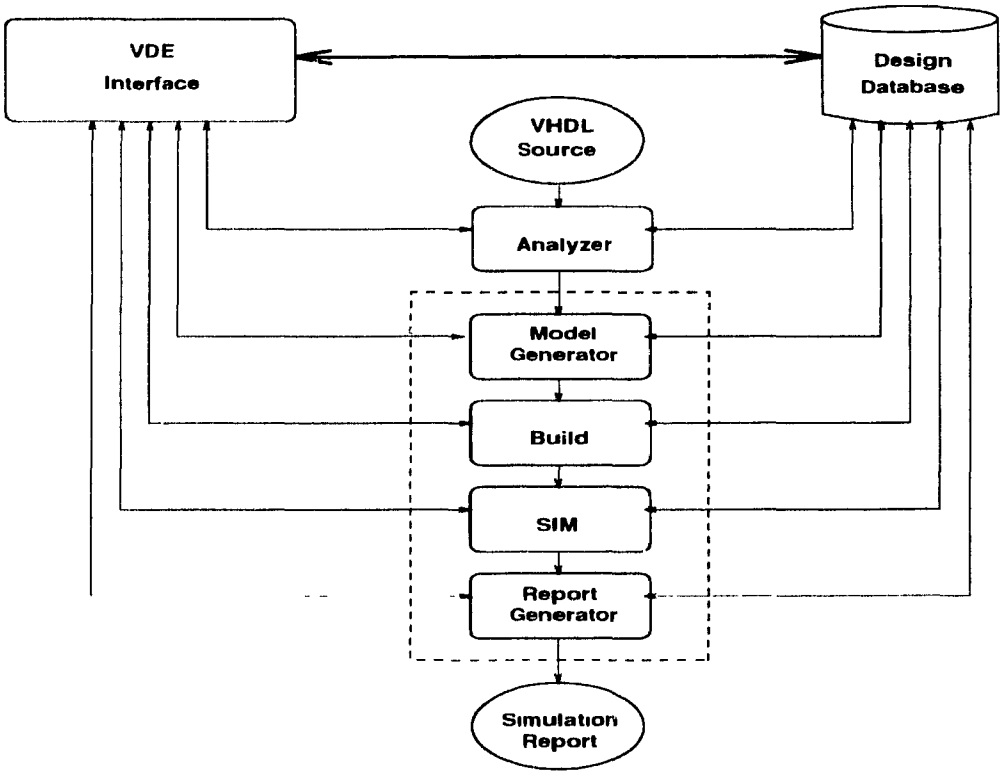


Figure 5.1: The Intermetrics VHDL Design Environment (VDE)

translates it to C object code. The Build program then links the object modules with the Simulation Core library. A simulation kernel is created and stored in the design database. Finally, the Sim program executes this simulation kernel. A simulation run will create a signal directory and a signal trace file which, again, are stored in the database. The simulator is invoked through the VDE interface and can run either in batch or interactive mode. An interactive simulation is very convenient for the debugging of source code. Waveforms may be plotted with the waveform viewer as transition diagrams. A simulation report text file may also be generated using a "report control language".

The VDE tools described above communicate through the design database. Because this is a modular architecture, other tools for synthesis or for automatic test generation could be easily added to the system.

5.2 Mentor Graphics IDEA

Mentor Graphics provides a complete set of application programs to assist in the electronic design automation (EDA) process. These application programs have been created to automate schematic entry, design analysis, IC and PCB layout, test and manufacturing, packaging and documentation. Communication of design data is shared among the application programs via the Mentor Graphics Common Database. Version 7.0 of this package has been installed on Apollo DN3500 workstations running under Domain/OS.

For the purpose of the convolution system, only the schematic entry and the design analysis tools were used. Two application programs support schematic entry:

- *NetED* [Men, 1989b], the network editor, creates schematic sheets of components, nets and ports provided by different libraries. A set of properties are bound to each network entity and holds information such as net name, net delay or pin direction. These properties are then used by downstream applications. NetED supports top-down hardware development by using hierarchical sheets of functional blocks. This structured approach usually produces designs that are more reliable and easier to understand.

- *SymED*, the symbol editor, lets the designer create or modify component symbols which can be placed on the schematic sheets. With SymED, basic blocks that represent off-the-shelf or full custom parts can be drawn and assigned different properties.

Once the design entry is completed, the hierarchical schematic must be *expanded* to produce a flattened design. The *extractor* further processes the design by extracting appropriate information and models to be used by the simulator.

Design analysis is supported by the Quicksim [Men, 1989a] logic simulator

which allows the designer to verify the functionality of the designs produced with NetED and SymED Quicksim is an interactive simulator which means that the user may stimulate the inputs with different test vectors and observe the consequences. Any signal in the design may be traced, listed or monitored. Mentor provides simulation models which can be altered to accommodate similar components with different timing characteristics.

5.3 Xilinx FPGAs and the Development System

5.3.1 Xilinx LCA Architecture

Field programmable gate arrays (FPGAs) are high-density ASICs that are programmed by the end user rather than the semiconductor manufacturer. Xilinx offers a family of FPGAs with different logic capacities, speed and temperature ranges. The XC3000 family can implement logic functions that range from 2000 gates (XC3020) to 9000 gates (XC3090).

Xilinx's proprietary FPGA architecture is called "Logic Cell Array (LCA)" [Xil, 1991] and is very similar to standard gate arrays. The LCA is made of three types of configurable elements: a perimeter of I/O blocks, a core array of logic blocks and resources for interconnection between blocks. Much like a programmable computer peripheral, the functions implemented in an LCA are determined by a configuration program downloaded during system initialization. Since volatile SRAM cells hold the configuration, it is possible to change the configuration of the LCA during operation by loading a new program. This feature, together with the high-density of the FPGA, contributes to minimize the chip count on a board.

The array of configurable logic blocks (CLBs) provides the elements from which the user's logic is constructed. Each CLB has two flip-flops, a combinatorial section

and a control section. The combinatorial part is made of a look-up table which can implement one or two independent functions The propagation delay between the input and the output is independent of the logic function. Since the table has 32 entries, a single function of 5 variables can be generated although 7 variables (a, b, c, d, e, Qx, Qy) are available. Variables a, b, c, d and e may come from another block but Qx and Qy are driven by an internal feedback path coming from the output of the two flip-flops. The two edge-triggered flip-flops share an asynchronous reset and they obtain their input either from the output of the combinatorial logic or from the 'data-in' block input.

Each I/O block (IOB) provides the interface between the user logic and the external world. An IOB includes both direct and registered input paths. Program-controlled memory cells configure an IOB for output by setting signal inversion, tri-state control inversion, direct or registered output as well as control of the slew rate (fast or slow). The global input buffer threshold of the IOBs can be programmed to be compatible with either TTL or CMOS levels. A pull-up resistor may also be selected to prevent unused inputs from floating.

Programmable interconnection resources provide the interconnection of any blocks (IOBs or CLBs) in the LCA. Three types of resources are available, namely, the general purpose interconnects, the direct connections and the long lines. The direct connections can be used to link adjacent blocks only while long lines are intended primarily for signals that must travel a long distance Long lines also allow multiplexed buses and wide AND gate functions when combined with tri-state buffers adjacent to CLBs.

5.3.2 XACT Development System

The development system, XACT [Xil, 1989], is a CAE tool which eases the design implementation process. Two versions are currently available at McGill. Version

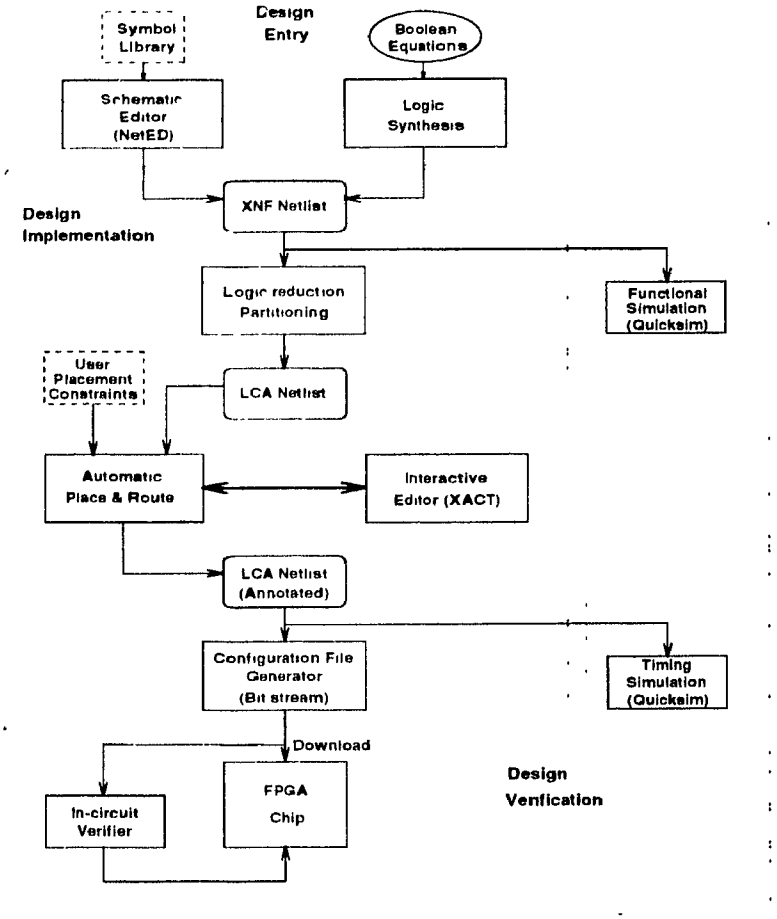


Figure 5.2: Xilinx design flow

3.0 runs on PC-286/386 while version 2.2 runs on Apollo DN3500 computers. A complete interface with parts library enables the designer to enter LCA designs with the Mentor Graphics NetED schematic editor. A timing simulation may be performed with Quicksim before and after the implementation process.

Figure 5.2 depicts the usual design implementation flow. The design methodology involves three main steps:

Design ENTRY: LCA designs may be entered by any combination of schematic capture and boolean equations. Many popular schematic editors such as *NetED* can be used. However, the design must be drawn with the symbol library provided by Xilinx. The boolean equations should be written into a PALASM2-compatible text file. Each portion of the design is then converted

to an intermediate description called the Xilinx netlist file (XNF) The separate XNF files may be merged into a single XNF file subsequently

Design IMPLEMENTATION: Implementing a design in an FPGA entails three operations: technology mapping, placement and routing and configuration generation. From the XNF design file, the XNF2LCA utility maps the logic into the CLBs and IOBs of the LCA architecture Then, an automatic placement and routing program (APR), influenced by the user placement constraints, looks for a suitable block arrangement using an algorithm called "simulated annealing" [Huang *et al.*, 1986] The program then routes these constraint nets to interconnect the blocks. The APR is the critical step in the FPGA design flow since placement and routing choices have a major effect on the performance of the resulting implementation Indeed, Xilinx provides an interactive editor which can be used to "improve" an unsatisfactory routed design. The last operation consists in creating the binary configuration program to be downloaded into the FPGA. This is analogous to a microprocessor assembler.

Design VERIFICATION: Design verification can occur at two points in the design process. After a design is entered, a functional simulation can be run to verify the operation of the logic. Following implementation, another verification may include full timing simulation and in-circuit testing.

5.4 Input Converter Implementation

5.4.1 Schematic Entry

The design of the input converter, as presented in chapter 4, points out three different modes of operation which can be divided into two separate configurations. The first configuration is used to convert either 8- or 16-bit fixed-point numbers

while the second simply performs bytes reversal on incoming 64-bit floating-point numbers. In both configurations, the interface with the external world remains essentially the same despite the internal structural differences. This feature makes possible the development of two "soft" configurations that can be downloaded into a RAM-based programmable gate array such as Xilinx. At any time, the FPGA will hold either configuration depending on the mode of operation. Therefore, by using the reprogrammability feature of the chip, an appreciable circuit size reduction can be achieved. In the future, it will also be possible to upgrade the functions of the input converter without changing the printed circuit board as long as the new features fit into the same chip.

Schematics of both configurations have been drawn in a top-down fashion as shown in Appendix A. The top-level sheet of each circuit contains the input and output ports which connect the chip to the external world. In addition, two functional blocks, a controller and a data path, lead to the lower levels of the hierarchy. A functional block may either point to a sheet with graphic components or to a text file which holds boolean equations. In this case, the former is used to describe the structure of the design using parts from the Xilinx standard library. The latter option yields a more compact logic description which is sometimes closer to the way hardware designers think. Most of the signals generated in the controller have been expressed in PALASM, a PAL description language developed by Monolithic Memories Inc.

Some portions of the schematics contain *frames*. Frames are tools provided by NetED to avoid drawing multiple occurrences of a repetitive portion of the circuit. An example of this can be found in the third sheet (3/3) of the transparent configuration (Contig 64 -> 64 in Appendix A) where a unique multiplexer is drawn although it is instantiated 8 times. An indexed variable is appended to the signals inside the frame to keep track of the proper connections.

5.4.2 Behavioral and Functional Simulations

The schematics of both configurations have been simulated at the gate level with Quicksim. The functional simulation is a unit-delay simulation whose only goal is to debug logic before the implementation into the FPGA. A set of stimuli have been applied to the different nodes in the circuits so as to verify the behavior of the lower levels of the hierarchy.

When the verification of the individual blocks was completed, another set of stimuli was applied to the primary inputs of the converter. Different integer numbers, especially those at the boundaries were then converted and the results were compared with those of a software routine written in the C language, which executes conversions using the same algorithm. The validity of the algorithm was also checked for full compliance with the IEEE standard by means of the internal integer to floating-point conversion routines provided by an IEEE-754-compatible C compiler.

5.4.3 Logic Partitioning

Once the designs proved to be functional, the schematics were flattened (i.e. the hierarchy was removed) and each configuration was partitioned. Partitioning consists in mapping the logic to LCA elements such as the I/O blocks and configurable logic blocks (CLBs). This usually results in a drastic transformation of the logic. In particular, the boolean equations are optimized for an efficient implementation in the CLBs and the unused logic is removed. The final design file is called an LCA file.

Table 5.1 summarizes the partitioning results for each configuration and give the percent chip utilization of two FPGAs of the Xilinx XC3000 family. As mentioned earlier, the number of IOBs used is almost the same; the difference is due to the non-

Configuration	LCA resources used				Percent utilization			
					XC3030PC84		XC3042PC84	
	CLBs	IOBs	CLB Flip-flops	Nets	CLBs	IOBs	CLBs	IOBs
Config 1 (converter)	86	36	65	134	86%	49%	60%	49%
Config 2 (transparent)	57	35	83	99	57%	47%	40%	47%

Table 5.1: Input converter partitioning summary

utilization of the *MODE* input in configuration 2. The largest circuit, configuration 1, requires 86 CLBs even though the other one uses more flip-flops. As a result, both configurations seem to fit well in either FPGA. Nevertheless, the larger FPGA, the XC3042PC84, was selected to ease the placement and routing process. This decision stems from experience gained with previous FPGA experiments and analysis of other FPGA designs which have shown that a CLB utilization of 70% or less usually yields the best implementation results.

For many circuits, the designer has to trade off between integration and speed. A circuit which takes most of the FPGA configurable blocks will be usually more difficult to route because of the scarce interconnect resources provided in the LCA architecture. Consequently, non-negligible net delays should be expected and the speed performance will decrease accordingly.

Knowledge of the internal architecture of the FPGAs helps design a circuit which will be easier to implement. This usually results in an improved performance both in speed and logic density. FPGAs are especially suited for synchronous designs that mix boolean functions and registers. Indeed, the pipeline architecture of the input converter fits into this category.

5.4.4 Placement and Routing

The designer can exercise some control over the automatic placement and route (APR) process by specifying a set of constraints either in the schematic or in a separate file. For instance, constraints may predetermine the placement of I/O blocks, prohibit usage of certain pins or assign weights to critical nets for which delays should be minimized. In regard to the two input converter configurations, a set of 19 pins which have dual functions were prohibited. These special function pins are used to control the downloading of the configuration program upon chip initialization; they become normal user I/O pins afterwards. In order to avoid insertion of supplementary external circuitry to prevent potential conflicts and circuit damage during configuration, the usage of these pins as I/O pins was forbidden. The designer should also avoid the temptation to predetermine I/O block placement before implementing the logic to ease printed board design. Such practice may result in routing congestion near the perimeter of the device. Hence the necessity to rely on the APR program to find the most appropriate IOB placement based on circuit topology. Lastly, crucial nets, especially those which drive a large number of CLBs, were flagged as "critical". The APR usually routes these nets first when most of the interconnect resources are still available. Other nets that must travel a long distance or must have minimum skew were flagged as "longline". Some of these flags may be observed on the schematics included in Appendix A.

The two designs were routed with the APR Version 3.0. It took close to 50 minutes on a PC/386 computer to route configuration 1 while configuration 2 required half an hour. Two reasons explain the APR time difference. First, the transparent configuration, as shown in table 5.1, fits into less CLBs. Second, the I/O block placement set up by the APR for configuration 1 was also used for configuration 2 since they are both destined to the same physical chip. The APR program was executed multiple times until satisfactory results were obtained. Nevertheless, the final implementation of configuration 2 had to be improved manually using the XACT interactive design editor.

5.4.5 Timing Analysis

The final step consisted in the verification of dynamic timing violations with Mentor's Quicksim. Mentor Graphics also provides an application program called "Quickpath" which automatically analyses critical path timings. Unfortunately, this tool is still not available at McGill. Therefore, a manual static timing analysis had to be performed on selected critical paths. In particular, the simulations have pointed out that the input converter implementation has not lived up to the initial expectations. Careful analysis has revealed a maximum operating frequency of 12 MHz for configuration 1; the target frequency was 12.5 MHz. As for configuration 2, it presented no timing violations at the target clock rate. It should be noted, however, that these are worst-case figures valid for a wide range of physical (temperature, manufacturing) conditions. Operation in a typical environment would probably work faster. Additional tests on the prototype board shall support this statement.

Figure D.1 of appendix D presents a simulation run illustrating a typical conversion as performed by configuration 1. In this diagram, the 16-bit integer A9E3 is converted into the 64-bit double precision floating-point number 40E5 BC60 0000 0000. Initially, a conversion cycle begins with the reading of the integer. Two of the read FIFO signals *RDFIFO_B* are first driven low for two clock cycles. The 65 ns access time of the input FIFOs (CY7C429-65) combined with the 35 ns set-up time of DIN make it impossible to read within one clock cycle. Thus, a read cycle is started at clock pulse D of the previous pipeline cycle so as to latch the integer at clock pulse 0 of the subsequent pipeline cycle. From this point on, the conversion proceeds and lasts one pipeline cycle. The results are available at the output (FPOUT) in chunks of 8 bits shortly after clock pulse D. The first four bytes are always 00 (least significant bits of the mantissa). The remaining bytes 60, BC, E5 and 40 appear on clock pulses 5, 8, A and C, respectively.

It was computed that the worst case delay for FPOUT occurs at clock cycle 5.

In this case, the signal FPOUT appears at the output pin 60 ns after the rising edge of the clock. However, this long delay causes no problem since the delay memory circuit will only latch this value on clock cycle 7. This leaves more than enough time for the signal to propagate to the DMC The other bytes will be latched at pulses 9, B and D respectively (in the DMC, they are referred as pulses 8, A and C). It should be noted that the DMC has its own pipeline cycle counter which is leading by one clock pulse the cycle counter of the input converter In other words, a pipeline cycle in the DMC starts one clock pulse earlier than in the input converter. This makes for proper pipeline synchronization.

A simulation example for configuration 2 is presented in section D.2 of appendix D. For evaluation purposes, the timing diagram may be compared to the unit-delay diagram of figure 4.4 since they share the same input vectors. In this case, the 64-bit value FEDC BA98 7654 3210 is to be reorganized and sent to the DMC. The number is read in chunks of 16 bits according to the following sequence: BA98 is read out of the FIFO first, then FEDC, 3210 and 7654 follow at intervals of 4 clock pulses. Then, it takes one pipeline cycle to reverse the data order. The first byte (10) appears at FPOUT on clock pulse E. Subsequently, the remaining bytes will appear every other pulse. The worst case propagation delay for FPOUT occurs on clock pulses 6 and E when signal SEL1 toggles. A 63 ns delay has been computed from the simulation. This leaves only 17 ns for the signal to set up at the input of the DMC before the next rising edge of the clock.

Simulation also reveals some glitches which can be observed on signal SEL1. However, these have no effect on the FPOUT and thus, can be safely ignored. The traditional method used to eliminate glitches which consists in adding redundant logic to a boolean equation is not applicable in this case since, in the Xilinx FPGAs, the combinational logic is implemented with look-up tables. The outputs will be glitch-free only when all inputs change simultaneously.

5.5 Output Converter Implementation

The output converter, like the input converter, has two configurations which shall be implemented in a Xilinx XC3042PC84-100. Configuration 1 (the converter) was implemented in 1990 before the installation of the interface between Mentor and the Xilinx development system on the Apollo computers. A schematic has been drawn with OrCAD and implemented using the version 2.23 of the APR on a PC/286 computer. It should be pointed out that the design has not been simulated at the gate-level yet. Nevertheless, the whole design will be transferred to the Apollo platform soon for full simulation and improved implementation. At the time of this writing, configuration 2 (transparent) is being implemented on the Apollo platform. Preliminary results are shown in table 5.2. The analysis of the

Configuration	LCA resources used				Percent utilization	
			CLB		XC3042PC84	
	CLBs	IOBs	Flip-flops	Nets	CLBs	IOBs
Config 1 (converter)	76	39	69	165	53%	53%
Config 2 (transparent)	49	27	88	86	34%	36%

Table 5.2: Output converter partitioning summary

APR report file has revealed that a signal needs 155 ns to propagate along the critical path of the converter (config 1). This delay breaks up into 5 components:

$T_{registers}$: 7 ns

T_{adder} : 56 ns

$T_{comparator}$: 35 ns

T_{set-up} : 7 ns

T_{net} : 50 ns

Total : 155 ns

Provided that a binary search iteration lasts two clock cycles (recall section 4.3.2) and that the target clock period is 80 ns, only 5 ns ($2 \times 80 - 155 = 5$) are left to read the look-up table. At best, a very fast but expensive static RAM with 15 ns access time could be used. This would yield a total delay of 170 ns and a clock rate of 11.8 MHz.

It can be stated with enough confidence that the performance of the circuit would improve if it were re-routed with the latest version of the APR program. Another alternative would be to implement the circuit into the new Xilinx XC4000 family of FPGAs which offers on-chip RAM and fast carry logic. In this case, the on-chip RAM could replace the look-up table and speed-up the operating rate of the output converter.

5.6 Delay Memory Circuit Implementation

5.6.1 Behavioral Simulations

A thorough understanding of the data flow in the systolic array was prerequisite to the development of the DMC. For this reason, a short program was written to simulate the systolic flow of data in the array. In this program, the multiple stages and registers of a systolic cell are embodied into a simple C structure [Horspool, 1986] which can be further instantiated into an $M \times M$ array to simulate the behavior of the convolver. A simple routine generates an image whose pixels are sent to the M rows of the array. By monitoring the content of selected pipeline stages, data streams timing and pipeline delays were evaluated precisely.

The next step consisted in the development of an algorithm that would provide for the features (up-sampling, borders etc...) desired in the DMC. In this regard,

a second C program was written to assess the performance and validity of the algorithm. The program emulates the behavior of the DMC according to a set of parameters such as kernel size, sampling mode, image width and row pipeline delay which are entered by the user.

Different simulations have clearly shown that the adopted algorithm is valid for some restricted cases only. In particular, the pipeline delay of a single row of the systolic array must be even (odd) if the total image width (image + borders) is even (odd). In addition, the algorithm works properly when the ratio $\frac{M-1}{s}$ is an even integer where M is the size of the array and s is the up-sampling rate. As a result, this algorithm does not scale well; only a restricted set of array sizes can be accommodated. However, these rather theoretical limitations should have no real consequence on this project since the array size is fixed at 9×9.

Subsequently, a VHDL description of the DMC was written to evaluate different circuit architectures. In this case, the behavior of the DMC is defined at two levels. First, the DMC is divided into a set of processes whose behavior is described using sequential VHDL programming. Examples of processes include state machines, memory and multiplexers. Second, the relationship between those processes is defined using the concurrent programming possibilities of VHDL. More specifically, each process description starts with a *process* statement [Lipsett *et al.*, 1989] to which is attached a sensitivity list. During simulation, the change of a signal state triggers a concurrent execution of all the processes which are sensitive to it.

The VHDL description was mostly used as an intermediary step to alleviate the gate-level design of the DMC. Besides, it enabled the author to focus on potential architectural conflicts without having to worry about the details of a particular implementation.

5.6.2 Schematic Entry

The schematics of the delay memory circuit presented in Appendix B show that the controller block is implemented in a Xilinx XC3030PC84-100 programmable gate array while the data path consists of 12 TIBPAL22V10-15BC from Texas Instruments and 8 SRAMs CY7C186 from Cypress Semiconductor Corporation. These high-performance RAMs have a maximum access time of 35 ns and are organized as 8K words of 8 bits each. During 1-D operation, the memory chips are deselected and enter a power-down feature which reduces power consumption by 73%. However, during 2-D operation, they are continuously selected so as to minimize read and write cycle times.

The schematics of the controller present a hierarchical structure from which two major blocks stand out. The first block, the pointer, consists of a 13-bit modulo counter which generates the address lines to access the RAMs. In this case a comparator implements the modulo feature by resetting the counter at the appropriate time based on the current counter value, the up-sampling mode and the image width. The second block points to a PALASM file that describes in a compact format the behavior of the three state machines. Such a representation improves schematic intelligibility but, contrary to a traditional gate representation, results in a slightly less efficient implementation due to the lack of control over the partitioning process. In fact, during logic synthesis, most boolean equations are optimized for the LCA architecture and some internal signals are buried into the CLBs. The designer who aims at a better partitioning and routing is given little influence over this "technology mapping" by the development system.

As shown in table 5.3, the configuration fills 72% of the CLBs of the XC3030 FPGA. This high resource utilization combined with the weak control over the partitioning phase have made the circuit very difficult to route. Indeed, Version 2.24 of the APR has never managed to route all the nets of this circuit. The final implementation has been completed only when Version 3.0 became available at the

Configuration	LCA resources used				Percent utilization	
			CLB		XC3030PC84	
	CLBs	IOBs	Flip-flops	Nets	CLBs	IOBs
DMC (controller)	72	35	40	103	72%	46%

Table 5.3: DMC controller partitioning summary

end of 1991.

With the early unsuccessful attempts at implementing the controller, it was decided not to use field programmable gate arrays for the data path section of the DMC. The data path, although very regular in nature, is also I/O intensive and time critical. This makes it a good candidate for implementation in programmable logic devices (PLDs) which offer lower density but have net delays known in advance. The regular structure permits the 12 PALs necessary for this circuit to be programmed with only one standard JEDEC file. The target device, the PAL 22V10, was selected for its wide availability although there exist more suitable PALs with additional user I/O pins which would reduce chip count. A gate-equivalent description of the PAL circuit is presented in Appendix C. This appendix also includes the design file processed by the Texas Instruments' ProLogic compiler [Tex, 1991] to obtain a JEDEC fuse map. This standard file will be used by a logic programmer to customize the device to a specific function by "blowing" selected fuses.

5.6.3 Timing Analysis

Preparing for timing simulation entails modifying the simulation model of the generic parts which have been instantiated in the schematic. In fact, Mentor Graphics libraries usually provide a single generic model for a part even if different speed versions of this part are commercially available. So, prior to simulation, the CY7C186 SRAM and PAL 22V10 timing models were changed to match the

characteristics specified in the data books. These include set-up and hold times, propagation delays and memory access time.

According to simulations, the DMC works properly at the target speed (12.5 MHz) if 35 ns access time SRAMs are used along with 15 ns propagation delay PALs. Memory accesses have proved to be critical because only one clock period is allowed for a read or write cycle. When no up-sampling is performed on the image, reads and writes occur alternately. In this case, memories are continuously selected so as to minimize access time. Even so, the address should never be allowed to change when the write enable line is asserted as this could result in spurious writes. As it stands now, address changes occur immediately after a write operation, leaving only a few nanoseconds for the write enable signal to be deactivated. In order to guarantee proper operation in "the real world", one has to take into account some extra delays not handled by the simulator. These include off-chip delays which are due to capacitive loading on memory control lines and on the address bus. They are usually proportional to wire length and signal fan out. Consequently, there is a non negligible probability that, on the prototype, the write enable signal will be skewed enough to cause spurious writes during address changes. This will have to be verified very carefully on the prototype.

Different solutions exist to solve this problem. One would consist in delaying the address lines further in the FPGA with additional buffers. However, this simple trick is defeated because the Xilinx development software usually removes all unnecessary logic including additional buffers. Another solution, which implies extra circuitry on the board, would be to add a *delay-type* IC to delay the address lines at the output of the FPGA. Another option would be to have an alternate clock running at a multiple frequency of the main clock. The extra rising and falling edges provided by the clock during memory access would permit better synchronization control.

Further simulations have shown that a read operation always occur during odd

pipeline cycles and start at most 24 ns after the rising edge of the clock when the address bus has settled. Then, it takes 35 ns before the data appear on the bus. At this point, 13 ns are needed for the data to propagate to the flip-flops of the PAL, leaving only 8 ns of slack before the next clock cycle. On the other hand, a write cycle always occurs immediately after a read operation without changing the address lines. A write is completed when the write enable line is deactivated at the next rising edge of the clock. However, only 54 ns are necessary, including 39 ns to enable the tri-state buffers and 15 ns for data to set-up before disabling the write control line.

At each clock cycle, a delay of 35 ns has been noticed before the new 4-bit values appear at the 9 output rows of the DMC. Of these 35 ns, 20 ns are taken for the generation of the *SEL* signal which selects the proper multiplexer inputs in the PALs, and 15 ns are needed for the data to propagate through the multiplexers to the row outputs. This leaves 45 ns before the next clock cycle, at which point the data is latched into the systolic array. A simulation run showing memory cycles and two rows of the DMC is included in Section D.3 of Appendix D for reference.

Particular attention has been devoted to the control signals generated inside the FPGA. Signals such as those which set up the operating mode (*DIM, WIDTH, S0 and S1*) have been flagged "non critical" prior to APR because they never change during a convolution operation. As a result, delays as high a 29 ns are not uncommon on these nets. Whenever these parameters change, it is important to wait at least two complete clock cycles before starting a new convolution. Analysis of the APR report file has also revealed that it takes at most 37 ns to propagate the *HALT* signal to the output pin. This leaves more than enough time (45 ns) for the signal to travel to the input converter and set up before the next clock tick. Additional information concerning control signals is presented in Section D.4 of Appendix D.

5.7 System Considerations

5.7.1 Initialization

At power-up or upon system reset, the convolution system enters an initialization cycle before any computations may take place. This procedure consists mainly in downloading configuration files to the FPGAs, loading the systolic array with a set of 81 coefficients and transferring a new set of values to the look-up table of the output converter if integer outputs are desired. These tasks are handled by the DMA engine (recall Section 3.5.1) which also sets up the various board signals (up-sampling, 1-D/2-D signal, image width, 8/16-bit conversion etc.) according to the specifications entered on the host computer by the user.

The choice of a configuration mode for the FPGAs is influenced by the actual operating environment. Since the FPGAs are being used with a 68020 microprocessor (DMA engine), it is natural that the FPGAs be considered as peripheral devices accessed by the 68020 through its data bus and a few control lines. In the peripheral mode, an FPGA is mapped to a single byte location in the address space of the 68020. The 68020 simply downloads the configuration file by writing the bytes successively to the same location.

In case of a board with multiple FPGAs, it is possible to ease the configuration procedure by connecting the devices in a daisy chain and then download a composite configuration program. In respect to the convolution system, two FPGAs out of three, namely, the input converter and the DMC are daisy chained. The lead device (the input converter) is set to the peripheral mode and is directly written to by the 68020. When it has received its configuration program, the remaining bytes are passed on to the last device in the chain (DMC) which is set to the slave mode. The last device (output converter) stands alone and can be configured using the peripheral mode. This way, it will be possible to reconfigure the output converter freely without interfering with the other FPGAs. However, a reconfiguration of

the input converter also implies a reconfiguration of the DMC, both devices being tied in a daisy chain.

The configuration program is approximately 22 KBytes long for an XC3030 device and 30 Kbytes long for an XC3042. Downloading the configuration files into the three FPGAs will require close to 100 ms. The reader is referred to [Xil, 1991] for further information concerning FPGA programming.

5.7.2 Pipeline Delays

The key contribution of pipelining is that it provides a way to increase throughput by starting a new task before a previous one has been completed [Stone, 1987]. This is achieved by splitting the processes into steps of equal duration. Then the steps, or stages, work in parallel but on different operands. In this way, a task can be completed every cycle provided that all the stages are filled with data. But when a pipeline is reset, a transient of N cycles is incurred before the first valid result comes out because data have to be pumped along the N stages of the pipeline.

In like manner, the convolution system completes the convolution of a sample or pixel every 16 clock ticks, that is one pipeline cycle. However, the first N results produced by the convolver have to be rejected since the pipeline is filling up. This N-cycle transient is computed by summing up the delays in the DMC, the systolic array and the output converter. Note that the input converter does not influence the transient since the first samples to be sent to the array are zero-valued pixels generated in the DMC. The total delay in the systolic array is 252 pipeline cycles (9 rows × 28 stages/row) but only 2 cycles in the output converter, 1 if it is operating in transparent mode. The total delay in the DMC depends on multiple parameters such as image width, 1-D or 2-D operation and up-sampling mode. In 1-D operation the delay is null because data do not circulate through row memories, but in 2-D

operation, the delay is expressed as follows:

$$D_{dma} = 8 + s * (W - 20/s)$$

where W is the original image width and s is the up-sampling rate (1x, 2x or 4x). In summary, the total pipeline delay (Λ) for 1-D is 253 in output transparent mode and 254 in output conversion mode. The same delay applies for 2-D operation to which D_{dma} is added.

Practically, upon system reset, N transient results will have to be discarded before the pipeline becomes full. At this point, a result is available every pipeline cycle. For a typical 512 × 512 image non up-sampled, this transient represents only 1.6% of the total convolution time but, for smaller images, it becomes increasingly important.

5.7.3 Design for Testability

With the ever increasing circuit density, a new trend has been observed in chip and board testing: Design for testability. DFT techniques aim at decreasing test costs while improving fault coverage. At the chip and board levels, DFT makes for easily controllable and observable nodes with the addition of extra logic or with the adoption of novel architectures.

One of the many advantages of Xilinx's FPGA reprogrammability is that different test configurations can be developed to implement self-diagnostics. For instance, some configurations already exist to test the internal elements of an FPGA and guarantee 100% fault coverage. In addition, each FPGA provides a readback mode which may be used for verification of configuration or as a method of determining the state of internal logic nodes during in-circuit debugging. In spite of that, an easier method to probe internal nodes would be to connect unused FPGA I/O pins to selected internal nodes during the testing phase of the design.

[Fawcett, 1989].

PALs may also be easily verified by specifying a set of vectors in the Prologic description. Upon compilation, these vectors will be inserted in the standard JEDEC file and be used by the device programmer to functionally test the PAL 22V10. As for the static RAMs, a test configuration can be developed in the FPGA that implements the DMC to perform reads and writes of all storage bits in the memories.

To facilitate board-level testing, the JTAG/IEEE boundary-scan standard [Maunder and Tulloss, 1992] has been established and is currently picking up the attention of many integrated circuit vendors. A boundary-scan-compatible chip provides a 5-pin interface that gives access to the input and output pins of a chip via a shift-register path between the pins. By shifting in test vectors and commands into the device to control the driving of their outputs and then by shifting out the results, it is possible to find PCB shorts/opens and I/O faults and even perform functional testing.

In regard to this recent technique, the FPGAs used in the convolution system could be replaced with the more recent XC4000 family of gate arrays which implements the JTAG standard. This upgrade would not only provide a framework for efficient and complete testing but also an improved circuit speed performance.

Chapter 6 Conclusion

Several convolution processor boards have been developed in the past to speed up image processing computations, but those offering appreciable performance only support integer arithmetic. However, recent advances in image processing research have led to the use of floating-point arithmetic which allows a larger dynamic range and higher accuracy.

After a brief review of the relevant literature, this thesis presented the design of a double precision floating-point system for convolution. The core of the system is the systolic array of custom VLSI processors which implement the basic multiply-and-accumulate operation. The systolic array system is reconfigurable; that is, it can apply either a 9×9 kernel on an image or a 81×1 kernel on a unidimensional signal. An input converter was designed to efficiently convert integer format data to double precision floating-point format. Similarly, at the output, a converter translates the floating-point convolved data to integer format for display purposes. A delay memory circuit re-organizes the incoming data sequence to create 9 streams, one for each row of the systolic array. It also solves the border effect problem by extending the image with a frame of zero-valued pixels and performs 1, 2 or 4 times up-sampling on the data as desired.

Since the design is completed and simulated, the next step is to build a prototype which should achieve a performance of 126 MFLOPS at a target clock rate of 12.5 MHz. The convolver will be integrated into the Sensor Computing Environment of the McGill Research Center for Intelligent Machines and will be able to perform a 9×9 convolution on a 512×512 frame in about one-third of a second.

Eventually, the custom VLSI cells should be re-implemented using a 0.8 micron process. Such a process would allow the circuit to operate at a higher clock rate

and should allow multiple cells to fit within a single chip. Similarly, the auxiliary subsystems should be implemented with the Xilinx XC4000 family of FPGAs which offers high performance and better resource utilization.

References

[Annaratone, 1987] M. Annaratone, "The warp computer: Architecture, implementation and performance," *IEEE Transactions on Computers*, vol. C-36, pp 1523–1538, December 1987.

[Babaud *et al*, 1986] J Babaud, A. P. Witkin, M. Baudin, and R. O. Duda, "Uniqueness of the Gaussian kernel for scale-space filtering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 8, pp. 26–33, January 1986.

[Balsara and Irwin, 1991] P T. Balsara and M. J. Irwin, "Image processing on a memory architecture," *Journal of VLSI Signal Processing*, vol. 2, pp. 313–324, May 1991

[Barberi *et al*, 1991] F Barberi, R Bernstein, *et al*. "Displaying morphological and lithological maps A numerically intensive computing and visualization application," *IBM Journal of Research and Development*, vol. 35, pp. 78–87, Jan-March 1991.

[Can, 1989] Canadian Microelectronics Corporation, Carruthers Hall, Queen's University, Kingston, Canada, *Guide to the Integrated Circuit Implementation Services of the Canadian Microelectronics Corporation*, GICIS version 4:0 ed., March 1989.

[Côté, 1990] J.-F. Côté, "The design of a testable floating point convolution processor," Master's thesis, McGill University, November 1990.

[Crisman and Webb, 1991] J. D. Crisman and J. A. Webb, "The warp machine on navlab," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 451–465, May 1991.

[Deriche, 1990] R. Deriche, "Fast algorithms for low-level vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 78–87, January 1990.

[Dikaiakos and Steiglitz, 1991] M. D. Dikaiakos and K. Steiglitz, "Comparison of tree and straight-line clocking for long systolic arrays," *Journal of VLSI Signal Processing*, vol. 2, pp. 287–299, May 1991.

[Drolet *et al*, 1990] J. Drolet, J. Panisset, J. Côté, F. Larochelle, and A. Malowany, "A double precision floating point convolution system," in *Proceedings of the ASME International Computers in Engineering Conference, Vol 2*, (Boston, Mass.), pp. 1–6, American Society of Mechanical Engineers, August 1990.

[Drolet *et al.*, 1991] J. Drolet, J. Panisset, and A. Malowany, "Design of a floating point convolution processor," in *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, (Quebec City, Canada), pp. 13.5.1–13.5.4, Canadian Society for Electrical and Computer Engineering, September 1991.

[Duncan, 1990] R. Duncan, "A survey of parallel computer architectures," *Computer*, vol. 23, pp. 5–16, February 1990

[Fawcett, 1989] B F Fawcett, "User-programmable gate arrays: Design methodology and development systems," *Microprocessors and Microsystems*, vol 13, pp. 321–327, June 1989

[Ferguson, 1991] W. Ferguson, Jr., "Selecting math coprocessors," *IFIT Spectrum*, vol. 28, pp. 38–41, July 1991.

[Fisher and Kung, 1984] A. Fisher and H. Kung, "Synchronizing large VLSI processor arrays," *IEEE Transactions on Computers*, vol. C-34, pp 734-740, May 1984.

[Flynn, 1966] M J Flynn, "Very high speed computing systems," *Proceedings IFIT*, vol. 54, pp. 1901–1909, December 1966.

[Foulser, 1987] D. E Foulser, "The saxpy matrix-1: A general-purpose systolic computer," *Computer*, vol. 20, pp. 35–43, July 1987

[Fox, 1991] E. A. Fox, "Advances in interactive digital multimedia systems," *Computer*, vol. 24, pp. 9–21, October 1991.

[Freer, 1987] J Freer, *Systems Design with Advanced Microprocessors* Howard W. Sams & Co., 1987.

[Gall, 1991] D L. Gall, "MPEG: A video compression standard for multimedia applications," *Communications of the ACM*, vol. 34, pp 46–58, April 1991

[Godon *et al.*, 1990] F. Godon, D. Al-Khalili, and R Inkol, "A memory controller for mapping an array of circular buffers into a RAM," in *33rd Midwest Symposium on Circuits and Systems VOL II*, (Calgary, Alb., Canada), pp 645–648, IEEE Circuits and Systems Society, August 1990.

[Haule, 1990] D. D Haule, "Design of a VLSI system for image processing," Master's thesis, McGill University, March 1990

[Hennessy and Patterson, 1990] J. L. Hennessy and D. A Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Inc., 1990.

[Hillis, 1985] W. D. Hillis, *The Connection Machine* Massachusetts Institute of Technology Press, 1985.

[Horspool, 1986] R. N. Horspool, *C Programming in the Berkeley Unix Environment*. Prentice Hall, 1986.

[Huang *et al.*, 1986] M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "A efficient general cooling schedule for simulated annealing," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, 1986.

[IEE, 1985] IEEE, *IEEE Standard 754-1985 for Binary Floating-Point Arithmetic*, 1985.

[IEEE, 1988] IEEE Std 1076-1987, *IEEE Standard VHDL Language Reference Manual*, March 1988.

[Int, 1990] Intermetrics Inc., *VHDL Design Environment User's Manual*, December 1990

[Kandle, 1987] D A Kandle, "A systolic signal processor for signal-processing applications," *Computer*, vol. 20, pp. 94–95, July 1987.

[Kehtarnavaz *et al*, 1991] N Kehtarnavaz, N Griswold, and J. Lee, "Visual control of an autonomous vehicle (bart) - the vehicle-following problem," *IEEE Transactions on Vehicular Technology*, vol 40, pp 654–662, August 1991.

[Kung and Leiserson, 1978] H Kung and C Leiserson, "Systolic arrays (for VLSI)," *Sparse Matrix Symposium (SIAM 1979)*, pp. 256–282, 1978.

[Kung and Webb, 1984] H Kung and J. Webb, "Wafer-scale integration and two-level pipelined implementations of systolic arrays," *Journal of Parallel and Distributed Computing*, pp 32–63, 1984.

[Kung *et al*, 1981] H Kung, L. M. Ruane, and D W. Yen, "A two-level pipelined systolic array for convolutions," in *Proceedings of the CMU Conference on VLSI Systems and Computations*, pp. 255–264, 1981.

[Kung, 1982] H. Kung, "Why systolic architectures?," *Computer*, vol. 15, pp. 37–46, January 1982.

[Kung, 1988] S Kung, *VLSI Array Processors*. Prentice Hall, 1988.

[Kwan and Samuel, 1990] H.-K. Kwan and T. Samuel, "2-D systolic arrays for realization of 2-D convolution," *IEEE Transactions on Circuits and Systems*, vol. 37, pp. 267–273, February 1990

[Larochelle *et al.*, 1989] F Larochelle, J Côté, and A. Malowany, "A floating point convolution systolic cell," in *Proceedings of Vision Interface '89*, (London, Ont., Canada), pp. 77–80, June 1989.

[Lee and Aggarwal, 1990] S.-Y. Lee and J. K. Aggarwal, "A system design/scheduling strategy for parallel image processing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 194–204, February 1990.

[Levine, 1985] M D. Levine, *Vision in Man and Machine*. McGraw-Hill Inc., 1985.

[Lin *et al*, 1987] K-S Lin, G. A. Frantz, and R. Simar, Jr., "The TMS320 family of digital signal processors," *Proceedings of the IEEE*, vol. 75, pp. 1143–1159, September 1987.

[Lipsett *et al.*, 1989] R. Lipsett, C. F. Schaefer, and C. Ussery, *VHDL: Hardware Description and Design*. Kluwer Academic Publishers, 1989.

[Lopresti, 1987] D. P. Lopresti, "P-nac: A systolic array for comparing nucleic acid sequences," *Computer*, vol. 20, pp. 98–99, July 1987.

[Lovett and Thakkar, 1988] T. Lovett and S. Thakkar, "The symmetry multiprocessor system," in *Proceedings of the 1988 International Conference on Parallel Processing*, (University Park, Pennsylvania), pp. 303–310, 1988.

[Malowany and Malowany, 1988] M. Malowany and A. Malowany, "Making curvature estimates of range data amenable to a VLSI implementation," in *Proceedings of the SPIE conference on Vision Communications and Image Processing IV*, (Cambridge, Mass.), pp. 345–353, 1988.

[Manohar and Baudet, 1990] S. Manohar and G. Baudet, "Pragmatic approach to systolic design," *IEE Proceedings Part E, Computers and Digital Techniques*, vol. 137, pp. 2[illegible]90.

[Marr and Hildreth, 1980] D. Marr and E. Hildreth, "Theory of edge detection," in *Proceedings of the Royal Society of London, Ser. B, Vol 207*, pp. 187–217, 1980.

[Maunder and Tulloss, 1992] C. M. Maunder and R. E. Tulloss, "Testability on tap," *IEEE Spectrum*, vol. 29, pp. 34–37, February 1992.

[McRCIM, 1990] McRCIM, "Mcgill research center for intelligent machines annual report," tech. rep., McGill University, 1990.

[Mead and Conway, 1980] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison Wesley, 1980.

[Men, 1989a] Mentor Graphics Inc., *Quicksim User's Manual (Software V7.0)*, April 1989.

[Men, 1989b] Mentor Graphics Inc., *Schematic Capture User's Manual (Software V7.0)*, March 1989.

[Moreno and Lang, 1990] J. H. Moreno and T. Lang, "Matrix computations on systolic-type meshes," *Computer*, vol. 23, pp. 32–51, April 1990.

[Mot, 1982] Motorola Semiconductor Products Inc., *VMEbus Specification Manual*, MVMEBS/D1, revision B ed., August 1982.

[Nash and Petrozolin, 1985] J. Nash and C. Petrozolin, "VLSI implementation of a linear systolic array," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pp. 1009–1012, IEEE, 1985.

[Nelson and Aloimonos, 1989] R. Nelson and J. Aloimonos, "Obstacle avoidance using flow field divergence," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 1102–1106, October 1989.

[Neumann, 1987] J V Neumann, *"First Draft of a Report on the EDVAC" (1945), Papers of John von Neumann on Computing and Computer Theory*. The MIT Press, Cambridge, Mass , 1987

[Nowinski, 1990] W L Nowinski, "Parallel implementation of the convolution method in image reconstruction," in *Lecture Notes in Computer Science. CONPAR 90-VAPP IV, Joint International Conference on Vector and Parallel Processing*, (Zurich, Switzerland), pp 355–364, September 1990

[Panisset *et al* , 1990] J Panisset, J Drolet, J. Côté, F Larochelle, and A. Malowany, "A floating point convolution system," in *33rd Midwest Symposium on Circuits and Systems*, (Calgary, Alb., Canada), pp. 397–400, IEEE Circuits and Systems Society, August 1990

[Papamichalis and Simar, 1986] P. Papamichalis and R. Simar, Jr., "The TMS320c30 floating-point digital signal processors," *IEEE Micro Magazine*, vol. 8, pp. 10–28, December 1986

[Parthenis *et al.*, 1990] K Parthenis, C Metaxaki-Kossionides, and B. Dimitriadis, "An automatic computer vision system for blood analysis," *Microprocessing and Microprogramming*, vol. 28, pp 243–246, March 1990.

[Proakis and Manolakis, 1988] J. G. Proakis and D. G. Manolakis, *Introduction to Digital Signal Processing*. New-York: Macmillan, 1988

[Robbins and Robbins, 1989] K. A. Robbins and S. Robbins, *The Cray X-MP/Model 24. A Case Study in Pipelined Architecture and Vector Processing*. Springer-Verlag, 1989

[Schafer and Rabiner, 1973] R. W Schafer and L. R. Rabiner, "A digital signal processing approach to interpolation," *Proceedings of the IEEE*, vol. 61, pp. 692–702, June 1973

[Schmidt and Caesar, 1991] U. Schmidt and K. Caesar, "Datawave: A single-chip multiprocessor for video applications," *IEEE Micro*, vol. 11, pp. 22–25,88–93, June 1991.

[Sheblee *et al.*, 1989] J. Sheblee, C. Allen, and C. Goutis, "A multi-processor accelerator for 2-D image handling," in *SPIE vol. 1199 Visual Communications and Image Processing IV*, pp. 1137–1145, 1989.

[Skinner *et al* , 1990] D R. Skinner, K. K Benke, and M. J. Chung, "Application of adaptive convolution masking to the automation of visual inspection.," *IEEE Transactions on Robotics and Automation*, vol. 6, pp. 123–127, February 1990.

[Sklar, 1988] B Sklar, *Digital Communications, Fundamentals and Applications*. Prentice Hall, 1988.

[Stewart, 1991] K. W. Stewart, "Multisensor visualization for underwater archaeology," *IEEE Computer Graphics and Applications*, vol 11, pp 13–18, March 1991.

[Stoll, 1991] E P. Stoll, "Picture processing and three-dimensional visualization of data from scanning tunneling and atomic force microscopy," *IBM Journal of Research and Development*, vol 35, pp 67–77, Jan-March 1991

[Stone, 1987] H. S. Stone, *High-Performance Computer Architecture* Addison-Wesley, 1987.

[Tex, 1991] Texas Instruments, *The ProLogic Computer (Release 2.0)*, 1991.

[Tillett, 1989] R. Tillett, "Locating mushrooms for robotic harvesting," in *SPIE Vol. 1193, Intelligent Robots and Computer Vision VIII Systems and Applications*, (Philadelphia, Pennsylvania), pp. 260–267, 1989.

[Weems *et al.*, 1991] C C. Weems, C. Brown, J. A. Webb, T. Poggio, and J. R. Kender, "Parallel processing in the DARPA strategic computing vision program," *IEEE Expert*, vol. 6, pp. 23–38, October 1991

[Whitby-Strevens, 1985] C Whitby-Strevens, "The transputer," in *Proceedings of the 12th International Symposium on Computer Architecture*, (Boston, Mass.), pp. 292–300, 1985.

[Xil, 1989] Xilinx Inc., San Jose, CA, (USA), *XACT Logic Cell Array Development System, Volumes I,II,III and IV*, 1989.

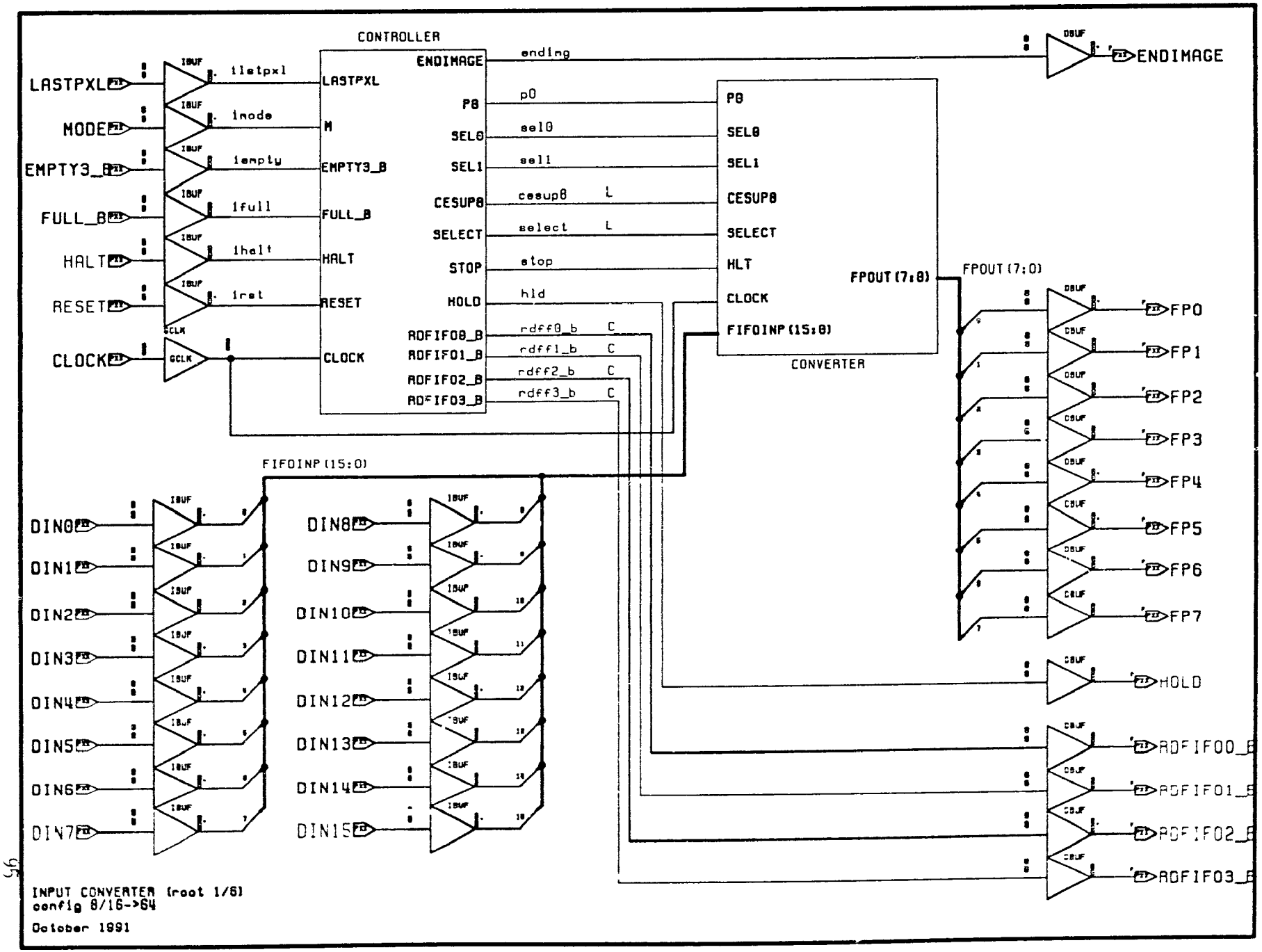
[Xil, 1991] Xilinx Inc., San Jose, CA, (USA), *The Programmable Gate Array Data Book*, 1991.

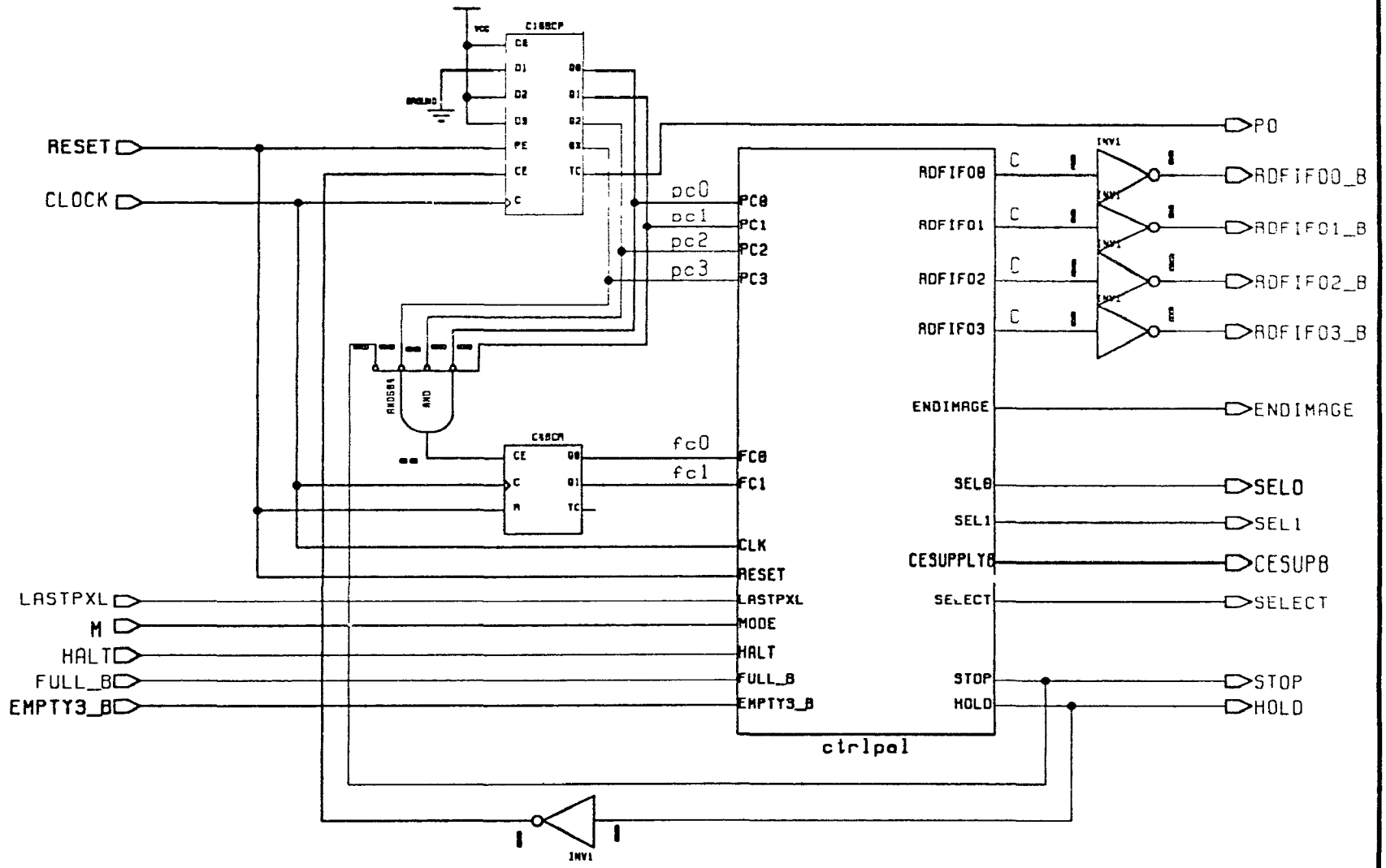
Appendix A Input Converter Schematics

Schematics included in this appendix:

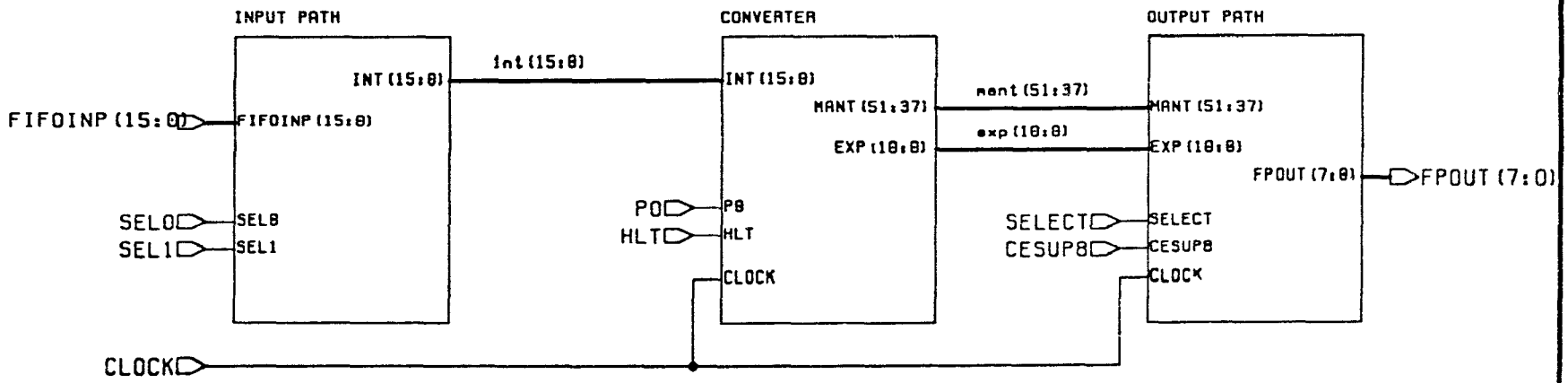
- Configuration 1 (8/16->64) of input converter. This configuration performs the conversion from 8- or 16-bit integers to 64-bit floating-point numbers. The schematics consist of 6 sheets plus 1 text sheet describing boolean equations referred to by the *ctrlpal* block in sheet 2/6.
- Configuration 2 (64->64) of input converter. This is the transparent configuration which rearranges the incoming floating-point numbers. The schematics consist of 3 sheets plus 1 text sheet that describes boolean equations referred to by the *ctrlpal2* block in sheet 1/3 of the schematics.

Both configurations fit into the same XC3042PC84-100 field programmable gate array.


CONTROLLER
LASTPXL
MODE
EMPTY3_B
FULL_B
HALT
RESET
CLOCK
IBUF
GCLK
ilstpxl
imode
iempty
ifull
ihalt
irst
LASTPXL
M
EMPTY3_B
FULL_B
HALT
RESET
CLOCK
ENDIMAGE
P0
SEL0
SEL1
CESUP0
SELECT
STOP
HOLD
RDFIFO0_B
RDFIFO1_B
RDFIFO2_B
RDFIFO3_B
ending
p0
sel0
sel1
cesup0
L
select
L
stop
hld
rdff0_b
C
rdff1_b
C
rdff2_b
C
rdff3_b
C
P0
SEL0
SEL1
CESUP0
SELECT
HLT
CLOCK
FIFOINP(15:0)
CONVERTER
FPOUT(7:0)
FPOUT(7:0)
OBUF
ENDIMAGE
FP0
FP1
FP2
FP3
FP4
FP5
FP6
FP7
HOLD
RDFIFO0_B
RDFIFO1_B
RDFIFO2_B
RDFIFO3_B
FIFOINP(15:0)
DIN0
DIN1
DIN2
DIN3
DIN4
DIN5
DIN6
DIN7
DIN8
DIN9
DIN10
DIN11
DIN12
DIN13
DIN14
DIN15
INPUT CONVERTER (root 1/6)
config 8/16->64
October 1991

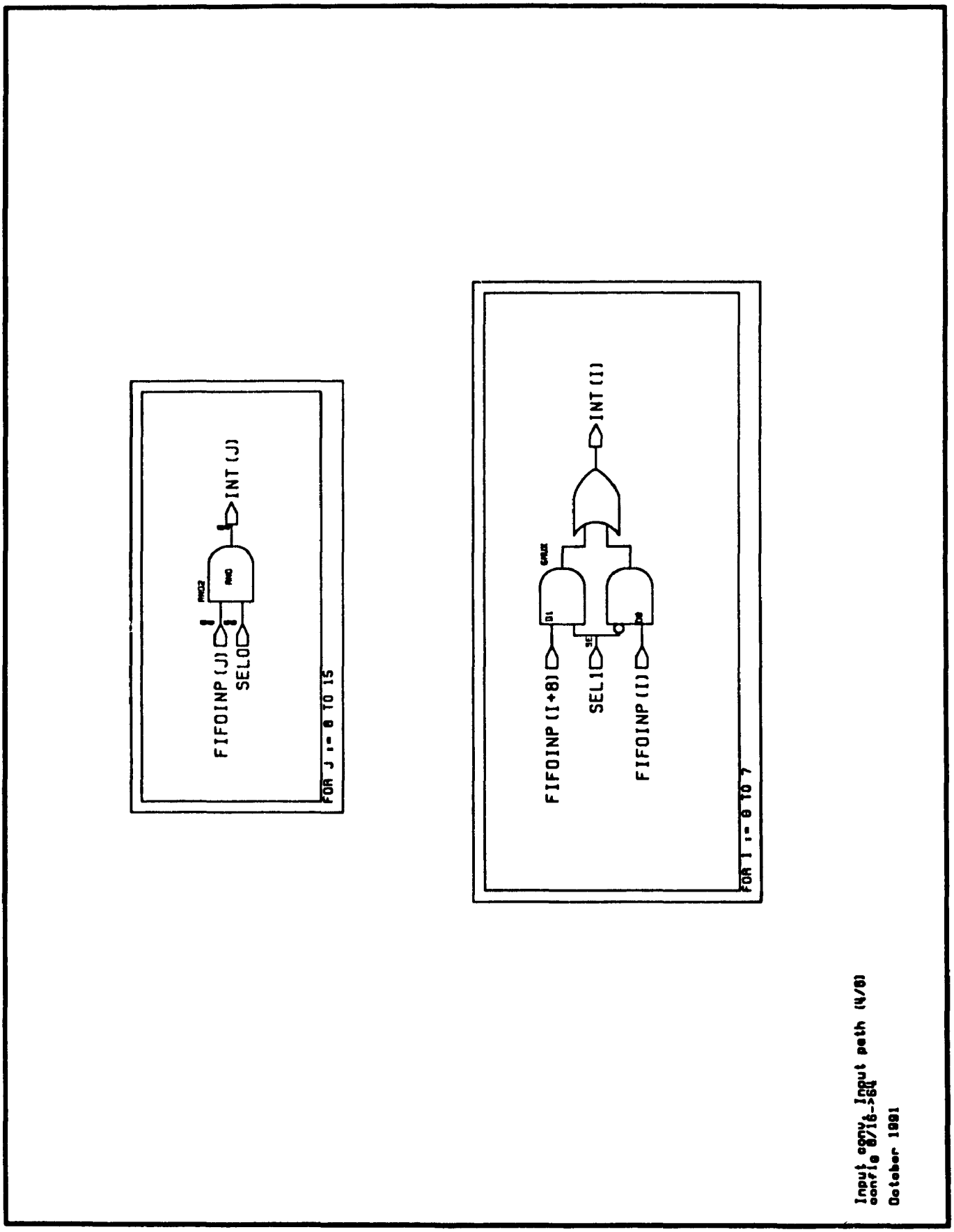


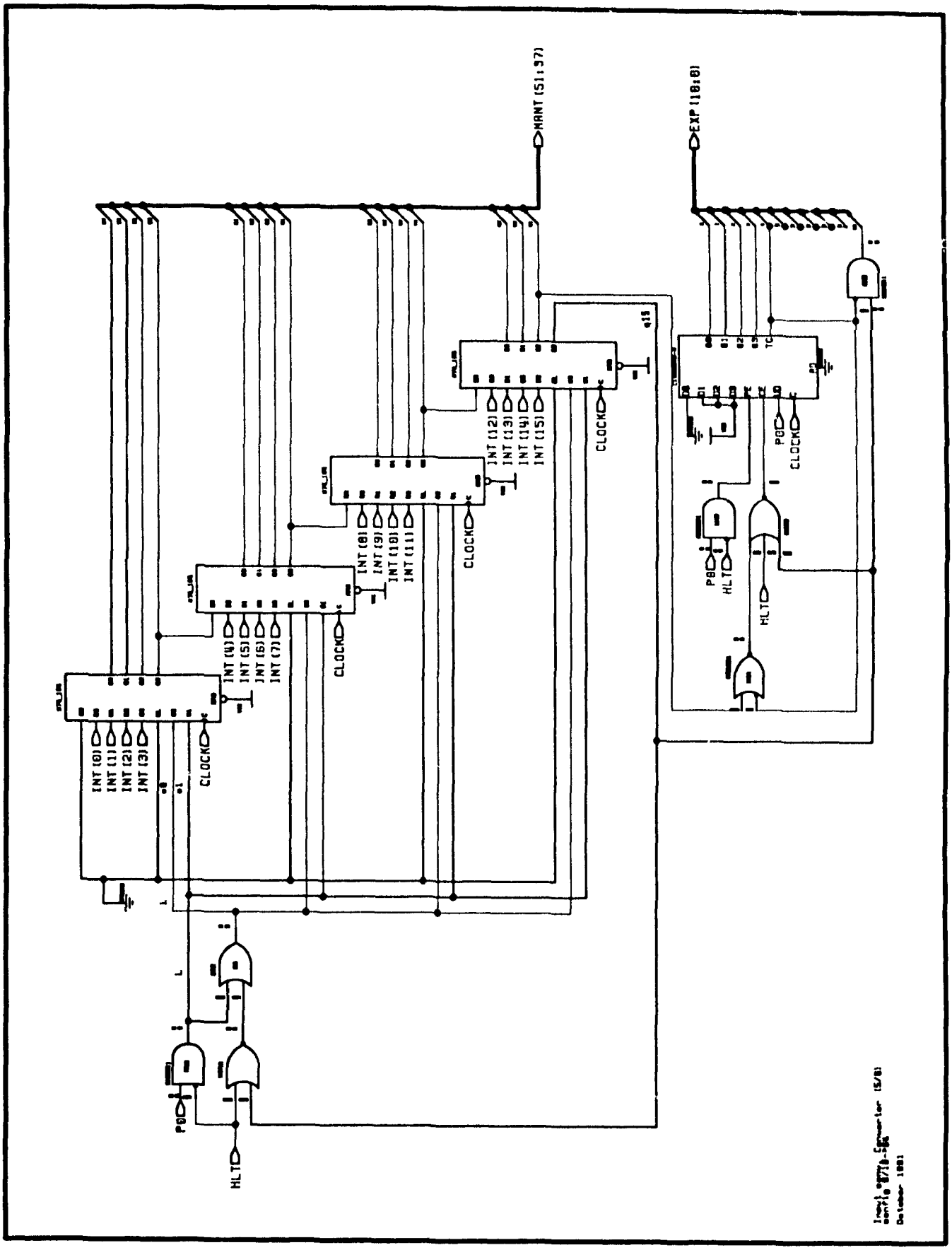
Input Conv. Controller (2/6)
config 8/16->64
October 1991

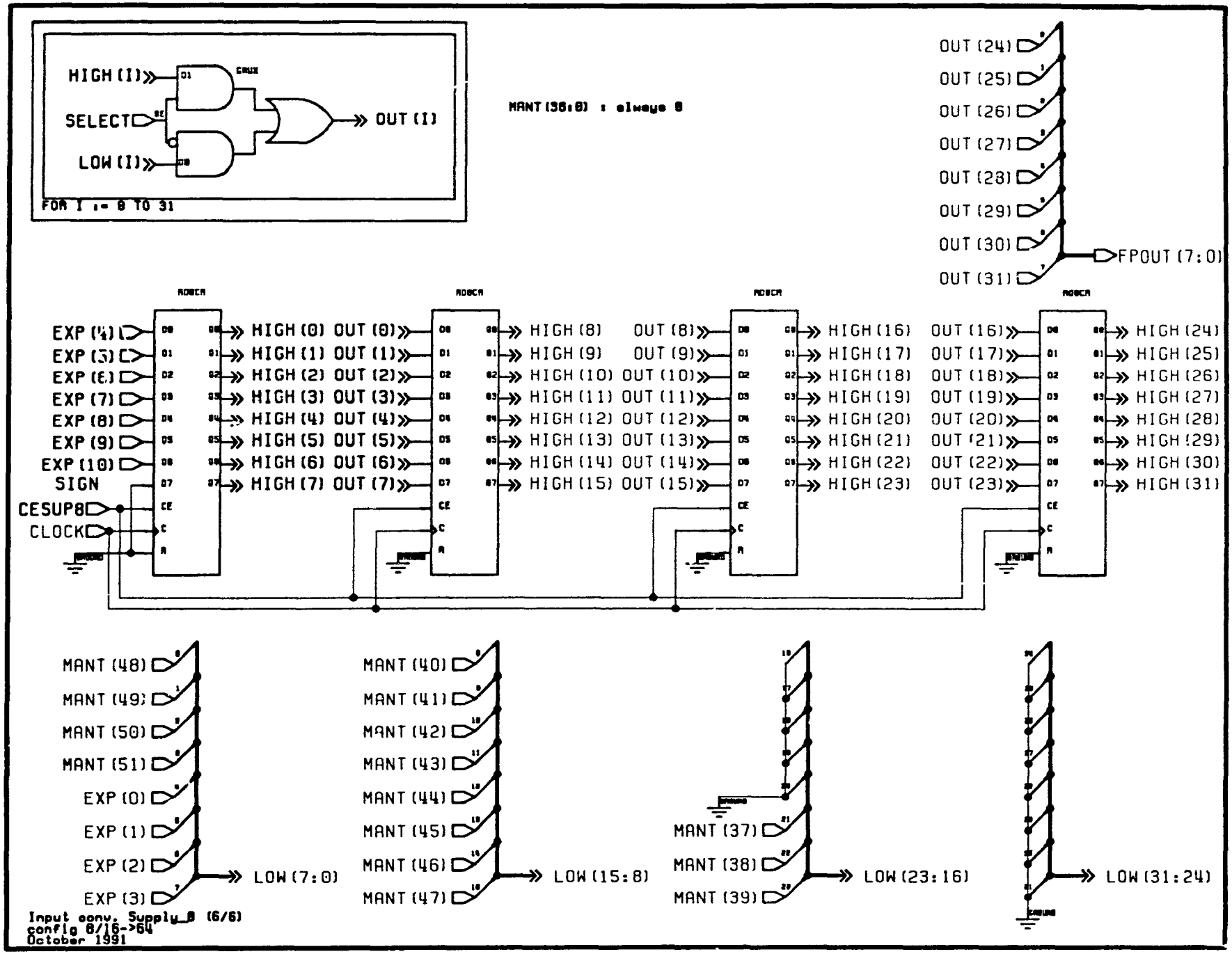


Input conv. Data path (3/6)
config 8/16->64

October 1991

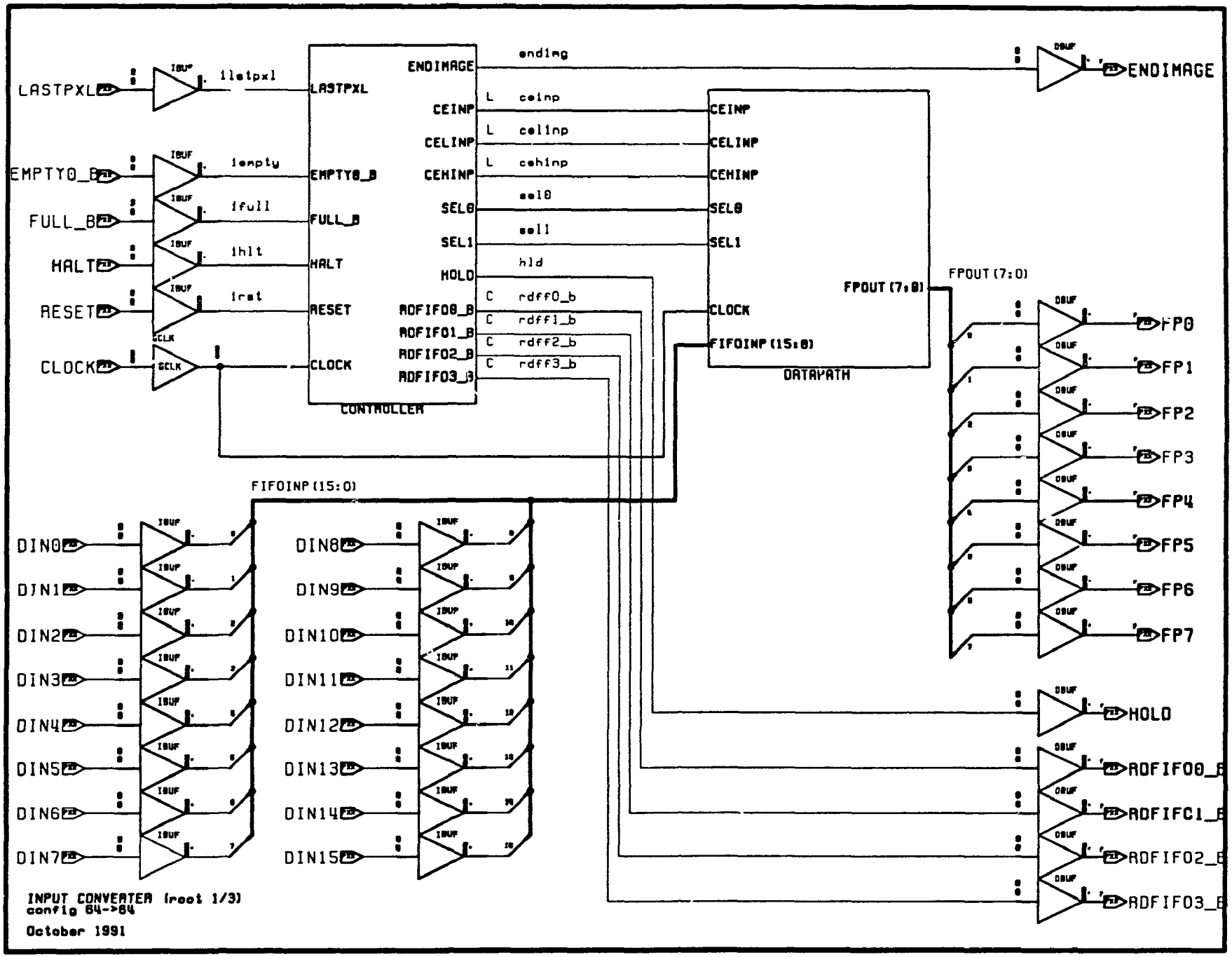

FIFOINP (I+8)
SEL1
FIFOINP (I)
INT (I)
FOR I := 0 TO 7
FIFOINP (J)
SEL0
INT (J)
FOR J := 0 TO 15
Input conv. Input path (4/8)
config 8/16->64
October 1991

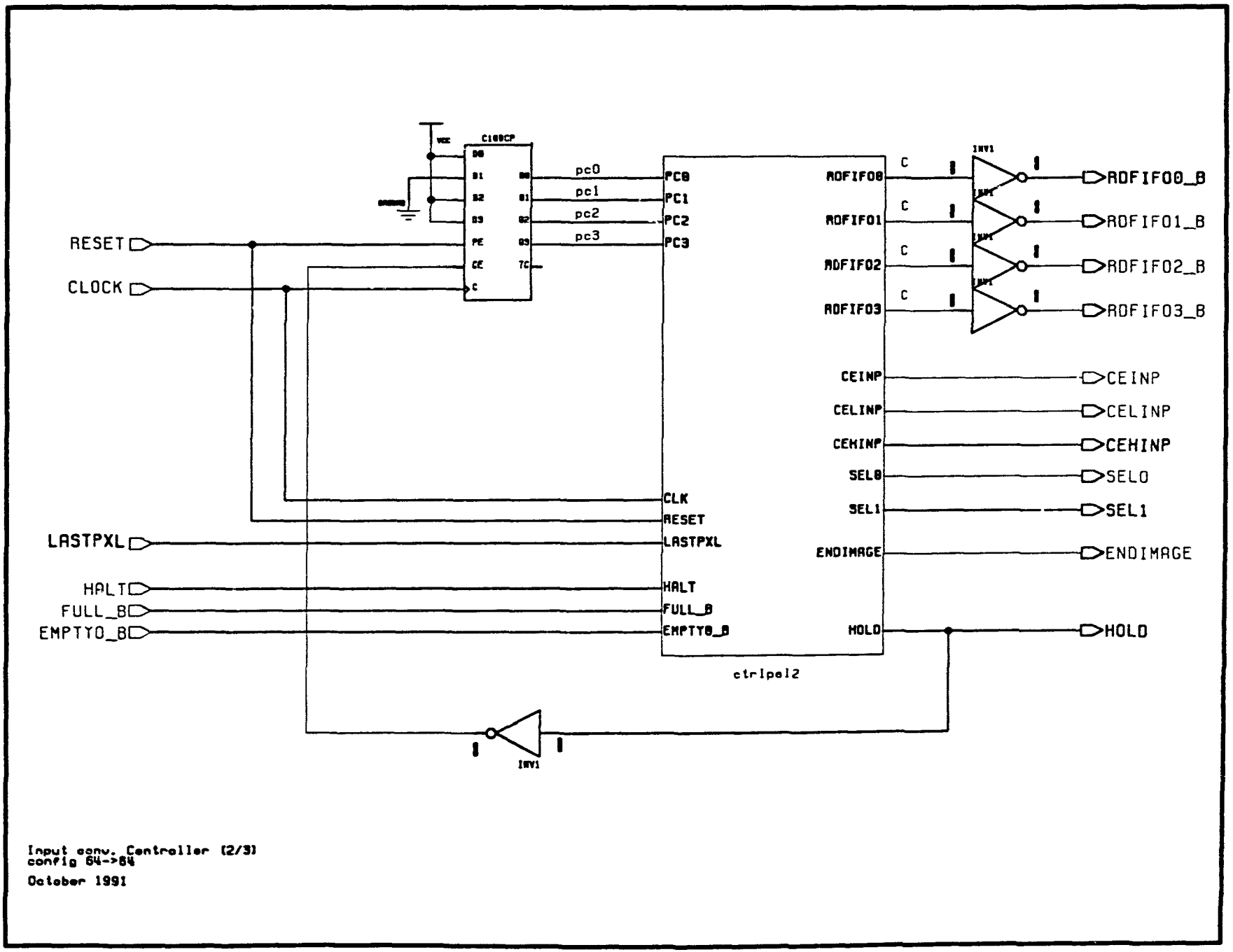

MANT(51:37)
EXP(10:0)
INT(0)
INT(1)
INT(2)
INT(3)
CLOCK
INT(4)
INT(5)
INT(6)
INT(7)
CLOCK
INT(8)
INT(9)
INT(10)
INT(11)
CLOCK
INT(12)
INT(13)
INT(14)
INT(15)
CLOCK
PB
CLOCK
PB
HLT
HLT
L
L
PB
HLT

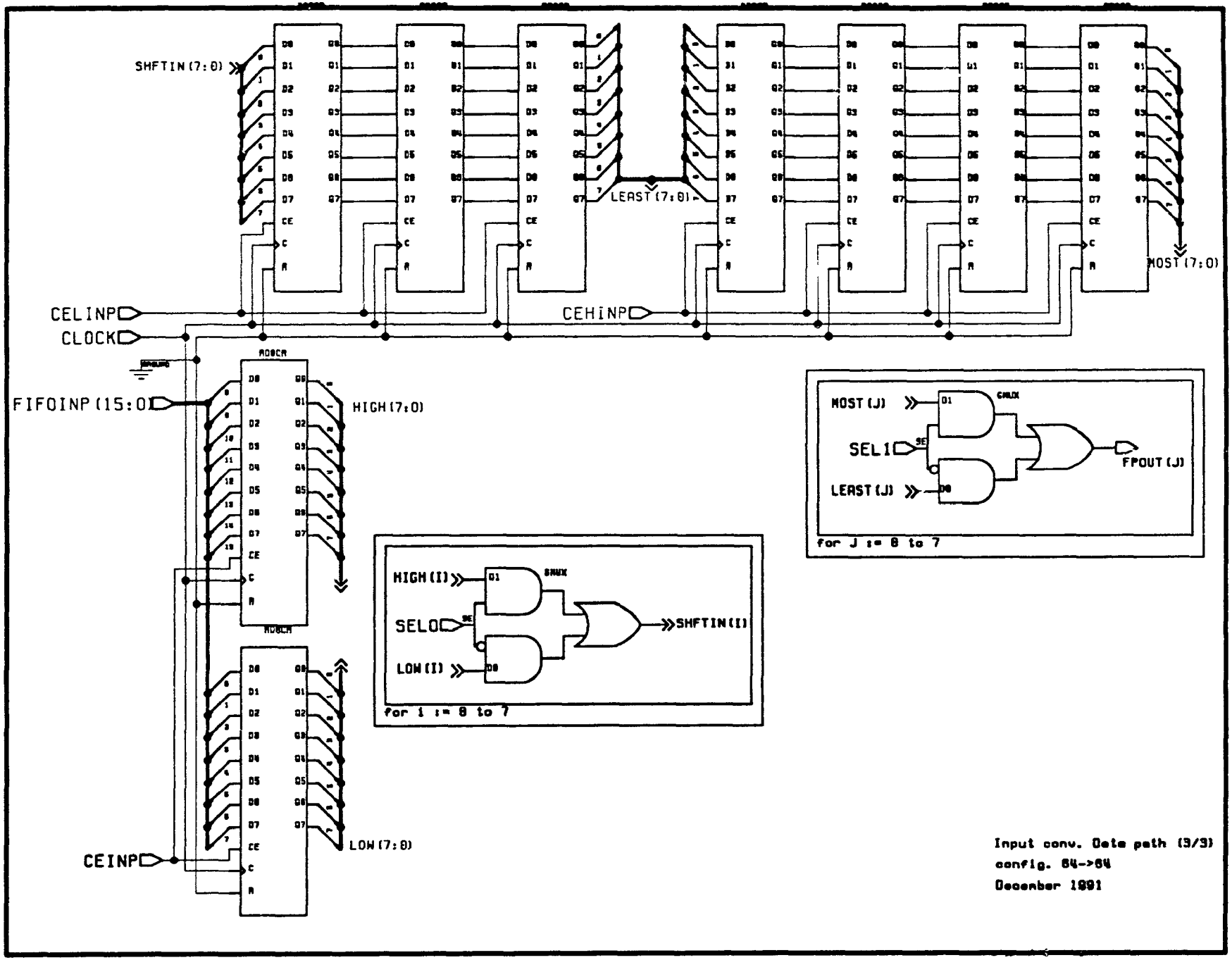

HIGH (I)
SELECT
LOW (I)
OUT (I)
FOR I := 0 TO 31
MANT (38:0) : always 0
OUT (24)
OUT (25)
OUT (26)
OUT (27)
OUT (28)
OUT (29)
OUT (30)
OUT (31)
FPOUT (7:0)
EXP (4)
EXP (5)
EXP (6)
EXP (7)
EXP (8)
EXP (9)
EXP (10)
SIGN
CESUP8
CLOCK
HIGH (0) OUT (0)
HIGH (1) OUT (1)
HIGH (2) OUT (2)
HIGH (3) OUT (3)
HIGH (4) OUT (4)
HIGH (5) OUT (5)
HIGH (6) OUT (6)
HIGH (7) OUT (7)
HIGH (8) OUT (8)
HIGH (9) OUT (9)
HIGH (10) OUT (10)
HIGH (11) OUT (11)
HIGH (12) OUT (12)
HIGH (13) OUT (13)
HIGH (14) OUT (14)
HIGH (15) OUT (15)
HIGH (16) OUT (16)
HIGH (17) OUT (17)
HIGH (18) OUT (18)
HIGH (19) OUT (19)
HIGH (20) OUT (20)
HIGH (21) OUT (21)
HIGH (22) OUT (22)
HIGH (23) OUT (23)
HIGH (24)
HIGH (25)
HIGH (26)
HIGH (27)
HIGH (28)
HIGH (29)
HIGH (30)
HIGH (31)
MANT (48)
MANT (49)
MANT (50)
MANT (51)
EXP (0)
EXP (1)
EXP (2)
EXP (3)
LOW (7:0)
MANT (40)
MANT (41)
MANT (42)
MANT (43)
MANT (44)
MANT (45)
MANT (46)
MANT (47)
LOW (15:8)
MANT (37)
MANT (38)
MANT (39)
LOW (23:16)
LOW (31:24)
Input conv. Supply_8 (6/6)
config 8/16->64
October 1991

```
TITLE                  controlpal pds
AUTHOR                 Jean [illegible]
COMPANY                McGill University
Last revision          December [illegible], [illegible]

CHIP   controlpal    LCA

Input pins
              CLK  [illegible]  MODE  EMPTY_B  FULL_B
              HALT RESET LASTF L
Output pins
              [illegible]  ENDIMAGE  SEL0  SEL1
              DESUPPL,B  SELECT  HOLD  STOP

EQUATIONS
[illegible]
```

[illegible]


LASTPXL
illstpxl
EMPTY0_B
iempty
FULL_B
ifull
HALT
ihlt
RESET
irst
CLOCK
iclk
LASTPXL
EMPTY0_B
FULL_B
HALT
RESET
CLOCK
CONTROLLER
ENDIMAGE
CEINP
CELINP
CEHINP
SEL0
SEL1
HOLD
RDFIFO0_B
RDFIFO1_B
RDFIFO2_B
RDFIFO3_B
ending
ceinp
celinp
cehinp
sel0
sel1
hld
rdff0_b
rdff1_b
rdff2_b
rdff3_b
CEINP
CELINP
CEHINP
SEL0
SEL1
CLOCK
FIFOINP(15:0)
DATAPATH
FPOUT(7:0)
FPOUT(7:0)
ENDIMAGE
FP0
FP1
FP2
FP3
FP4
FP5
FP6
FP7
HOLD
RDFIFO0_B
RDFIFO1_B
RDFIFO2_B
RDFIFO3_B
FIFOINP(15:0)
DIN0
DIN1
DIN2
DIN3
DIN4
DIN5
DIN6
DIN7
DIN8
DIN9
DIN10
DIN11
DIN12
DIN13
DIN14
DIN15
INPUT CONVERTER (root 1/3)
config 64->64
October 1991


C16BCP
pc0
pc1
pc2
pc3
RESET
CLOCK
LASTPXL
HALT
FULL_B
EMPTY0_B
PC0
PC1
PC2
PC3
CLK
RESET
LASTPXL
HALT
FULL_B
EMPTY0_B
ctrlpel2
RDFIF00
RDFIF01
RDFIF02
RDFIF03
C
INV1
RDFIF00_B
RDFIF01_B
RDFIF02_B
RDFIF03_B
CEINP
CELINP
CEHINP
SEL0
SEL1
ENDIMAGE
HOLD
CEINP
CELINP
CEHINP
SEL0
SEL1
ENDIMAGE
HOLD
INV1
Input conv. Controller (2/3)
config 64->64
October 1991


SHFTIN(7:0)
LEAST(7:0)
MOST(7:0)
CELINP
CLOCK
CEHINP
FIFOINP(15:0)
HIGH(7:0)
LOW(7:0)
CEINP
MOST(J)
SEL1
LEAST(J)
FPOUT(J)
for J := 0 to 7
HIGH(I)
SEL0
LOW(I)
SHFTIN(I)
for I := 0 to 7
Input conv. Data path (3/3)
config. 64->64
December 1991

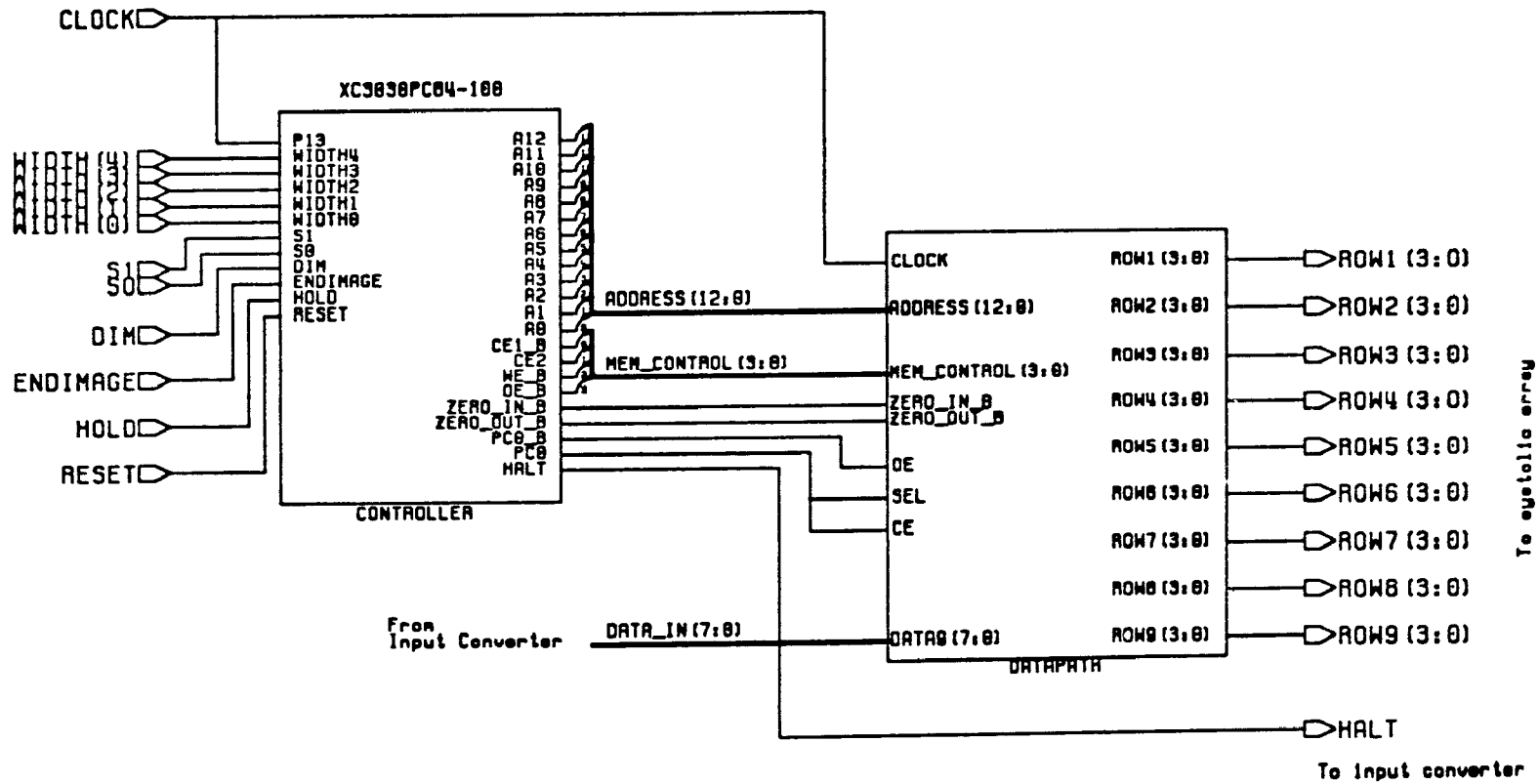
```
TITLE                   ctrlpal2.pds
AUTHOR                  Jean Drolet
COMPANY                 McGill University
LAST REVISION           December 10,1991

CHIP    controlpal      LCA

 INPUT PINS
                        CLK  PC   PC1  PC2  PC3  EMPTY_B  FULL_B
                        HALT RESET LASTPAL
 OUTPUT PINS
                        PCFIFO0  PCFIFO1  PCFIFO2  PCFIFO3  ENDIMAGE  CEINP
                        CELINP  CEHINP  SEL0  SEL1  HOLD

EQUATIONS
[illegible]
```

Appendix B Delay Memory Circuit Schematics

Schematics included in this appendix:

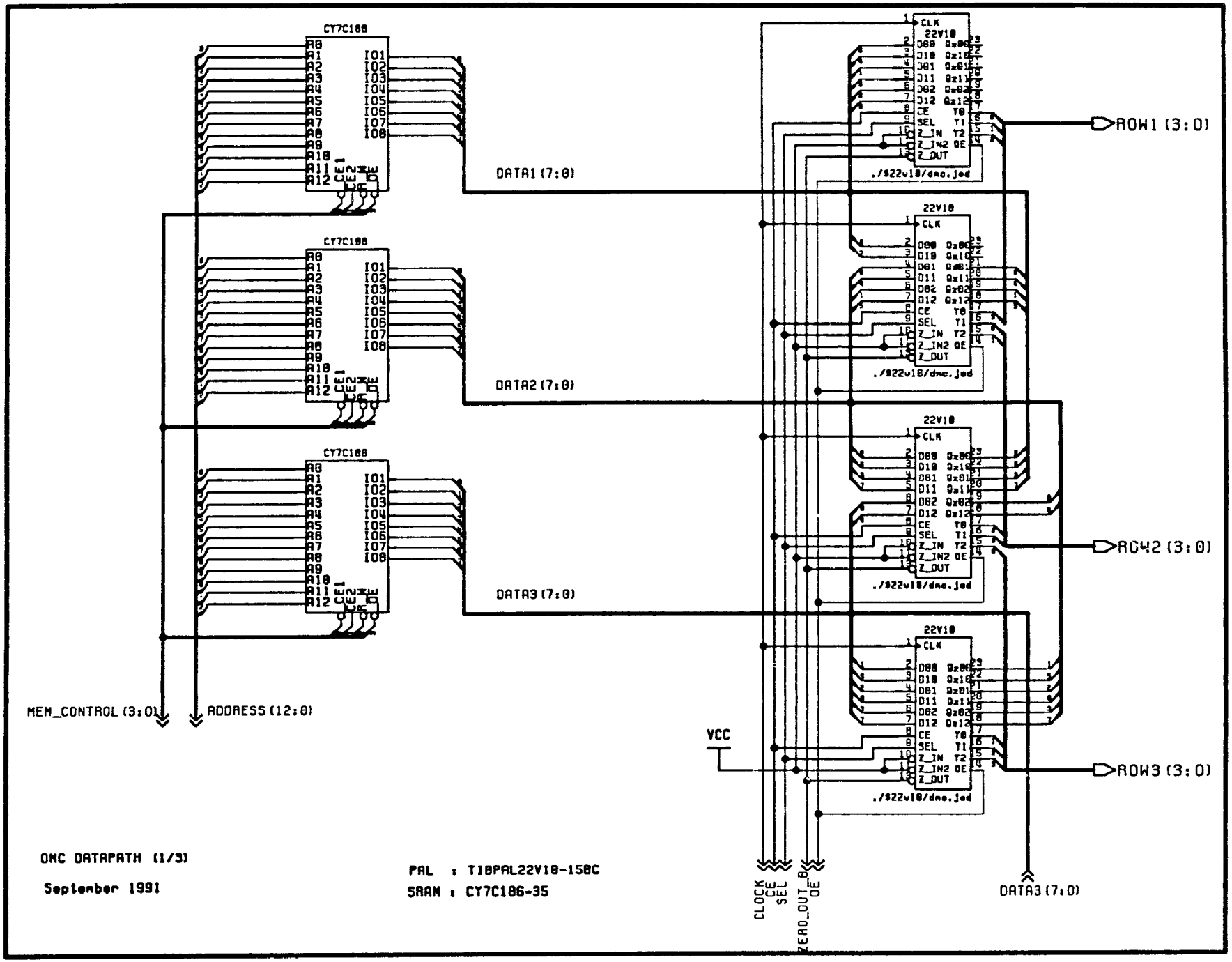
- The delay memory circuit (DMC) top sheet (FullDMC).
- The data path section. Includes 3 sheets (same level of hierarchy) showing 12 PALs 22V10 (see also appendix C) and 8 SRAMs.
- The controller section. Includes 4 sheets plus 1 text file which describes the boolean equations referred to by the *st_mach* block in sheet 2/4.

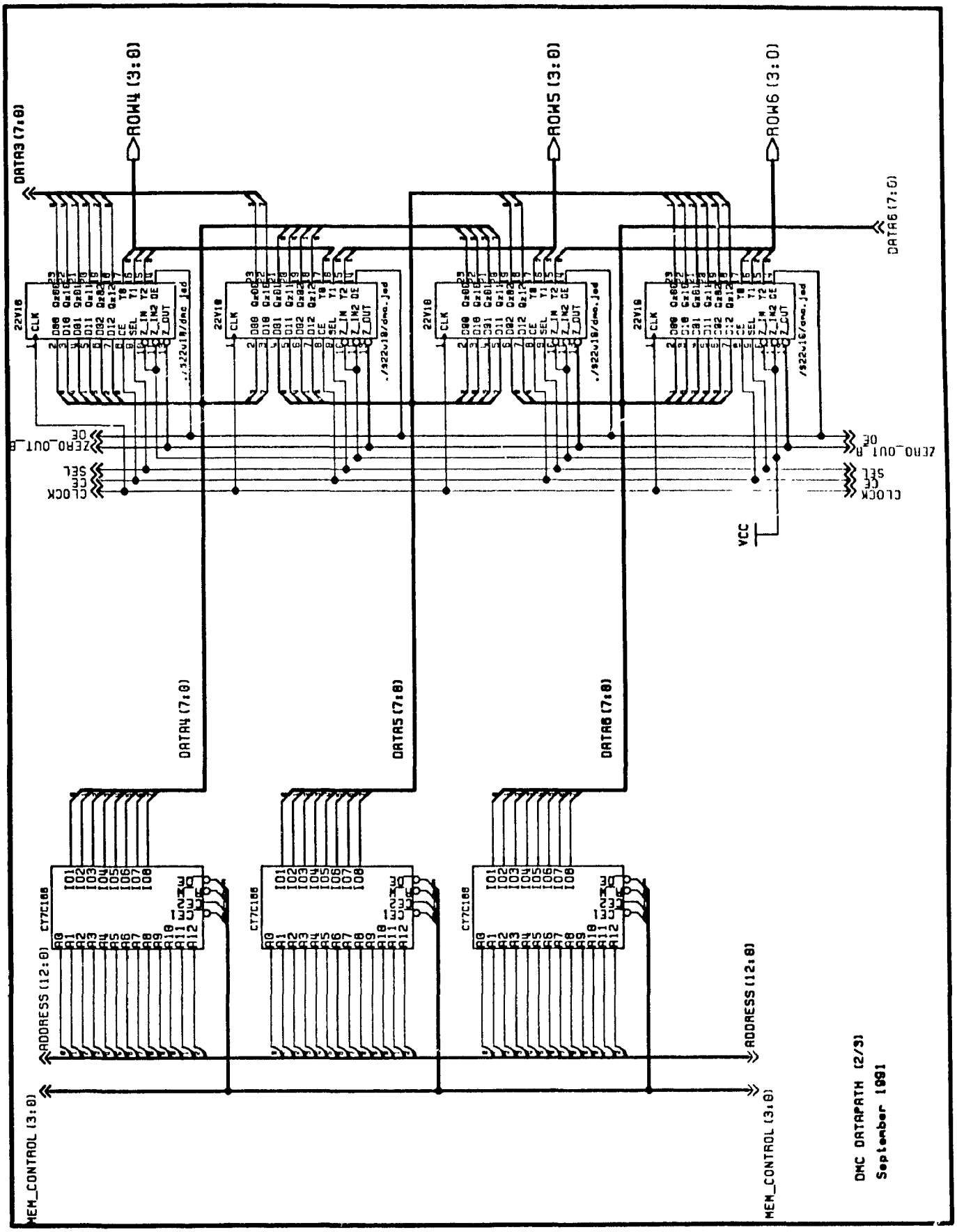
Delay Memory Circuit

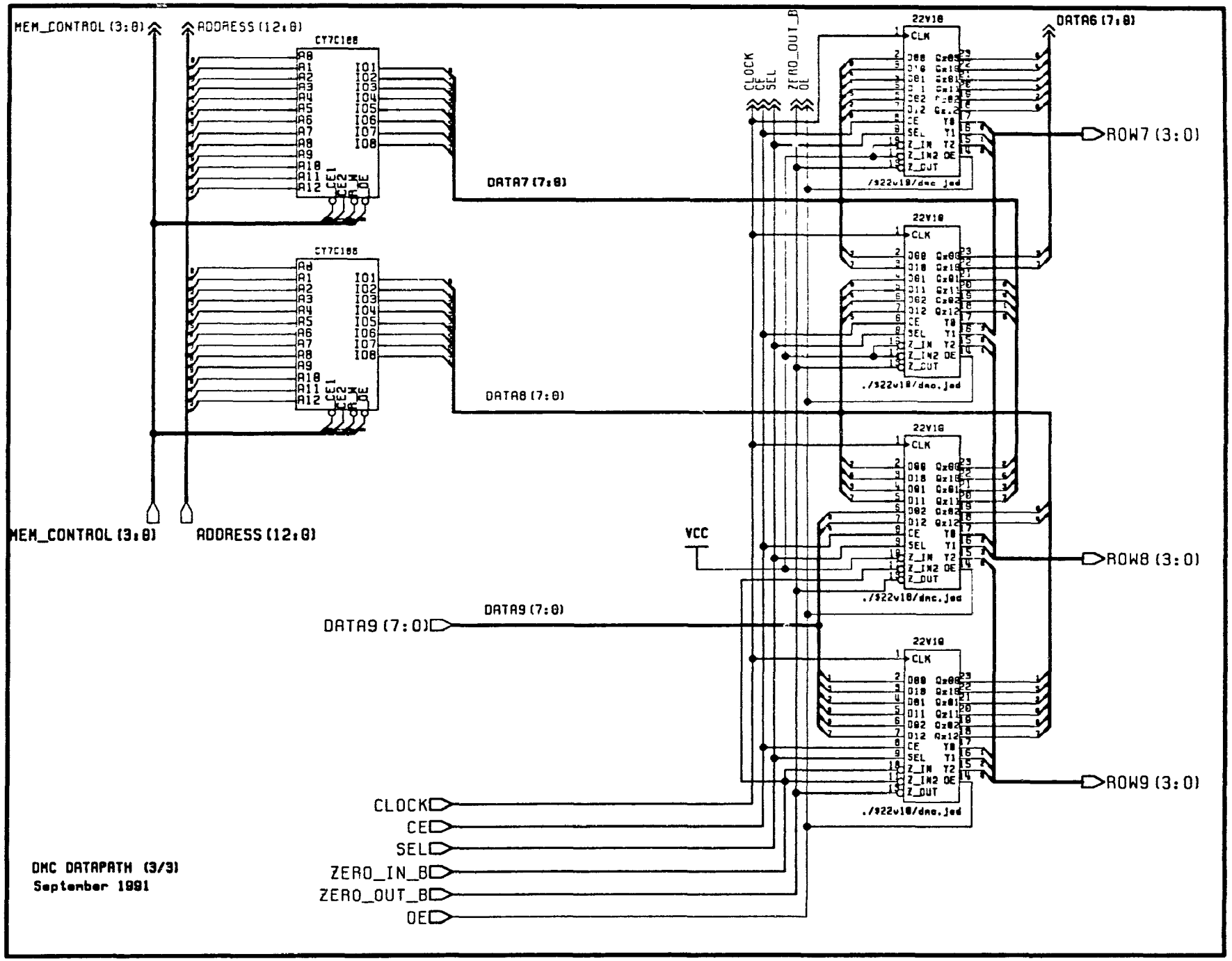


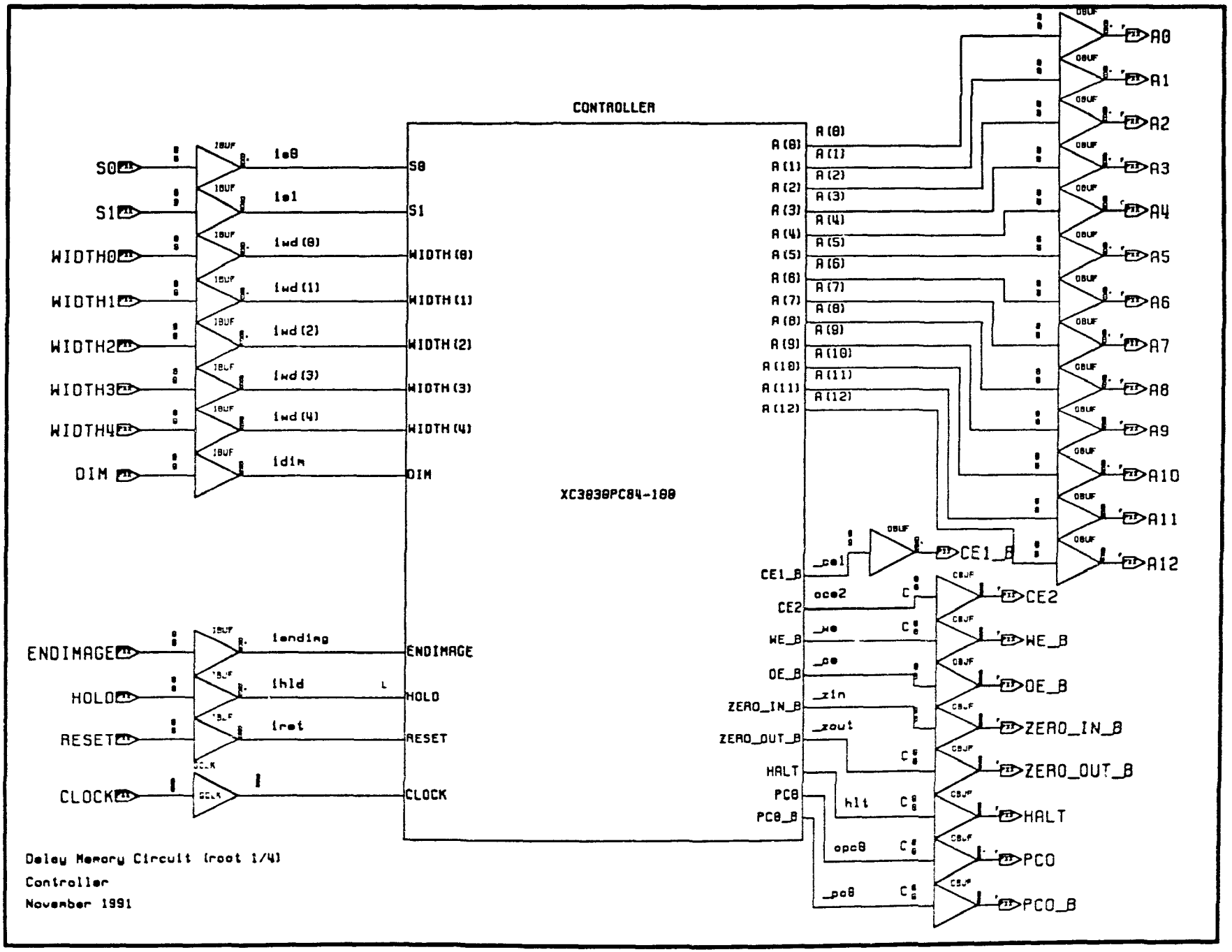
DELAY MEMORY CIRCUIT
(FullDMC) (1/1)

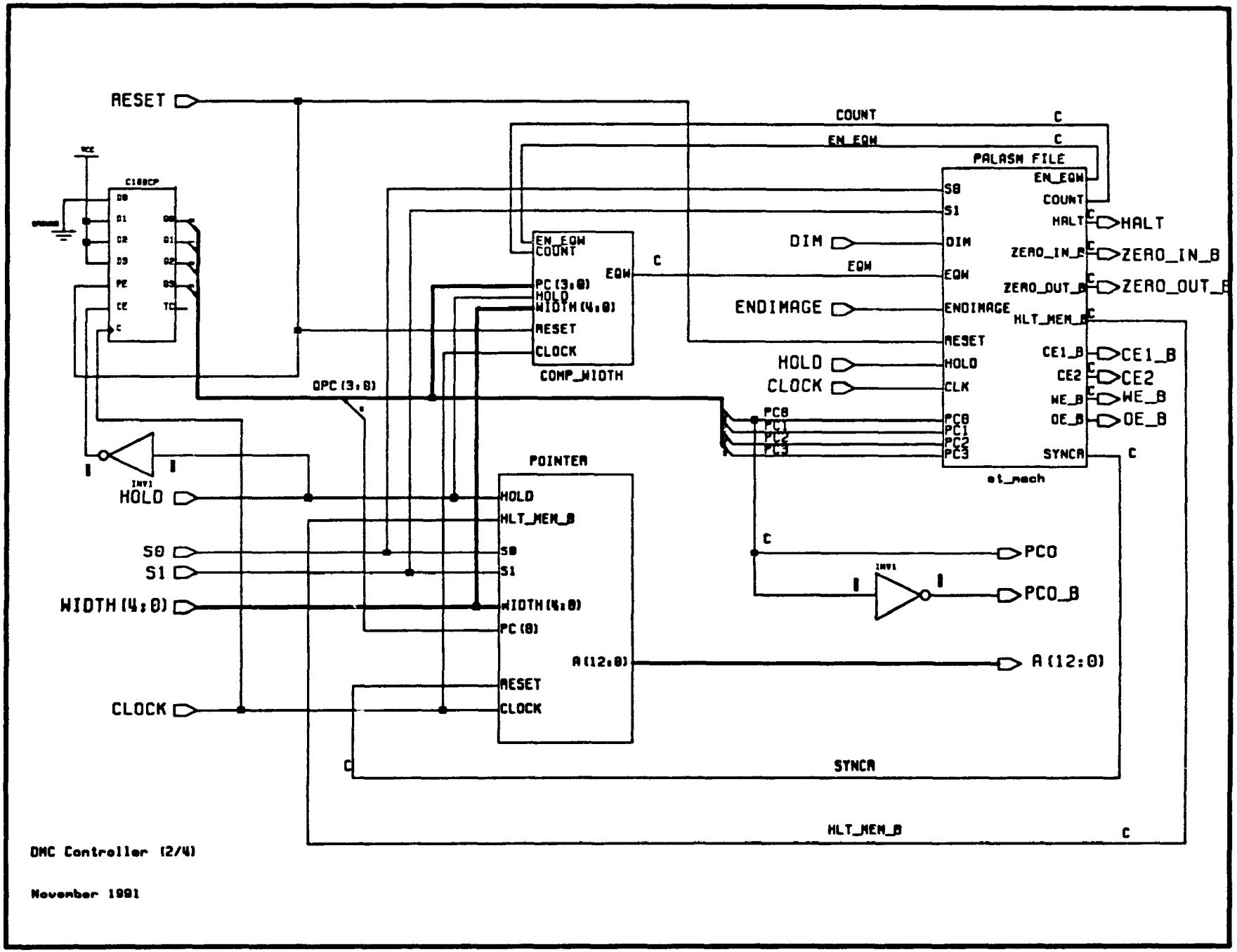
November 1991

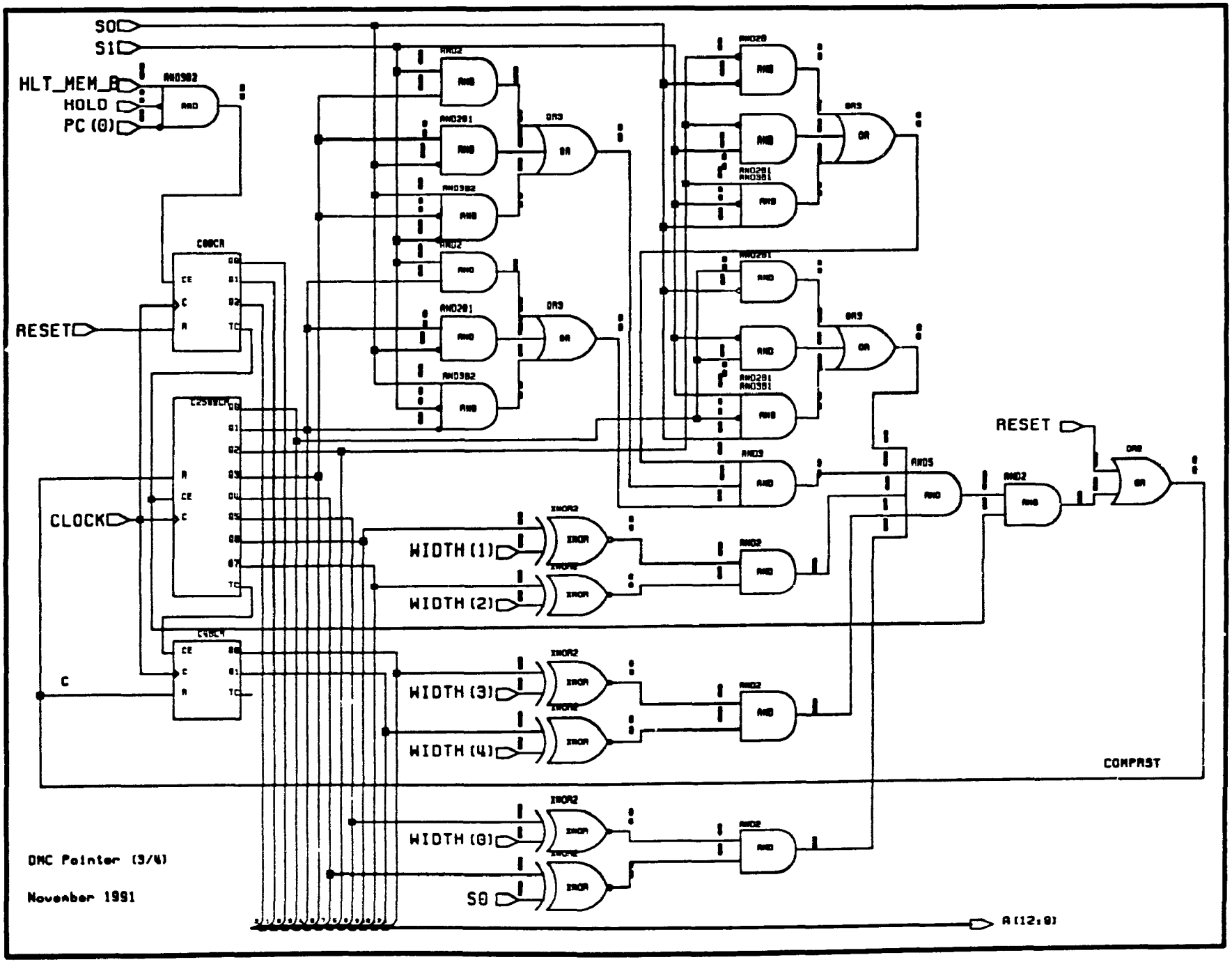

CY7C186
DATA1 (7:0)
DATA2 (7:0)
DATA3 (7:0)
22V18
./922v18/dmc.jed
ROW1 (3:0)
ROW2 (3:0)
ROW3 (3:0)
MEM_CONTROL (3:0)
ADDRESS (12:0)
VCC
CLOCK
CE
SEL
ZERO_OUT_B
OE
DATA3 (7:0)
DMC DATAPATH (1/3)
September 1991
PAL : T18PAL22V18-15BC
SRAM : CY7C186-35

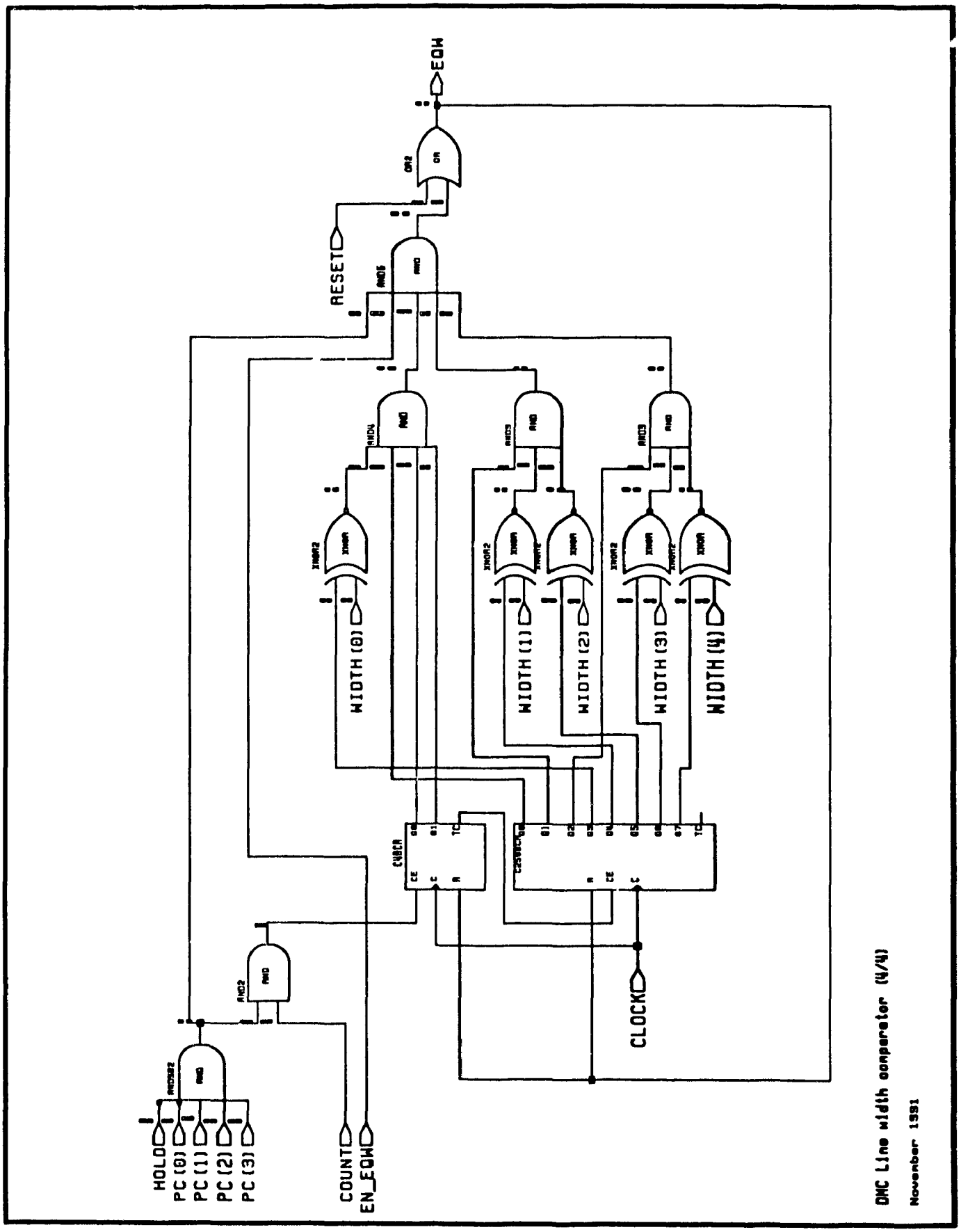

ROW4 (3:0)
ROW5 (3:0)
ROW6 (3:0)
DATA3 (7:0)
DATA6 (7:0)
ZERO_OUT_B
CLOCK
VCC
DATA4 (7:0)
DATA5 (7:0)
DATA6 (7:0)
ADDRESS (12:0)
MEM_CONTROL (3:0)
DMC DATAPATH (2/3)
September 1991


MEM_CONTROL (3:0)
ADDRESS (12:0)
CY7C188
DATA7 (7:0)
DATA8 (7:0)
DATA9 (7:0)
22V10
CLOCK
CE
SEL
ZERO_OUT_B
OE
DATA6 (7:0)
ROW7 (3:0)
ROW8 (3:0)
ROW9 (3:0)
VCC
CLOCK
CE
SEL
ZERO_IN_B
ZERO_OUT_B
OE
DMC DATAPATH (3/3)
September 1991


CONTROLLER
XC3030PC84-100
S0
S1
WIDTH0
WIDTH1
WIDTH2
WIDTH3
WIDTH4
DIM
ENDIMAGE
HOLD
RESET
CLOCK
ie0
ie1
iwd(0)
iwd(1)
iwd(2)
iwd(3)
iwd(4)
idim
iending
ihld
iret
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
CE1_B
CE2
WE_B
OE_B
ZERO_IN_B
ZERO_OUT_B
HALT
PC0
PC0_B
Delay Memory Circuit (root 1/4)
Controller
November 1991


RESET
COUNT
EN_EQW
PALASM FILE
S0
S1
DIM
EQW
ENDIMAGE
RESET
HOLD
CLOCK
HALT
ZERO_IN_B
ZERO_OUT_B
HLT_MEM_B
CE1_B
CE2
WE_B
OE_B
SYNCA
st_mach
COMP_WIDTH
PC(3:0)
WIDTH(4:0)
QPC(3:0)
POINTER
WIDTH(4:0)
PC(0)
A(12:0)
PCO
PCO_B
INV1
C1800P
SYNCA
HLT_MEM_B
DMC Controller (2/4)
November 1991

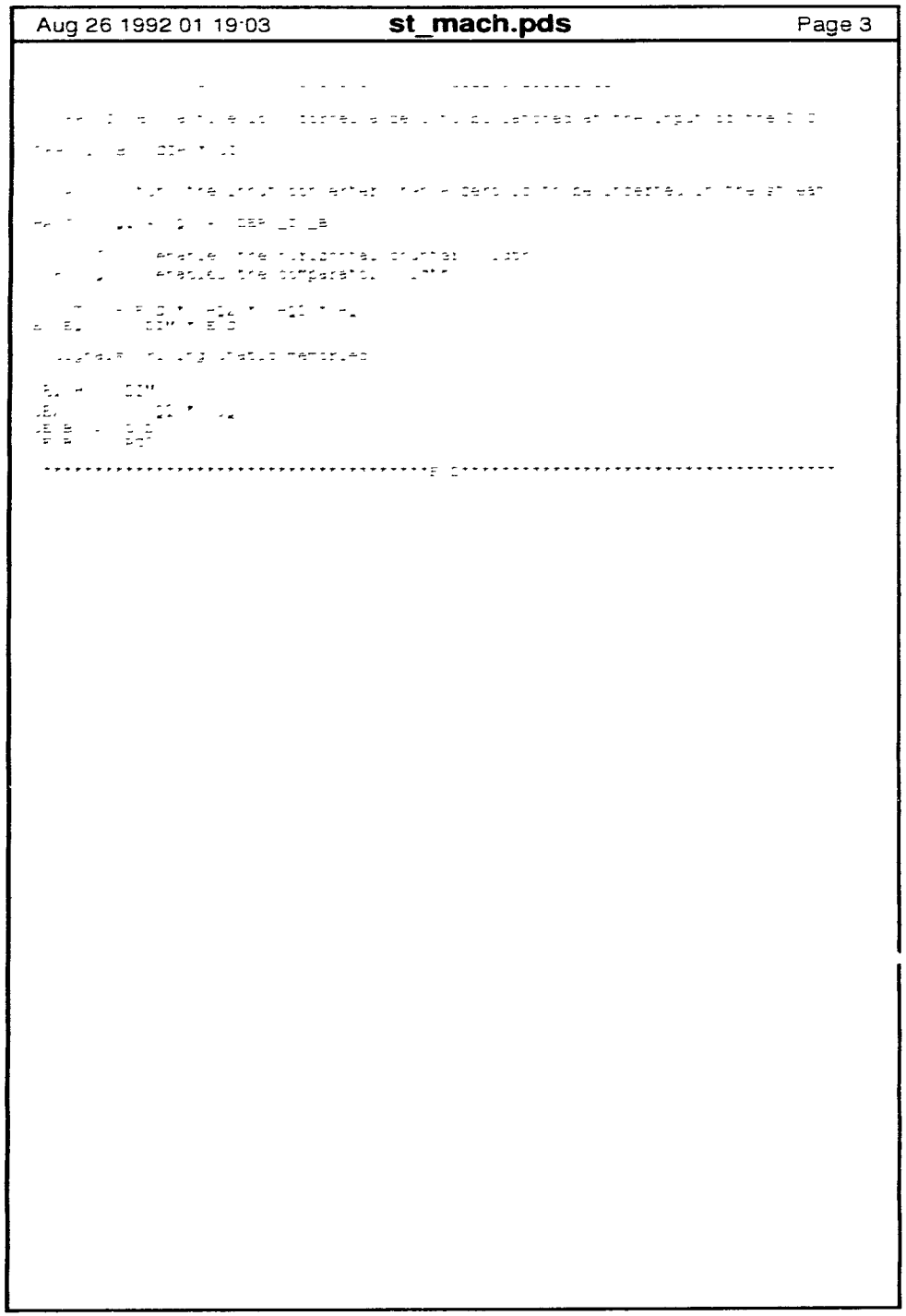

S0
S1
HLT_MEM_B
HOLD
PC (0)
RESET
CLOCK
C
WIDTH (1)
WIDTH (2)
WIDTH (3)
WIDTH (4)
WIDTH (0)
S0
RESET
COMPRST
A(12:0)
DMC Pointer (3/4)
November 1991


EQW
RESET
CLOCK
WIDTH(0)
WIDTH(1)
WIDTH(2)
WIDTH(3)
WIDTH(4)
HOLD
PC(0)
PC(1)
PC(2)
PC(3)
COUNT
EN_EQW
DMC Line width comparator (4/4)
November 1991

```
TITLE                   st_mach.pds
AUTHOR                  Jean Diclet
COMPANY                 McGill University
Last Revised            January 1994

MACH    state machine

Input pins:
          CLK  B1   B0  B2I  B2O  S   S1  DIN  EN   ENDIMAGE  HOLD  RESET

Output pins:
          DEB2_IN_B  DEB2_OUT_B  PLT_MEM_B  HALT  STROBE  DE1_B  DE2  DE_B
          OE_B  OUT  EN_B1

  S1 and S0   set the up sampling mode      S1 S0   mode
                                             0  1
                                             1  0
                                            1
                                            1  1   4

  DIN   set the numerical of the signal being transmitted
        0   no signal image
        1   one signal image


```



```
Output Primary Logic

Output Feedback Logic

Toggle Ping-Pong for input B control
  if B =    => left buffer
  if B =    => right buffer
```

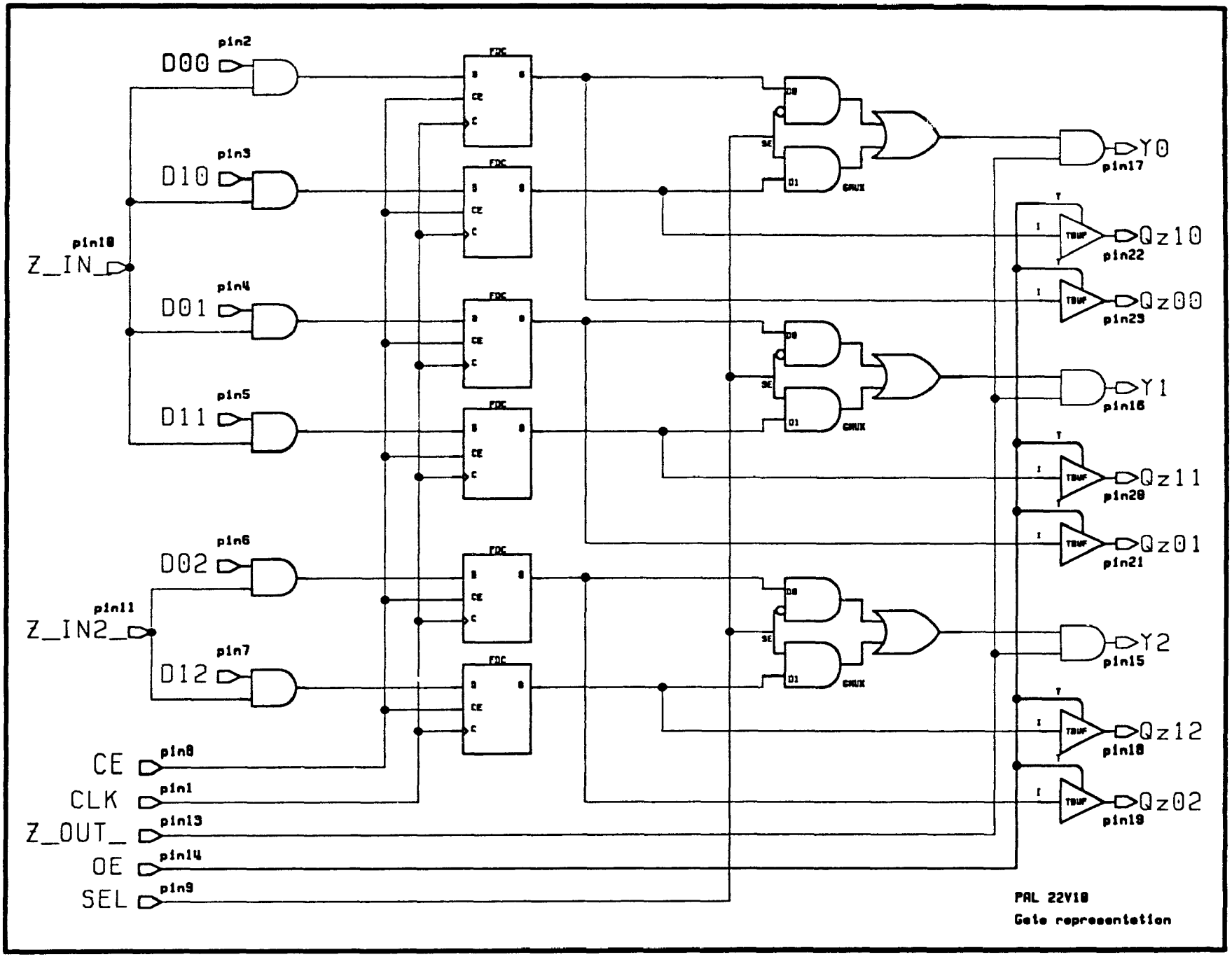


Appendix C PAL 22V10 Description

This appendix includes:

- A schematic of the equivalent gate description.
- The ProLogic description file (dmc.pld) with test vectors.

There are 12 PALs TIBPAL22V10 in the delay memory circuit datapath section. Each PAL implements 6 flip-flops, 3 2to1 multiplexers, a few AND gates and 6 tri-state lines. In addition, one PAL provides a maximum of 3 outputs lines (Y0, Y1 and Y2) to be connected to the systolic array. Since one row of the systolic array is 4 bits wide, it takes 1 1/3 PALs to implement a complete row. Six tri-state lines also provide a connection to the data bus of the SRAM in the previous row.


D00
pin2
D10
pin3
Z_IN_
pin18
D01
pin4
D11
pin5
D02
pin6
Z_IN2_
pin11
D12
pin7
CE
pin8
CLK
pin1
Z_OUT_
pin13
OE
pin14
SEL
pin9
FDC
GMUX
TBUF
Y0
pin17
Qz10
pin22
Qz00
pin23
Y1
pin16
Qz11
pin20
Qz01
pin21
Y2
pin15
Qz12
pin18
Qz02
pin19
PAL 22V10
Gate representation

```
[illegible]          [illegible]ice          TIBPAL[illegible]
                     Application       Dele[illegible] [illegible] data pa[illegible] for [illegible]
                     Author            Jean [illegible] [illegible] 1991

include [illegible]              * [illegible] TIBPAL[illegible] *

[illegible]                      * [illegible] input pins *
[illegible]
define [illegible]
define [illegible]
define [illegible]
define [illegible]
define [illegible]
define [illegible]
define [illegible]
define [illegible]
define [illegible]
[illegible] D_IN2 [illegible]
define [illegible] pin1
define [illegible] pin14
[illegible]                      * define output pins *
[illegible]
[illegible]
[illegible]  SEL
[illegible]  SEL
[illegible]  SEL
[illegible]
```



```
[illegible]
... H L L I
... L L L
... H [illegible]

test_vectors
[illegible] SEL D_OUT [illegible]
[illegible]  L L L
[illegible]  H L H
[illegible]  L L L
[illegible]  H H L

test_vectors

[illegible] OE OE D_IN D_IN2 [illegible]
[illegible]

test_vectors
[illegible] SEL D_OUT [illegible]
[illegible]
```

Appendix D Timing Simulations

This appendix presents some simulation traces from Mentor Graphics' Quicksim. Selected cases have been included for the input converter and the delay memory circuit

D.1 Configuration 1 of Input Converter

Figure D.1 illustrates an example of conversion. In this case, a 16-bit hexadecimal number, A9E3, is converted into a 64-bit floating-point number. First, A9E3 appears on bus DIN at pipeline cycle F (330 ns). The conversion takes place during the next 16 clock cycles. Then, on the subsequent pipeline cycles, the result appears in chunks of 8 bits on bus FPOUT The first four bytes are always zero, but the remaining 4 bytes, 60, BC, E5 and 40, appear on clock cycles 5, 8, A and C, respectively Note that the DMC latches the content of bus FPOUT every other clock cycle starting with the least significant bytes first. The result of the conversion is therefore 40E5 BC60 0000 0000.

D.2 Configuration 2 of Input Converter

Figure D.2 shows the behavior of the transparent configuration. In this case, the 64-bit floating-point number FEDC BA98 7654 3210 is read 16 bits at a time during the fist pipeline cycle in the following order : BA98, FEDC, 3210 and 7654. In the next pipelme cycle, the same number appears on bus FPOUT byte after byte starting with the least significant.

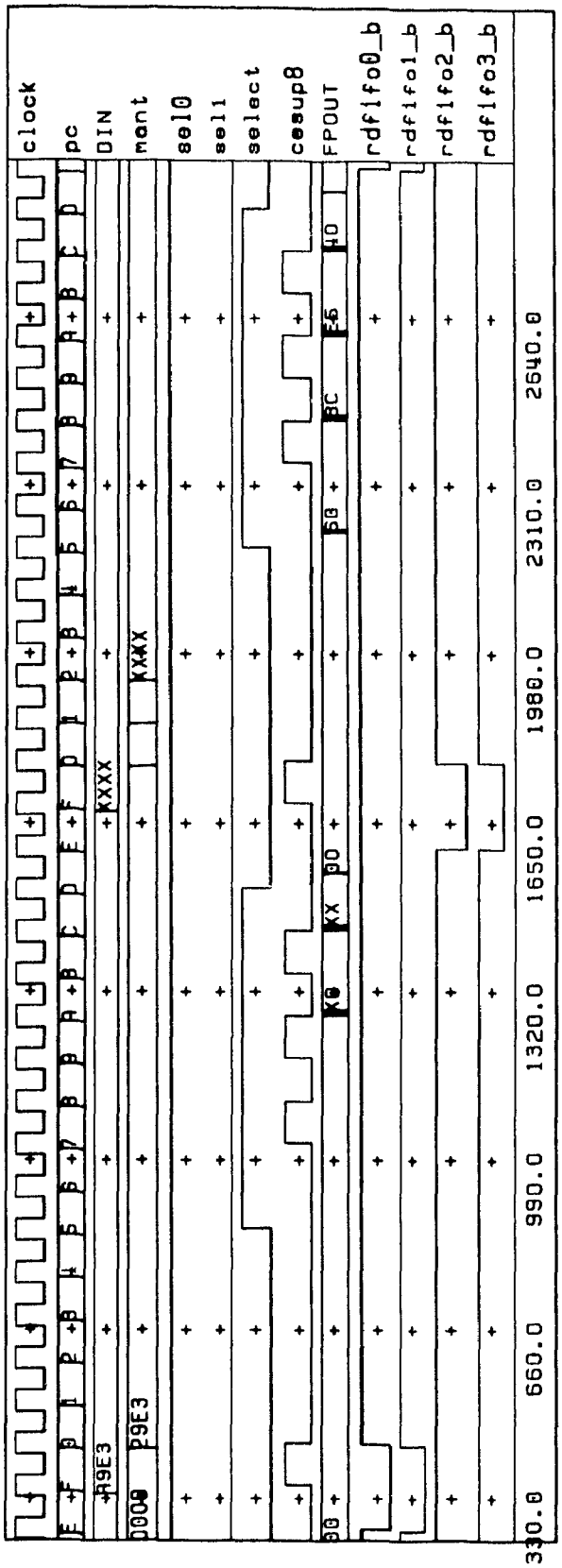


Figure D.1: Conversion of A9E3 into a 64-bit F.P. number

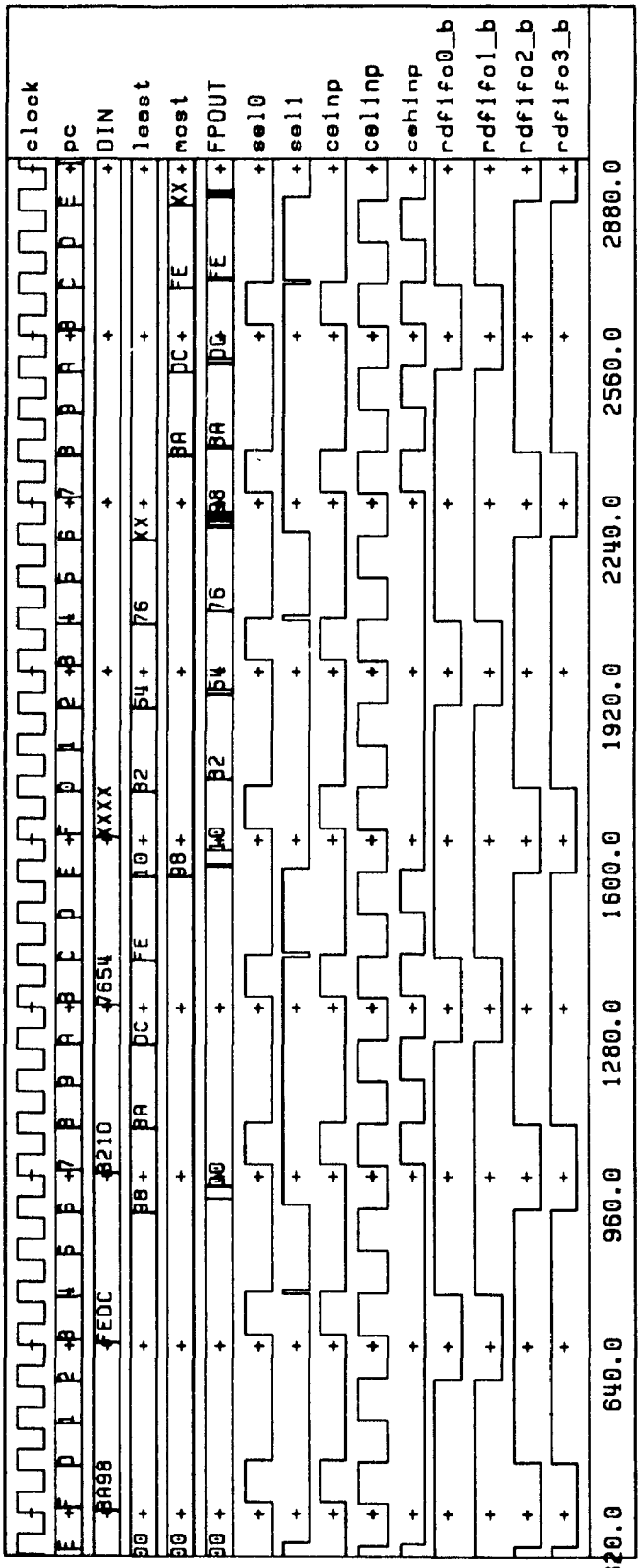


Figure D.2: Reordering of the hex number FEDC BA98 7654 3210

D.3 Delay Memory Circuit: Memory Access

This simulation run illustrates how data circulate in the DMC (2-D mode). As shown in figure D.3, value 32, which is a partial result coming from the input converter, appears on bus DATA9 during cycle B. At the same time, a read cycle is taking place. During memory access, an unknown value XX appears temporarily on bus DATA8 until the number FF is retrieved from memory. Upon rising edge of the clock (cycle C), value 32 is latched into the flip-flops and propagated to bus DATA8 for storage into the memory of row 8. It should be clear that, on odd clock cycles, bus DATA8 is driven by the output of memory and, on even clock cycles, this bus is driven by the output of the latches in row 9. Signals ROW9 and ROW8 are 4-bit buses connected to the systolic array inputs. During even clock cycles they hold the lower 4 bits of the bytes just latched and, during odd clock cycles, they hold the upper 4-bits.

D.4 Delay Memory Circuit: Control

Figure D.4 shows the trace of a few controller signals when the DMC operates in 2x up-sampling mode. In the first half of the simulation run, signal ZERO IN B is activated to insert two zero-valued pixels for the left border in the incoming data stream. The first zero is stored in memory locations 0 to 7 while the second is stored in locations 8 to F. The two remaining zero-valued pixels are not stored in memory since they will be automatically inserted in the outgoing streams by the up-sampler.

Signals V, H and U represent the current state of each of the three state machines already described in chapter 4. Notice that the U (horizontal up-sampling) state machine alternates between state 0 (insert normal pixel) and state 1 (insert null pixel for up-sampling). When U = 1, the HALT signal is driven high to stop the

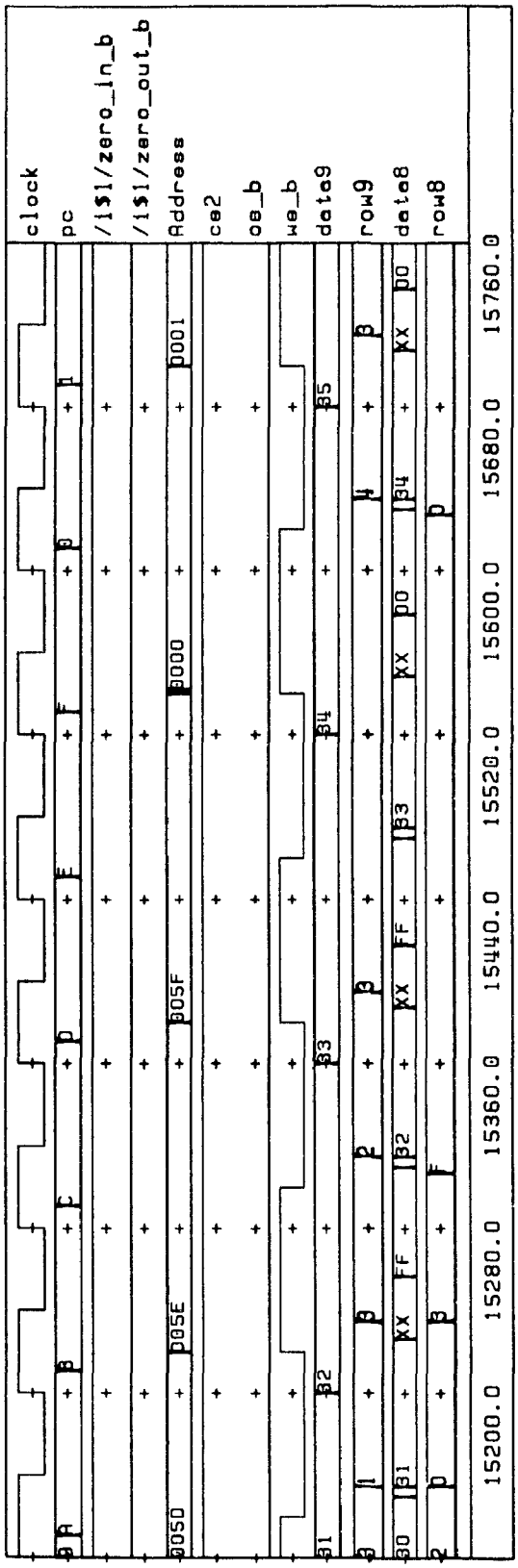


Figure D.3: DMC memory access

input converter during up-sampling and signal ZERO_OUT B is driven low to force insertion of null pixels in the outgoing streams. When U = 0, the ADDRESS bus is incremented 8 times to store in the delay memory the incoming floating-point value.

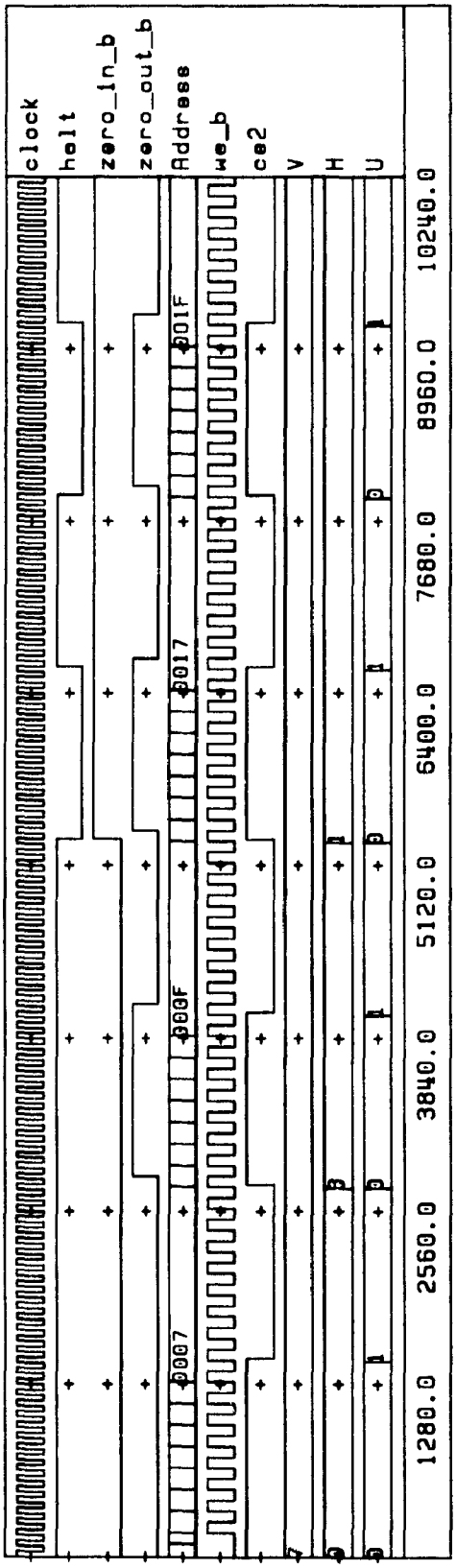


Figure D.4: DMC controller signals (2x up-sampling)