National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

VLSI Architecture for Discrete Wavelet Transform

By

Aleksander Grzeszczak

A Thesis Submitted to
the School of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the Degree of
Master of Applied Science
in Electrical Engineering

Ottawa-Carleton Institute for Electrical Engineering

Department of Electrical Engineering
Faculty of Engineering
University of Ottawa

ISBN 0-612-04924-8

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

I hereby declare that I am the sole author of this thesis.

I authorize the University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Aleksander Grzeszczak

I further authorize the University of Ottawa to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Aleksander Grzeszczak

Abstract

In this thesis, we present a new simple and efficient VLSI architecture (DWT-SA) for computing the Discrete Wavelet Transform. The proposed architecture is systolic in nature, modular and extendible to 1-D or 2-D DWT transform of any size. The DWT-SA has been designed, simulated and implemented in silicon.

The following are the features of the DWT-SA architecture:

- It has an efficient (close to 100%) hardware utilization.
- It works with data streams of arbitrary size.
- The design is cascadable, for computation of one, two or three dimensional DWT.
- It requires a minimum interface circuitry on the chip for purposes of interconnecting to a standard communication bus.

The DWT-SA design has been implemented using CMOS 1.2 um technology.

Acknowledgement

First of all, I would like to express my gratitude and appreciacion to my supervisors, Dr. T. H. Yeap and Dr. S. Panchanathan. Their guidance, encouragement and support were the driving force behind the succesful completion of my thesis work. Special thanks to Dr. Martin Snelgrove from Carleton University for his valuable comments, suggestions and help.

Thanks to the staff of the Department of Electrical Engineering, University of Ottawa, for their cheerful help.

The financial support from the Microelectronics Network (Micronet) under the Network Centers of Excellence (NCE) program of the Government of Canada is gratefully acknowledged.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, there has been an increasingly important requirement to address the bandwidth limitations over communication networks. The advent of broadband networks (ISDN, ATM, etc.) [1,2] as well as compression standards such as JPEG, MPEG, H.261, etc. is an attempt to overcome that limitation. Typical applications for compression, include high definition television, video conferencing, multimedia communications, etc. [3,4]. There are two kinds of redundancies in a video sequence, namely: spatial and temporal. The spatial correlations are usually removed using intra-frame techniques such as discrete cosine transform [5], vector quantization [6], discrete wavelet transform, etc. whereas the temporal redundancies are removed by using inter-frame techniques such as motion estimation/compensation [7,8].

The desirable characteristics for an image or video compression system are as follows:

- High performance image/video compression algorithms where there is a

graceful degradation in the quality with increasing compression.

- Low computational complexity algorithms with efficient parallel hardware (VLSI) implementations for real-time applications.

In JPEG and MPEG standards, Discrete Cosine Transform (DCT) is used for intra-frame compression. However, DCT suffers from the negative effects of blockiness and mosquito noise [5] resulting in poor subjective quality of reconstructed images at high compression (low transmission bit rates). Recently wavelet transform has emerged as a powerful technique for achieving compression. Wavelet based techniques possess the following features:

- Basis functions match the human visual profiles resulting in high quality of reconstructed images.
- Flexibility in representing non-stationary image signals.
- Multiresolution representation which allows direct scalable access to the data or any sub-set of the data.
- Low computational complexity, O(n).
- Superior objective and subjective performance[9,10,11].
- Efficient parallel VLSI implementation.

1.1 Thesis Motivation

The main reason for the interest in efficient VLSI architectures for DWT is the intensive computational requirements for its calculation. The designs proposed in literature range from very basic, almost abstract architectures that do not go past the "black box" level, to those that are detailed and posses a lot of merit. None of them have ever been implemented in silicon.

There is only one chipset, designed by Aware Inc. for implementing DWT. However, this is a complex design requiring extensive user control. Programming such a device is therefore tedious, difficult, time consuming and could be difficult to troubleshot. There is a clear need for designing and implementing a DWT chipset that explores the potential of DWT particularly in the area of decomposition algorithm and hardware implementation and that operates in a turnkey fashion. Here, the user is required to input only the data stream and the high-pass and low-pass filter coefficients.

1.2 Thesis Objective

The objective of this thesis is to design and implement in silicon an efficient architecture for computing DWT. The following are the features of the proposed design:

- It has an efficient (close to 100%) hardware utilization.
- It works with data streams of arbitrary size.
- The design is cascadable, for computation of one, two or three dimensional DWT.
- It requires a minimum interface circuitry on the chip for purposes of interconnecting to a standard communication bus.

1.3 Thesis Organization

The thesis is organized as follows: A brief literature review of Discrete Wavelet Transform and architectures for its computation are presented in Chapter 2. The proposed architecture for computing Discrete Wavelet Transform (DWT) in real-time is detailed in Chapter 3. The architecture is systolic in nature and performs all coefficient calculations with only one set of multipliers. In addition, the architecture is simple, modular, and easily extendible to transforms of any block size.

Chapter 4 presents circuit level design of basic DWT-SA cells, and the analog and digital simulation results. A number of audio and video applications in which the proposed chipset can be employed are detailed in Chapter 5. Appendix-A contains a complete set of gate level schematics of the DWT-SA architecture.

1.4 Main Contributions

The main contributions of this thesis are:

- Design of a simple, modular and cascadable architecture for real-time implementation of DWT.
- Implementation of the architecture as a DWT chipset using Cadence software.

Chapter 2

Background and Literature Review

This chapter presents a brief introduction to Discrete Wavelet Transform (DWT) and a review of existing architectures for DWT. Recently, several VLSI architectures for computing DWT have been proposed in literature. The proposed architectures vary from simple and abstract implementations of DWT to complex architectures whose potential for real-time implementation has been demonstrated by system level simulations. In addition, the only commercially available VLSI chipset: the WTP processor by AWARE Inc. is introduced. The results of cross-comparison between the proposed architectures are also presented.

2.1 Description of the DWT

DWT decomposes a signal **x(n)** into a weighted sum of small "wave-like" basis functions, called wavelets. The basis functions can be generated by dilating and translating the "mother" wavelet. DWT extracts information from the signal at different scales. The

first level of wavelet decomposition extracts the details of the signal (high frequency components) while the second and all subsequent wavelet decompositions extract progressively coarser information (low frequency components). In practice, this decomposition is performed only for a certain number of stages. To illustrate the operation, consider an image (two dimensional input data). The 2-D DWT decomposes the image recursively into a "low-pass" subimage and three "high-pass" subimages. The low-pass subimage which retains spatial information looks like a sub-sampled version of the original image. The high-pass sub-images contain only edges and fine texture information. This process is shown in Figure 2.1.

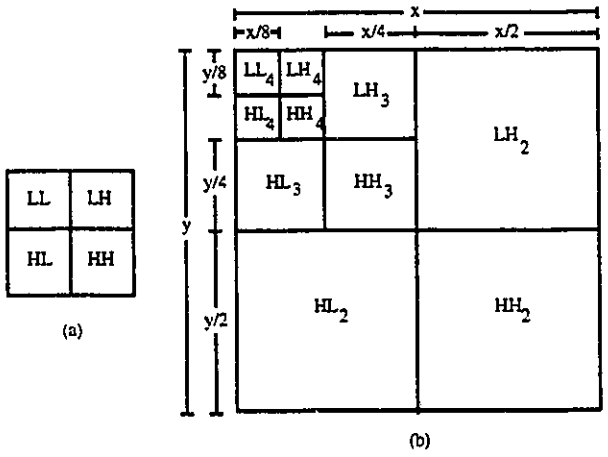


Figure 2.1. (a) Wavelet transform decomposition of an image into 4 sub-images.
(b) A three level image decomposition.

Generally in image processing, compactly supported wavelets are used. The coefficients are calculated from a few neighboring pixels and thus the corresponding "high" or "low" pass coefficients preserve all spatial information of the original image. This feature of DWT makes it suitable for use with advanced image processing techniques.

2-D DWT transform is an extension of the 1-D transform and hence can be easily

obtained from the 1-D transform computation. From now on, we only concern ourselves with the 1-D DWT computation which is illustrated by the following equations:

$$W_L(n,j) = \sum_{m=0}^{2n} W(m, j-1)h(2n-m) \tag{2.1a}$$

$$W_H(n,j) = \sum_{m=0}^{2n} W(m, j-1)g(2n-m) \tag{2.1b}$$

where $W(n,0)=x(n)$, $h(n)$ and $g(n)$ are Quadrature Mirror Filters derived from the wavelet. In other words, it is a multiresolution decomposition of a sequence of length N, which generates two output sequences: the "low pass" and the "high pass" series of total length N.

DWT can be viewed as a special case of sub-band coding [12], where the DWT operation, is executed by two filters derived from the wavelet, namely a high-pass filter H, and the corresponding low-pass filter L. This algorithm, universally known as the Pyramid Algorithm, is illustrated in Figure 2.2.

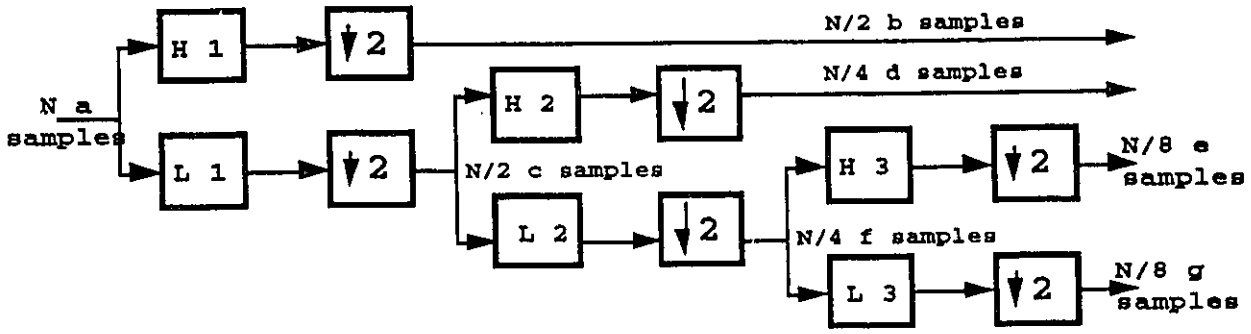


Figure 2.2. Sub-based coding filter bank (Pyramid Algorithm).

2.2 Computational Complexity of the DWT

The Pyramid Algorithm performs the following computations on the incoming stream of data: The data is passed through the high-pass (H1) and low-pass (L1) filters, followed by subsampling by a factor of two. The H1 filter calculates the DWT output values for the first octave. The L1 filter outputs are fed back into the filter bank in order to generate the next octave output values. This process is continued recursively. As shown in equation 2.2, the frequency of computed samples decreases exponentially. N/2 samples are generated at the highest resolution, N/4 samples at the next highest resolution and so on until two samples remain at the lowest resolution level: one low-pass and one high-pass. Including intermediate samples, a total of:

$$N+\frac{N}{2}+\frac{N}{4}+\ldots+1+1=2(N-1) \tag{2.2}$$

filtering operations or convolutions are executed.

2.3 Data Dependencies within DWT

DWT computation is complex because of the data dependencies at different octaves. The set of equations 2.4a to 2.4n, for which variables *a, b, c, d, e, f, g,* have been defined in Figure 2.2, represent a complete calculation of a three octave DWT transform. Data dependencies at various octaves are represented by identical symbols used in more than one octave equation. The transfer functions for a sixth order, non recursive FIR digital filter is given by the following equation, where the low-pass and high-pass components are as follows:

$$High(z) = g_0 + g_1z^{-1} + g_2z^{-2} + g_3z^{-3} + g_4z^{-4} + g_5z^{-5} \tag{2.3a}$$

$$Low(z) = h_0 + h_1z^{-1} + h_2z^{-2} + h_3z^{-3} + h_4z^{-4} + h_5z^{-5} \tag{2.3b}$$

1st octave:

$$b(0) = g_0a(0) + g_1a(-1) + g_2a(-2) + g_3a(-3) + g_4a(-4) + g_5a(-5) \quad (2.4a)$$

$$b(2) = g_0a(2) + g_1a(1) + g_2a(0) + g_3a(-1) + g_4a(-2) + g_5a(-3) \quad (2.4b)$$

$$b(4) = g_0a(4) + g_1a(3) + g_2a(2) + g_3a(1) + g_4a(0) + g_5a(-1) \quad (2.4c)$$

$$b(6) = g_0a(6) + g_1a(5) + g_2a(4) + g_3a(3) + g_4a(2) + g_5a(1) \quad (2.4d)$$

$$c(0) = h_0a(0) + h_1a(-1) + h_2a(-2) + h_3a(-3) + h_4a(-4) + h_5a(-5) \quad (2.4e)$$

$$c(2) = h_0a(2) + h_1a(1) + h_2a(0) + h_3a(-1) + h_4a(-2) + h_5a(-3) \quad (2.4f)$$

$$c(4) = h_0a(4) + h_1a(3) + h_2a(2) + h_3a(1) + h_4a(0) + h_5a(-1) \quad (2.4g)$$

$$c(6) = h_0a(6) + h_1a(5) + h_2a(4) + h_3a(3) + h_4a(2) + h_5a(1) \quad (2.4h)$$

2nd octave:

$$d(0) = g_0c(0) + g_1c(-2) + g_2c(-4) + g_3c(-6) + g_4c(-8) + g_5c(-10) \quad (2.4i)$$

$$d(4) = g_0c(4) + g_1c(2) + g_2c(0) + g_3c(-2) + g_4c(-4) + g_5c(-6) \quad (2.4j)$$

$$e(0) = h_0c(0) + h_1c(-2) + h_2c(-4) + h_3c(-6) + h_4c(-8) + h_5c(-10) \quad (2.4k)$$

$$e(4) = h_0c(4) + h_1c(2) + h_2c(0) + h_3c(-2) + h_4c(-4) + h_5c(-6) \quad (2.4l)$$

3rd octave:

$$f(0) = g_0e(0) + g_1e(-4) + g_2e(-8) + g_3e(-12) + g_4e(-16) + g_5e(-20) \quad (2.4m)$$

$$g(0) = h_0e(0) + h_1e(-4) + h_2e(-8) + h_3e(-12) + h_4e(-16) + h_5e(-20) \quad (2.4n)$$

As shown in Figure 2.2 and Equations 2.4, several intermediate results (c, f) are first computed, and then used to calculate multiple output samples. We note that the intermediate results must be stored and made available for further processing at specific time instants.

The rest of this chapter is organized as follows; We first, present a detailed overview of the DWT architectures available in the literature. Section 2.4 presents the basic Parallel Filter architecture (PF), which is a high level, abstract implementation of the Pyramid Algorithm. It also summarizes two other DWT architectures similar to the PF

architecture namely: the SIMD Linear Array architecture (LA) and the SIMD Multigrid Architecture (MA). Section 2.5 describes in detail two, complex approaches for VLSI implementation of DWT. In Section 2.6, we present the block based architecture for computation of 2-D DWT. The details of the only commercially available DWT chipset: the WTP processor from AWARE Inc. are outlined in Section 2.7. A comparison of the DWT architectures is shown in Section 2.8.

2.4 General DWT Architectures

2.4.1 Parallel Filter Architecture

The parallel filter architecture for 1-D DWT as presented in [13,14], is a very general, high level, abstract implementation of the Pyramid Algorithm. This architecture calculates the outputs of the first octave every second sample period, and computes all higher octave outputs between the first octave output calculations. The main component of this architecture (as in most other architectures) consists of two filter banks dedicated for computing the high-pass and the low-pass DWT outputs, respectively. The computation of J octave outputs is performed using a storage unit of size $L*J$, where L is the size of the support of the basic wavelet h. The block diagram of this architecture for computing J octaves of coefficients is presented in Figure 2.3. The parallel filters consist of L fixed point multipliers and L-1 adders. The storage unit, is a bank of J shift registers, each of length L, where the ith shift register ($1 \leq i \leq J$) holds the required inputs for the ith octave calculation. The outputs are written into the ith shift register every 2^{i-1} clock cycles and the L elements of the ith shift register are fed into the parallel filters every 2^i clock cycles. The control signals $\{c_i\}$ and $\{a_i\}$which are generated at specific time instants, put the inputs in or out of the shift registers. The execution time for the architecture is of order $O(N)$, and the pipeline delay is also of order $O(N)$.

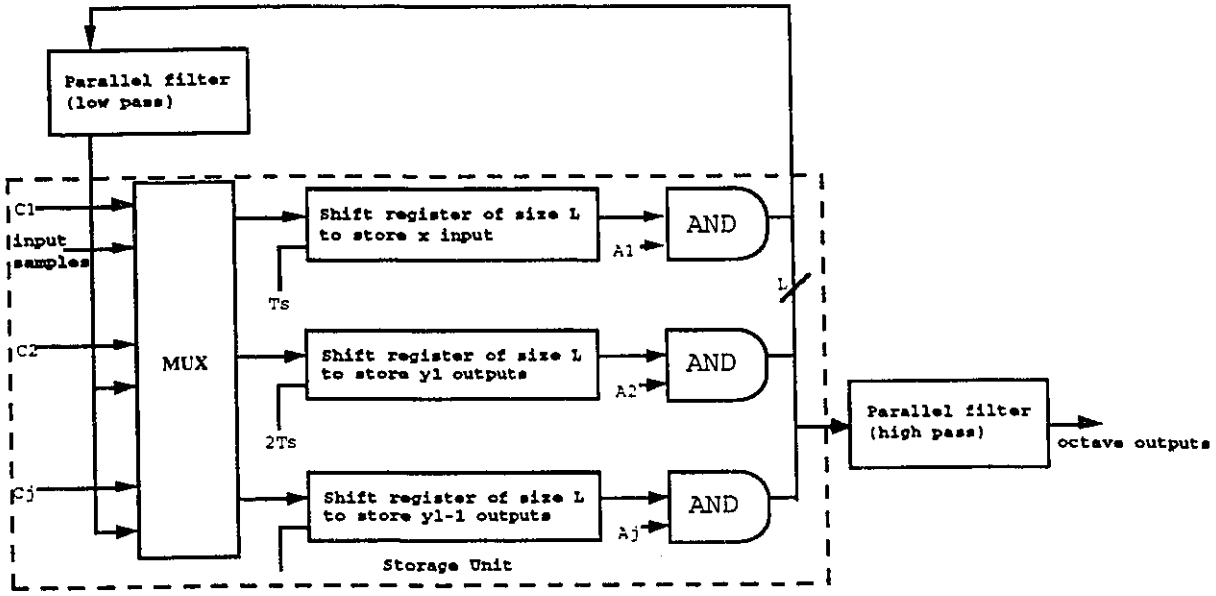


Figure 2.3. Parallel filter architecture for 1-D DWT.

2.4.2 SIMD Linear Array Architecture

The SIMD architecture [13,14] is a combination of N processors arranged in a linear array fashion with re-configurable interconnections where each processor is capable of passing data without delay. The architecture can be reconfigured, depending on which octave coefficients are being computed. The reconfigurable switches organize the N processors as an array of size $N/2^{m-1}$ for computation of the mth octave coefficients where $1 \leq m \leq J$. The interconnection scheme for the computation of 3 octaves on a 8 processor array is presented in Figure 2.4.

In this architecture, computation of all the outputs of any particular octave is executed at the same time. The computation consists of multiplying the incoming data values with the filter coefficients. Partial results are subsequently updated and the input data is passed to the active right neighbor. In order to compute J octaves using this

architecture, the time required is $L*J$ time units and the pipeline delay is also $L*J$ time units. The overall complexity of this architecture is of order $O(Nk)$ where $0 \leq k \leq J-1$.

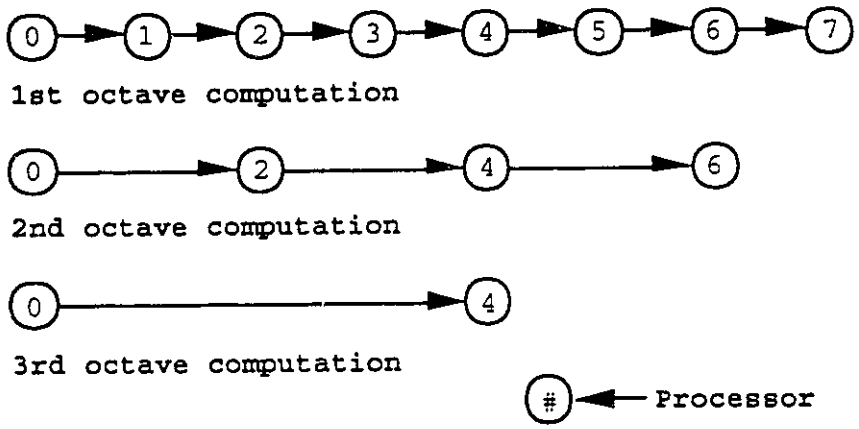


Figure 2.4. Interconnection among processors in the SIMD linear array.

2.4.3 SIMD multigrid architecture

An efficient way of computing 1-D DWT is mapping it onto a multigrid architecture [13,14]. There are ($\log N + 1$) levels for a multigrid architecture of size N, with $N/2^i$ processors in level i, where $0 \leq i \leq \log N$. Figure 2.5 presents the block diagram for $N = 8$. The function of the architecture is as follows: The ith level of the multigrid computes the $(i + 1)$th octave outputs, and sends the low-pass outputs to the level $(i + 1)$. Details of the computation of the ith octave outputs are similar to those presented for the SIMD linear array. The execution time for this architecture is $L*J$, and the pipeline delay is L. The complexity of the architecture is of order $O(N\log Nk)$. The pipeline delay of this architecture can be decreased to the order of $O(1)$ by increasing the number of processors at each level by a factor L.

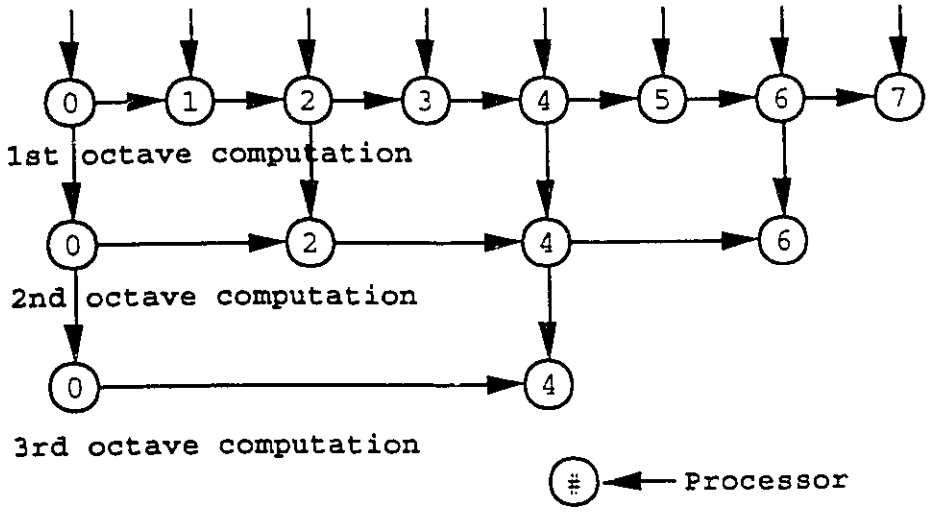


Figure 2.5. Interconnection among processors in the Multigrid architecture, N=8.

2.5 Complex DWT architectures

2.5.1 Folded Wavelet architecture

The folded wavelet architecture [15], computes all stages of the DWT decomposition, on a single processor. The process of folding (also known as multiplexing) is equivalent to scheduling the computations of different parts of the Pyramid Algorithm on one complex processor. The higher octave computations are thus performed in between the first octave computations. The scheduling of all calculations for any number of stages can be guaranteed within N samples, due to the complexity O(N) -1 of the DWT computation.

The DWT folded wavelet architecture is based on a schedule of computations as illustrated in equations 2.4. The schedule is based on the relative frequency of occurrence of 1st octave, 2nd octave and 3rd octave computations, and hence, are scheduled two, four and eight time units apart respectively. The schedule of computations is periodic with a

period equal to 8 for a three level decomposition. Its design is based on the Pyramid Decomposition Algorithm and achieves 7/8th hardware utilization. The Folded Wavelet Architecture uses an output converter to synchronize intermediate results and make them available for the computations of subsequent higher octave coefficients. It minimizes the number of registers by a systematic employment of "lifetime analysis" [16,17] in which a table of activity for intermediate results is constructed. The record is a list of all time units at which specific variables are used in one frame of computation. The minimum number of registers needed for implementation is then compiled. The computation of Wavelet coefficients is periodic, and hence, all registers are reserved for a particular intermediate result from one period to the next. The systolic VLSI architecture for calculating DWT proposed in this thesis, is based on the folded architecture. Details of the architecture are presented in Chapter 3.

2.5.2 Digit Serial Wavelet architecture

The digit serial architecture [15] has a higher hardware utilization and reduced routing and interconnection overhead compared to the folded wavelet architecture. It processes more than one but less than all bits of a sample at one time. Moreover, it has a structure similar to that of the Pyramid Algorithm. For a three octave wavelet decomposition, where the four output signals generate 4, 2, 1 and 1 output samples, the processor uses 1/2-word, 1/4-word and 1/8th word digit sizes respectively, and thus achieves complete hardware utilization.

The difficulty in the digit serial wavelet architecture is in achieving conversion from the one word-parallel bit serial, to two half-word parallel bit serial data format. The data rate [16] (number of units of data per input/output clock cycle) at the input and output must be identical and hence designing a digit serial architecture is a complex task. For a three octave wavelet decomposition, the *i*th octave wavelet implementation is obtained using a

word size $W/2^i$ where W is the length of the word and $i = 1, 2$ or 3. The problem is solved by rewriting the filter computation equation as follows:

$$g_0u(2k) + g_1u(2k - 1) + g_2u(2k - 2) + g_3u(2k - 3) =$$
$$= [g_0u(2k) + g_2u(2k - 2)] + [g_1u(2k - 1) + g_3u(2k - 3)] \qquad (2.5)$$

Here, two half words are generated in parallel in one cycle, and the constant data rate at the input and output is maintained. Signal $u(n)$ is processed to generate $u(2k)$ and $u(2k - 1)$. $u(2k - 2)$ is obtained from $u(2k)$ with delay of one word, whereas $u(2k - 3)$ is obtained from $u(2k - 1)$ also with one word delay. The computation of an output coefficient consists of computing two half output components, one even and one odd with half of the filter coefficients in each component. The half output coefficients are then added together to generate one output coefficient. This process is shown in Figure 2.6.

As shown in Figure 2.6, the basic building blocks of this architecture are serial to parallel converters and blocks of filters. The converters can be designed using activity period and register allocation techniques similar to those of the folded wavelet architecture. The digit serial multiplier and adder circuits are presented in [16,17,18].

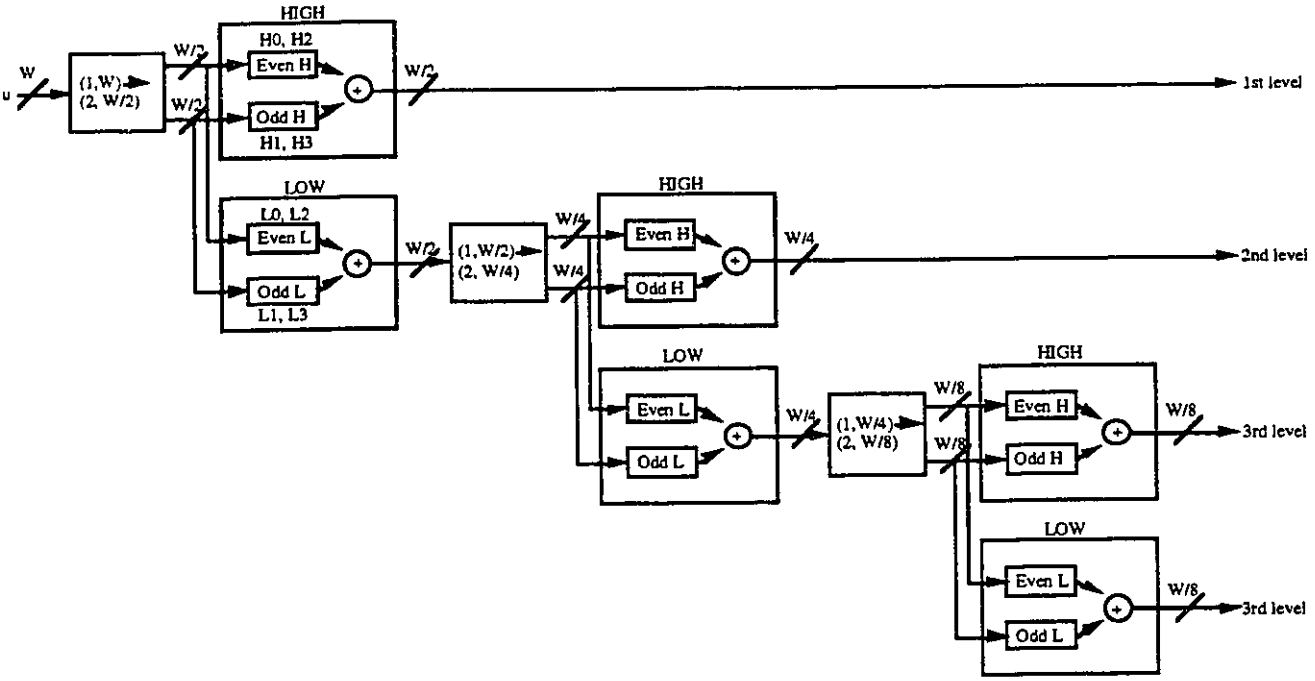


Figure 2.6. Three level analysis wavelet decomposition for the digit serial wavelet architecture.

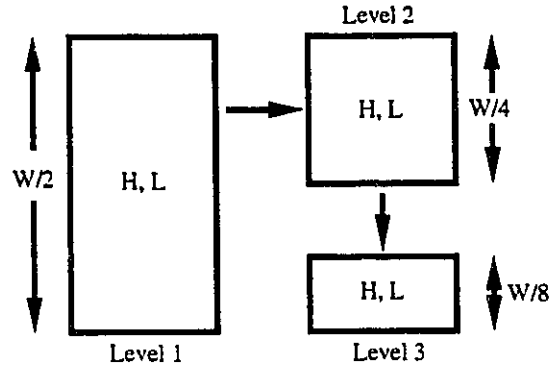


Figure 2.7. The layout floor-plan of the digit serial architecture.

The digit serial architecture solves the problem of decreasing utilization across the

octaves, found in the basic architectures of Section 2.4, by using digit-slice arithmetic of different length for each stage. The first stage employs full precision arithmetic hardware (e.g.: 16 bits), while the following stages employ only half the arithmetic precision of the preceding stage. Full precision operations are performed by doubling the number of passes through the hardware compared to the preceding stages. The computational throughput is thus matched to the incoming data rate. The silicon area of each stage, is thus approximately half of the preceding stage. The negative aspect of the digit serial architecture is that the word size must be a power of two, making this architecture unusable in applications requiring 6-tap filters.

2.6 2-D Block based DWT architecture

The architectures presented in section 2.4 and 2.5, are for 1-D DWT. We note that a 2-D DWT architecture can be implemented in a straight-forward manner by inserting a transposer and memory between two 1-D DWT modules. The transposer and the memory together realign the data stream between the first 1-D and the second 1-D computations and store the intermediate results.

A block based transform [19], is a non-optimal decomposition scheme, which has the potential for reducing memory requirements and eliminating the need for a transposer. However, it introduces blocking artifacts which in turn degrades the picture quality. In order to alleviate this problem, block treatment schemes such as zero padding, edge pixel repetition, edge pixel linear extrapolation, and overlapping edge pixels with neighboring blocks have been introduced [19]. The edge pixel repetition gives the optimal computational performance with minimum visual distortion.

The 2-D DWT block based architecture presented in [19] consists of four octaves of decomposition: first octave of Daubechies-4 wavelet decomposition followed by three octaves of Haar wavelet decomposition as shown in Figure 2.8. The two types of

decomposition are executed by the "D-4 PROCESSOR" and "HAAR PROCESSOR" respectively. The two processors can perform both forward as well as reverse transforms. The Daubechies-4 transform is computed using the following matrix multiplication operation on a 16 X 16 image block:

$$Y = W * D * W^T \tag{2.6}$$

where D is the image block (20 X 20) extended on each side by two columns (two rows), W is the matrix of interleaved low-pass and high-pass coefficients and Y is the output of the transform.

The operation in Equation 2.6 is implemented using a multiplier-and accumulator (MAC) block which multiplies a 4 X 4 block data with prestored filter coefficients. The MAC units generate in parallel the 2-D LL, LH, HL and HH coefficients. The LH, HL and HH coefficients are output from the chip whereas the LL coefficients are saved for further processing by the Haar processor. Figure 2.9 presents the block level diagram of that process.

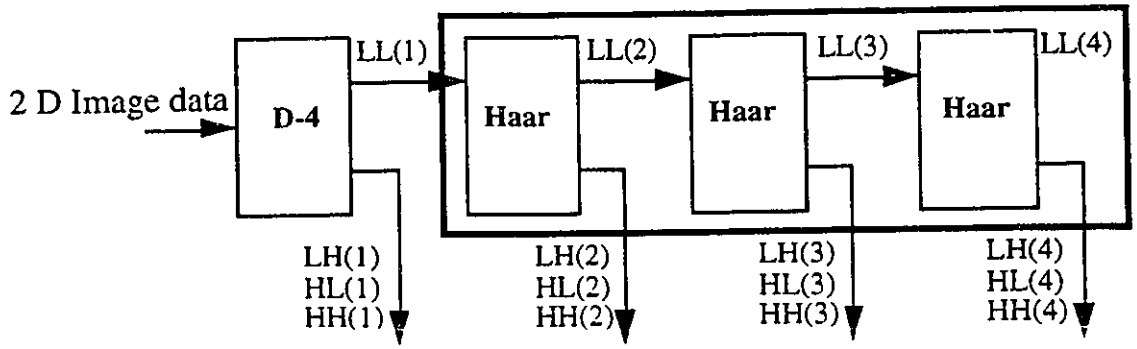


Figure 2.8. Block based architecture's forward transform.

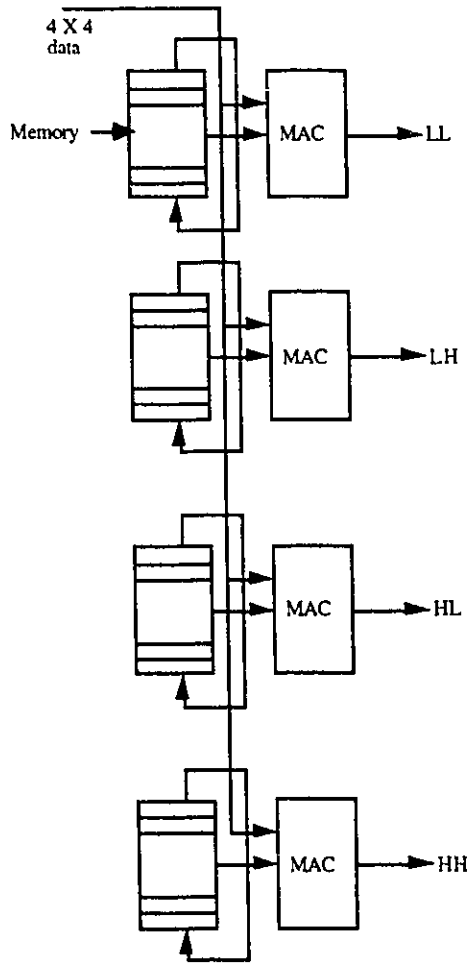


Figure 2.7. D-4 Processor.

The LL coefficients output from the D-4 processor are fed to the Haar processor which executes three consecutive row by row Haar transforms. The forward 2D Haar calculation is implemented using a 13-bit adder to add or subtract four LL(1) coefficients. Addition and/or subtraction operation for two stages is shown in Figure 2.10.

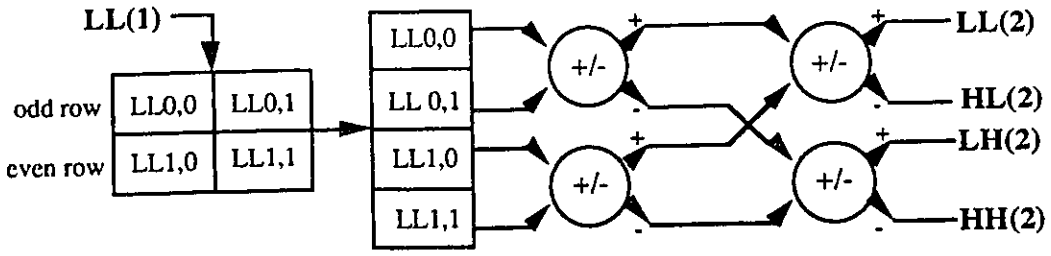


Figure 2.10. Foward 2D Haar transform.

In the first stage, the non overlapping pairs of data are added or subtracted and the results for the first row are stored in a shift register. Subsequently, the results for the first row are used in the second stage of processing to generate four subband coefficients. As in the first decomposition, LL(2) coefficients are saved for the third octave decomposition. In addition to the 13-bit adder, the Haar processor employs 84 (words) X 13 (bits) SRAM memory to store LL(1), LL(2) and LL(3) coefficients. Furthermore, it has three shift registers of eight, four and two words which are used to store intermediate results in the second, third and fourth octave computations respectively.

2.7 AWARE's WTP architecture

The Wavelet Transform Processor [20] (WTP) designed and manufactured by AWARE Inc., has been until recently the only simulated and tested DWT architecture. The WTP is capable of computing forward and inverse wavelet transforms for 1-D input data. It can execute the DWT using a maximum of six filter coefficients. In addition, it can be cascaded to execute transforms using longer support wavelets. The WTP has been clocked at speeds of 30 MHz and offers 16 bit precision on input and output data. The DWT computation is executed in a synchronous pipeline fashion and is under complete user control. The device is available in a 100 pin plastic pack (PQFP).

The core of the WTP chip, as shown in Figure 2.11 comprises of the wavelet transform pipeline, a cross-port switch, a group of registers and three data buses: BUSA, BUSB and DO. The transform pipeline which constitutes the computational engine of the WTP consists of four stages which in turn comprises of a 16-bit by 16-bit integer multiplier, an adder, a shifter, and data registers. In addition, each stage has a corresponding 16-bit filter coefficient register.

The input to and the output from the pipeline, is provided by the three 16-bit data busses, BUSA, BUSB and DO. The first two are bi-directional buses, which can be used as either input or output data buses, but one only in each direction at a time. DO is a unidirectional output data bus. To set the directions of the BUSA and BUSB data buses, a 16-bit, bi-directional switch XD is used.

The WTP processor has three operating modes, compute, load, and idle. The normal mode of operation is the compute mode, which the processor enters only after the load mode had been executed. In the load mode, the on-chip registers are loaded and/or updated. If not in either compute or load modes, the processor is set to idle mode.

The WTP enters the compute mode, when valid data is present on either one of the two bi-directional BUSA or BUSB buses. Control signal DIV and a clock CLK strobe the data into the wavelet pipeline. The DOV signal is used to validate the output signals after the input data has passed through the pipeline. The pipeline continues to operate until the last output has been validated by DOV at which point the processor returns to the idle mode.

In the load mode, the register group is loaded/updated under external control. Clocking of the pipeline is disabled, and BUSB is used to access the data registers. A control signal COA is used to enter or exit the load mode. When the pipeline is empty and the chip register group is not being accessed, the WTP is the idle mode.

At the clock speed of 33 MHz, the time necessary to compute one coefficient is 33

ns. The latency of the WTP processor is 9 clock cycles, i.e. the first coefficient is available 9 clock cycles after the first data sample enters the processor pipeline.

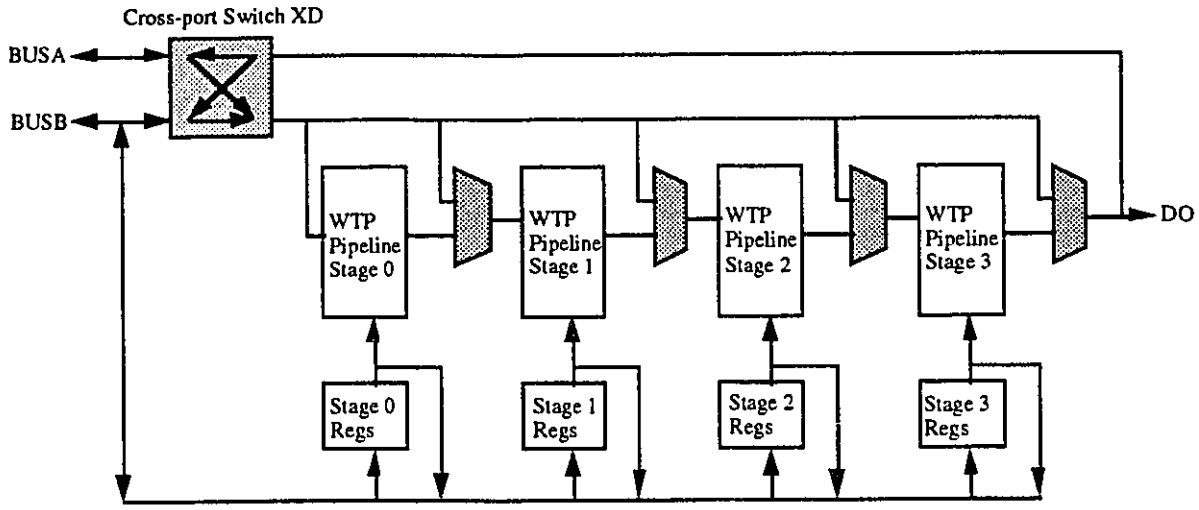


Figure 2.11. Block diagram of the Aware Wavelet processor.

2.8 Comparison and discussion

The three basic architectures presented in Section 2.4 vary in terms of the silicon area required for their implementation as well as the pipeline delay and the initial time period necessary for the computation of the first coefficient. Table 2.1 presents the results of a comparison in terms of area and pipeline delay for the three basic DWT architectures.

None of the three basic architectures have been implemented in silicon and hence concepts such as speed and power consumption can only be estimated. In addition, issues of scheduling intermediate data and control mechanism in general have not been clearly addressed. The three basic architectures have not shown any success in extracting parallelism present in DWT and have not been considered for the computation of 2-D DWT. In summary, the three basic DWT architectures are abstract and high level solutions to the problem of computing 1-D DWT.

Architecture	Area	Pipeline Period
Parallel filter	$O(Lk\log N)$	N
Dedicated SIMD linear array	$O(Nk)$	$L\log N$
Dedicated SIMD multigrid	$O(N\log Nk)$	L
AWARE's WTP	$O(Nk)$	$O(N\log N)$

Table 2.1. Comparison of the area and pipeline period for the 1-D DWT basic architectures.

The block based DWT architecture solves some key problems associated with full scale 2-D DWT. The chip uses four MAC (multiply and accumulate) units to execute forward and reverse transforms. The architecture is scalable and thus meets the throughput requirements of various applications. It requires only a minimal on-chip memory and implements 2-D wavelet transform directly and without data transposition. However, the advantage of small memory, can be a drawback. The block based architecture may introduce block boundary effects degrading the visual quality. In summary, the architecture implemented at 60 MHz, achieves real time DWT computation for full size, gray scale motion video at 30 frames per second.

Section 2.7 represents a summary of the literature on the Aware Inc. WTP chipset. The WTP architecture suffers from several problems. For example, the issue of control signaling, which in the supporting documentation has been characterized as flexible and completely defined by the user. Detailed analysis of the WTP documentation leads to the conclusion that "under complete user control" means that the user and not the chip is responsible for controlling the intermediate data inside the pipeline. As pointed out in Chapter 1, data dependencies between various octave computations necessitate that the

intermediate results be properly stored in temporary registers. The WTP is unable to control the internal sequence of events, and therefore, resembles more a filter bank controlled from outside. The WPT does not function in a turnkey mode and the user has to be sure that the output obtained is the expected output.

The most sophisticated DWT architectures (folded and digit serial) as well as the DWT-SA proposed in this thesis have potential for an improved performance. The three architectures presented in sections 2.5 and the DWT-SA architecture achieve high hardware utilization and are modular, cascadable and require minimal control by the user. In addition, they have the ability to function in a turnkey mode i.e. user supplies the data and the filter coefficients, whereas the chip computes the results.

The comparison of the two most sophisticated DWT architectures, (the folded and the digit serial) is summarized in Table 2.2.

Property	Folded	Digit-serial
No. of Multipliers	16	14
No. of Registers	164	258
Latency	28	70
Power Consumption	$16CV^2f$	$9.76CV^2f$
Interconnection	Complex	Simple
Silicon Area	Higher	Less

Table 2.2. Performance Comparison of the Folded and Digit-serial Architectures.

The number of word-level registers employed in both architectures was calculated using lifetime analysis. The difference is that in the digit serial architecture a digit serial processor together with a word converter [16] was used. The digit-serial architecture has a lower power consumption due to reduced to $0.6V$ and $0.4V$ respectively supply voltage in

the second and third level wavelet decompositions. V signifies the voltage level of the first level decomposition. The reduction in voltage results from the half size and the quarter size digits used in the second and third octave computations. Consequently, the number of adders in the second and third level computations is one half and one quarter the number of those found in the first level computations. The advantages of the folded architecture are lower latency, arbitrary size wordlength and pipeline with a finer grain. However, these benefits require more complex routing, larger chip area and higher power consumption. The digit serial architecture is superior in applications where latency is not critical. It requires small chip area and lower power consumption.

2.9 Conclusion

In this chapter, we presented a brief introduction to Discrete Wavelet Transform and a review of the existing architectures for DWT. The existing architectures vary from abstract implementations of the DWT algorithm, to complex architectures whose potential for real-time implementation has been demonstrated by system level simulations. In addition, we introduced the details of the only available commercially VLSI chipset: the WTP processor by AWARE Inc. The results of a cross-comparison between the architectures has also been presented.

Chapter 3

Systolic Architecture for DWT

This chapter presents the proposed architecture for computing 1-D DWT. The architecture is systolic in nature and performs both high-pass and low-pass calculations with only one filter, in contrast to the approaches presented in the literature [23,24,25]. The architecture is simple, modular, cascadable, and easily extendible for any block size DWT computation, and hence is well suited for VLSI implementation. We refer to the proposed architecture as DWT-Systolic Array (DWT-SA).

In Sections 3.1 and 3.2, we present the design of the DWT-SA. Here, we describe the components of the architecture and present two possible schemes for register allocation. Section 3.3 contains a detailed block description of the entire DWT-SA. Design of the high speed Booth multiplier employed in DWT-SA is presented in Section 3.4.

3.1 Design of the DWT-SA

The design of DWT-SA, is based on a computation schedule derived from Equations 3.2a - 3.2n, which are the result of applying the Pyramid Algorithn for N=8 (Fig. 2.2), to the six stage

filter equations 3.1a - 3.1b. Equations 3.1a and 3.1b represent the high-pass and low-pass components of the six stage FIR filter. Both sets of equations correspond to Equations 2.4a - 2.4n and 2.3a - 2.3b respectively, and are repeated below for ease of presentation.

$$High(z) = g_0 + g_1 z^{-1} + g_2 z^{-2} + g_3 z^{-3} + g_4 z^{-4} + g_5 z^{-5} \quad (3.1a)$$

$$Low(z) = h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} + h_4 z^{-4} + h_5 z^{-5} \quad (3.1b)$$

1st octave:

$$b(0) = g_0 a(0) + g_1 a(-1) + g_2 a(-2) + g_3 a(-3) + g_4 a(-4) + g_5 a(-5) \quad (3.2a)$$

$$b(2) = g_0 a(2) + g_1 a(1) + g_2 a(0) + g_3 a(-1) + g_4 a(-2) + g_5 a(-3) \quad (3.2b)$$

$$b(4) = g_0 a(4) + g_1 a(3) + g_2 a(2) + g_3 a(1) + g_4 a(0) + g_5 a(-1) \quad (3.2c)$$

$$b(6) = g_0 a(6) + g_1 a(5) + g_2 a(4) + g_3 a(3) + g_4 a(2) + g_5 a(1) \quad (3.2d)$$

$$c(0) = h_0 a(0) + h_1 a(-1) + h_2 a(-2) + h_3 a(-3) + h_4 a(-4) + h_5 a(-5) \quad (3.2e)$$

$$c(2) = h_0 a(2) + h_1 a(1) + h_2 a(0) + h_3 a(-1) + h_4 a(-2) + h_5 a(-3) \quad (3.2f)$$

$$c(4) = h_0 a(4) + h_1 a(3) + h_2 a(2) + h_3 a(1) + h_4 a(0) + h_5 a(-1) \quad (3.2g)$$

$$c(6) = h_0 a(6) + h_1 a(5) + h_2 a(4) + h_3 a(3) + h_4 a(2) + h_5 a(1) \quad (3.2h)$$

2nd octave:

$$d(0) = g_0 c(0) + g_1 c(-2) + g_2 c(-4) + g_3 c(-6) + g_4 c(-8) + g_5 c(-10) \quad (3.2i)$$

$$d(4) = g_0 c(4) + g_1 c(2) + g_2 c(0) + g_3 c(-2) + g_4 c(-4) + g_5 c(-6) \quad (3.2j)$$

$$e(0) = h_0 c(0) + h_1 c(-2) + h_2 c(-4) + h_3 c(-6) + h_4 c(-8) + h_5 c(-10) \quad (3.2k)$$

$$e(4) = h_0 c(4) + h_1 c(2) + h_2 c(0) + h_3 c(-2) + h_4 c(-4) + h_5 c(-6) \quad (3.2l)$$

3rd octave:

$$f(0) = g_0 e(0) + g_1 e(-4) + g_2 e(-8) + g_3 e(-12) + g_4 e(-16) + g_5 e(-20) \quad (3.2m)$$

$$g(0) = h_0 e(0) + h_1 e(-4) + h_2 e(-8) + h_3 e(-12) + h_4 e(-16) + h_5 e(-20) \quad (3.2n)$$

As shown in Equations 3.2, there are eight first octave coefficient computations, four second octave computations and two third octave computations.

Equations 3.1a and 3.1b show that the high-pass and the low-pass filter coefficients can be

computed using a six stage multiply and accumulate digital filter, where partial results are computed at the each stage separately and then progressively passed to the next higher stage for accumulation.

In addition, Equations 3.1a and 3.1b suggest that due to filter coefficient calculations spanning over 5 clock cycles, a temporary storage is required for the incoming data samples.

Closer analysis of Equations 3.2a -3.2n, indicates that there exist data dependencies between adjacent octave computations. The second octave coefficient calculations require the results from the first octave computations, whereas the third octave calculations depend on the results from the second octave computations. As a result, intermediate registers are required in order to store different octave outputs before they are used in the subsequent octave computations.

In Sections 3.1.1 and 3.1.2, we present the three components introduced above. We start with the description of the Filter Unit (FU) and its subcomponent, the Filter Cell (FC) We then continue with the presentation of the storage facilities: the Input Delay unit (ID) and the intermediate Register Bank (RB). In Section 3.2, we describe the design of the rest of the architecture and particularly of the control mechanism. Architectural details of the high-speed Booth multiplier employed in the Filter Cell are presented in Section 3.3.

3.1.1 Filter Unit

The Filter Unit (FU) proposed for this architecture is a six-stage non-recursive FIR digital filter whose transfer function for the high-pass and low-pass components are shown in Equations 3.1 where g_0 to g_5 and h_0 to h_5 are the filter's high-pass and low-pass coefficients, respectively.

Computation of any coefficient can be executed by employing a multiply and accumulate method where partial products are computed separately and subsequently added. This feature makes possible systolic implementation of DWT. The latency of each filter stage is 1 and since partial components of more than one DWT coefficient are being computed at any given time, the

latency of the filter once the pipeline has been filled is also 1. The systolic architecture of a six-stage filter is shown in Figure 3.1. Here, partial results (one per cell) are computed and subsequently passed in a systolic manner from one cell to the adjacent cell.

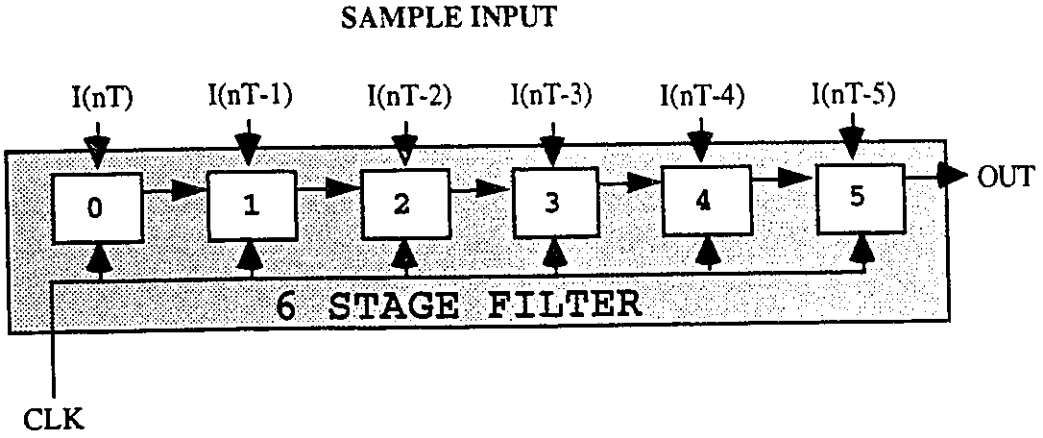


Figure 3.1. Systolic operation of the six stage filte.

Filter Cell

Equations 3.1 show that except for different values of filter coefficients high pass and low pass computations at specific time instants are identical. By introducing additional control circuitry, both computations can be performed by the same hardware. The difficulty however, is that each multiplication must be executed in half the clock cycle. The proposed filter cell consists therefore of only one multiplier, one adder and two registers to store the high-pass and the low-pass coefficients as shown in Figure 3.2.

The high-pass coefficient calculation is performed during the first half of the clock cycle and the low-pass calculation is performed during the second half.

A high speed Booth multiplier is used in the filter cell. Its detailed description is presented in Section 3.4.

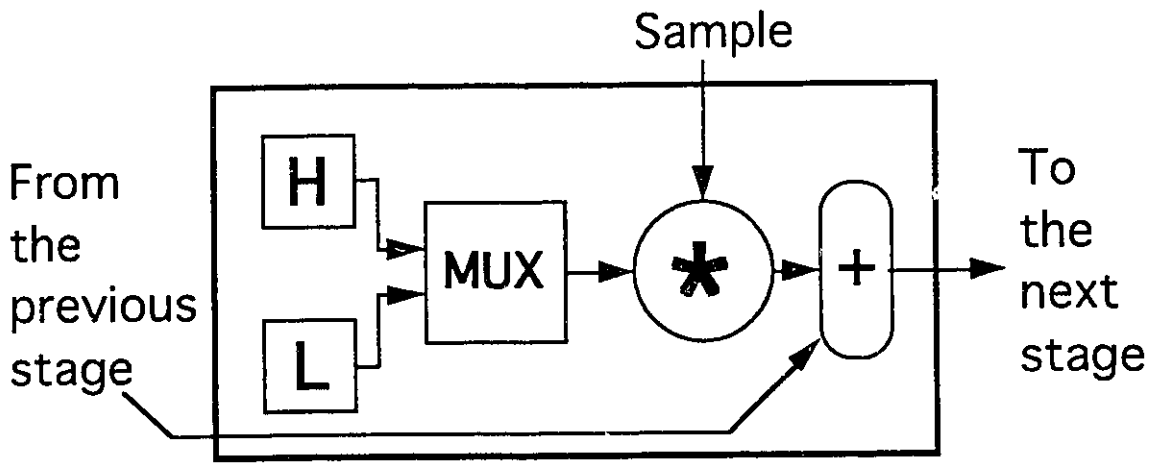


Figure 3.2. Proposed filter cell.

3.1.2 Storage

The Input Delay Unit (ID)

Equations 3.1a and 3.1b show that the value of computed filter coefficient depends on the present as well as the five previous data samples (the negative time indexes in Equations 3.1 correspond to the reference starting time unit 0). It is therefore required that the present and the past five input data values be held in registers and be retrievable by the FU and the CU. Figure 3.3 shows the block diagram of the ID unit.

As shown in figure 3.3, five registers are connected serially. At any clock cycle each register passes its contents to its right neighbor which results in only five past values being retained.

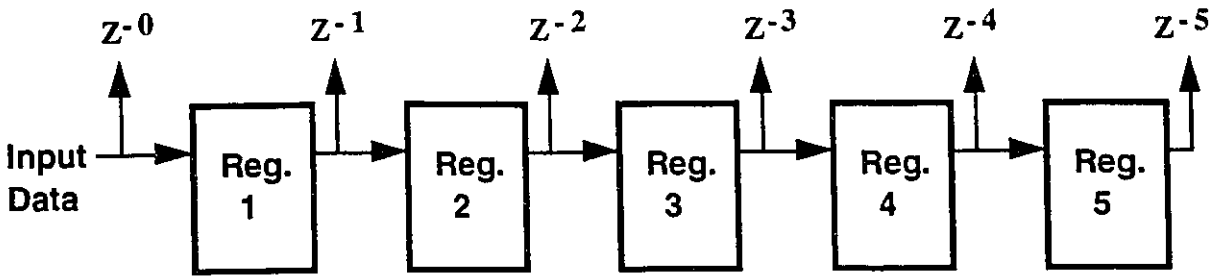


Figure 3.3. Input Delay unit (ID) architecture.

The Register Bank Unit (RB)

Several registers are required for storage of the intermediate partial results. Analysis of Equations 3.1a and 3.1b justifies the requirement for the register bank, however it does not explain its size. In the next section, it will be shown that the number of registers required is 26. The registers are connected serially and are controlled by an overall clock. The output of one register is directly connected to the input of the adjacent register, and thus the register bank is implemented as a shift register shown in Figure 3.4.

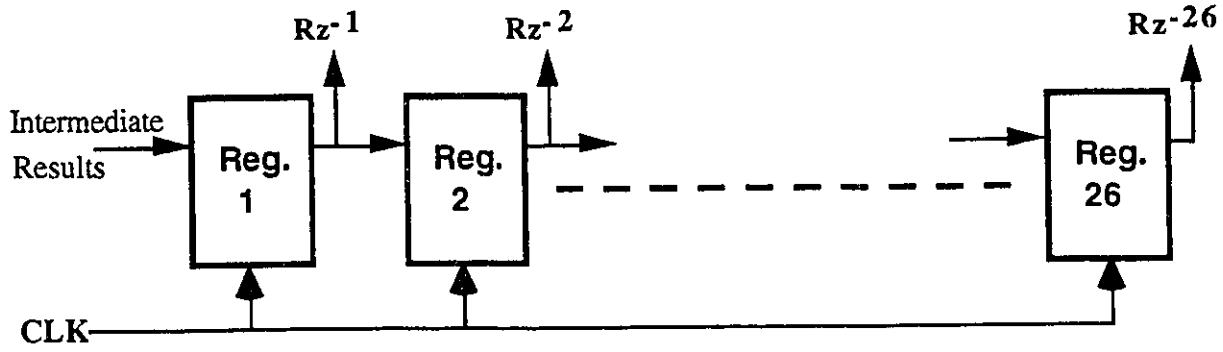


Figure 3.4. Register Bank (RB) block diagram.

3.2 DWT-SA Control (CU)

One of the most important aspects of the DWT-SA architecture is its potential for real-time

operation.

The proposed DWT-SA architecture computes N coefficients in N clock cycles and achieves real-time operation by executing computations of higher octave coefficients in between the first octave coefficient computations. The first octave computations are scheduled every $N/4$ clock cycles, while the second and third octaves are scheduled every $N/2$ and every N clock cycles respectively. The CU design incorporates architectural innovations that achieve the real-time operation.

There are several approaches for scheduling the octave computations. In the DWT-SA architecture, we propose a schedule based on the filter latency of 1. The computations are scheduled at the earliest possible clock cycle, and computed output samples are available one clock cycle after they have been scheduled as shown in Table 3.1. The delay is minimized through the pipeline facilitating real-time operation. For example, the computation of d(0) can only be executed after the calculation of c(0) has been completed. The calculation of c(0) scheduled for cycle 1 is completed in cycle 2 due to the filter latency 1. d(0) is therefore scheduled for computations in a later cycle, cycle 4.

Init Cycle	High - pass	Low - pass
1	b(0)	c(0)
2	-	-
3	b(2)	c(2)
4	d(0)	e(0)
5	b(4)	c(4)
6	f(0)	g(0)
7	b(6)	c(6)
8	d(4)	e(4)

Table 3.1. Schedule for one complete set of computations.

The schedule presented in Table 3.1 is periodic with period N, and the hardware is not utilized in cycles 2 or kN+2 where k = 1,2,3... The computation schedule in Table 3.1 leads to a high hardware utilization of 7/8 or 87.5%.

3.2.1 Register Allocation

The next step in designing the DWT-SA architecture is the design of the Control Unit (CU), and the Register Bank (RB). The two components synchronize the availability of operands. There are two schemes which can be employed for this purpose, namely Forward Register Allocation (FRA), or the Forward-Backward Register Allocation (FBRA). The FRA method uses a set of registers which are allocated to intermediate data on the first come first serve basis. It does not reassign any registers to other operands once its contents have been accessed. The FBRA scheme is similar, except that once the operand stored has been used, the register is reallocated to another operand. The FRA method is simpler, requires less control circuitry and permits easy adaptation of the architecture for coefficient calculation of more than 3 octaves. It results however, in less efficient register utilization.

In either scheme the coefficient computations are periodic and therefore, each register containing a specific variable will be reserved for the same variable in the next period. The construction of the register allocation tables for both FRA and FBRA are presented below.

FRA Register Allocation

To demonstrate the construction of the register allocation table using the FRA approach, we will consider the case of computing the coefficients c(0) and c(2).

As shown in Table 3.1. coefficient c(0) is computed in cycle one, whereas the coefficient c(2) is scheduled for computation in cycle 3. These two computed coefficients along with other four; c(4), c(6), c(8) and c(10) are needed for the computation of coefficient e(0). According to Table 3.1, e(0) computation is scheduled for cycle 4. However, the six coefficients c(0), c(2),

c(4), c(6), c(8), c(10) will not be available until cycle 12, and therefore the calculation of e(0) has to be scheduled for cycle 4 + N, or 12. Similarly, coefficient e(4) is computed not in cycle 8, but 8 + N, thus cycle 16. The number of registers needed becomes apparent once one complete frame of computations has been scheduled.

Systematic application of the described method, yields a complete schedule of computation for all intermediate and final coefficients as shown in Table 3.2. Due to its size, the table has been divided into two parts.

In the FRA register allocation approach where data moves systolically in one direction only, it is possible to increase the number of DWT decomposition octaves by placing additional registers in series after the register R26. The new registers hold the intermediate coefficients needed for the computation of the next decomposition octave. Hardware utilization of the higher octave decomposition registers is inversely proportional to the order of computed coefficients.

FBRA Register Allocation

Table 3.2 shows that not all registers in the FRA register allocation scheme are used at every time instant. In fact, close examination of Table 3.2, suggests that for the first to second octave calculations, registers R1 to R11 are used 87.5 % of the time, whereas for the second to third octave calculation, registers R12 to R26 are used 25 % of the time.

A higher register utilization can be achieved by applying the reallocation scheme where empty registers are reallocated to other variables once the original variables held in them are no longer valid. This approach, which in ideal case would make use of every register at every time instance, has a negative impact on the complexity of the architecture. It requires additional complex control circuitry for data reallocation, and results in a less modular architecture compared to the FRA approach. Moreover, the FBRA approach eliminates one of the most important aspects of the DWT-SA architecture, its modularity. Table 3.3 shows a complete register allocation table for the DWT-SA using the FBRA approach.

Cycle	Com	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13
1	**c(0)**													
2		c(0)												
3	**c(2)**		c(0)											
4	e(0)	c(2)		c(0)										
5	**c(4)**		c(2)		c(0)									
6	g(0)	c(4)		c(2)		c(0)								
7	**c(6)**		c(4)		c(2)		c(0)							
8	e(4)	c(6)		c(4)		c(2)		c(0)						
9	c(0)		c(6)		c(4)		c(2)		c(0)					
10		c(0)		c(6)		c(4)		c(2)		c(0)				
11	c(2)		c(0)		c(6)		c(4)		c(2)		c(0)			
12	**e(0)**	**c(2)**		**c(0)**		**c(6)**		**c(4)**		**c(2)**		**c(0)**		
13	c(4)	e(0)	c(2)		c(0)		c(6)		c(4)		c(2)			
14	g(0)	c(4)	e(0)	c(2)		c(0)		c(6)		c(4)		c(2)		
15	c(6)		c(4)	e(0)	c(2)		c(0)		c(6)		c(4)			
16	**e(4)**	**c(6)**		**c(4)**	e(0)	**c(2)**		**c(0)**		**c(6)**		**c(4)**		
17	c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)		c(6)			
18		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)		c(6)		
19	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)			
20	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)		
21	c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)			
22	g(0)	c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		
23	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)		
24	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	
25	c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	c(6)			e(0)
26		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		
27	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)		
28	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	
29	c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)			e(4)
30	g(0)	c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		
31	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)		
32	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	
33	c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	c(6)			e(0)
34		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		
35	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)		
36	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	
37	c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)			e(4)
38	**g0)**	c(4)	**e(0)**	c(2)		c(0)	**e(4)**	c(6)		c(4)	**e(0)**	c(2)		

Table 3.2. FRA Register allocation.

Cycle	Com	R14	R15	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26
1	**c(0)**													
2														
3	**c(2)**													
4	e(0)													
5	**c(4)**													
6	g(0)													
7	**c(6)**													
8	e(4)													
9	c(0)													
10														
11	c(2)													
12	**e(0)**													
13	c(4)													
14	g(0)													
15	c(6)													
16	**e(4)**													
17	c(0)													
18														
19	c(2)													
20	e(0)													
21	c(4)													
22	g(0)													
23	c(6)													
24	e(4)													
25	c(0)													
26		e(0)												
27	c(2)		e(0)											
28	e(0)			e(0)										
29	c(4)				e(0)									
30	g(0)	e(4)				e(0)								
31	c(6)		e(4)				e(0)							
32	e(4)			e(4)				e(0)						
33	c(0)				e(4)				e(0)					
34		e(0)				e(4)				e(0)				
35	c(2)		e(0)				e(4)				e(0)			
36	e(0)			e(0)				e(4)				e(0)		
37	c(4)				e(0)				e(4)				e(0)	
38	**g(0)**	**e(4)**				**e(0)**				**e(4)**				**e(0)**

Table 3.2. FRA Register allocation (continued).

	Co	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14
1	**c(0**														
2		c(0)													
3	**c(2**		c(0)												
4	e(0)	c(2)		c(0)											
5	**c(4**		c(2)		c(0)										
6	g(0)	c(4)		c(2)		c(0)									
7	**c(6**		c(4)		c(2)		c(0)								
8	e(4)	c(6)		c(4)		c(2)		c(0)							
9	c(0)		c(6)		c(4)		c(2)		c(0)						
10		c(0)		c(6)		c(4)		c(2)		c(0)					
11	c(2)		c(0)		c(6)		c(4)		c(2)		c(0)				
12	**e(0**	**c(2**		**c(0**		**c(6**		**c(4**		**c(2**		**c(0**			
13	c(4)	e(0)	c(2)		c(0)		c(6)		c(4)		c(2)				
14	g(0)	c(4)	e(0)	c(2)		c(0)		c(6)		c(4)		c(2)			
15	c(6)		c(4)	e(0)	c(2)		c(0)		c(6)		c(4)				
16	**e(4**	**c(6**		**c(4**	e(0)	**c(2**		**c(0**		**c(6**		**c(4**			
17	c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)		c(6)				
18		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)		c(6)			
19	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)				
20	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)			
21	c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)				
22	g(0)	c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)			
23	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)			
24	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)		
25	c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	c(6)			e(0)	
26		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	c(6)			e(0)
27	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)	e(0)	c(0)	e(4)			
28	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)	e(0)	c(0)	e(4)		
29	c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)	e(0)		e(4)	
30	g(0)	c(4)	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)	e(0)		e(4)
31	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	c(6)	e(4)	c(4)	e(0)		e(0)	
32	e(4)	c(6)		c(4)	e(0)	c(2)		c(0)	e(4)	c(6)	e(4)	c(4)	e(0)		e(0)
33	c(0)	e(4)	c(6)		c(4)	e(0)	c(2)	e(0)	c(0)	e(4)	c(6)	e(4)		e(0)	
34		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)	e(0)	c(0)	e(4)	c(6)	e(4)	e(0)	e(0)
35	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)	e(0)	c(0)	e(4)	e(0)	e(4)	e(0)
36	e(0)	c(2)		c(0)	e(4)	c(6)		c(4)	e(0)	c(2)	e(0)	c(0)	e(4)	e(0)	e(4)
37	c(4)	e(0)	c(2)		c(0)	e(4)	c(6)	e(4)	c(4)	e(0)	c(2)	e(0)	e(0)	e(4)	
38	**g(0**	c(4)	**e(0**	c(2)		c(0)	**e(4**	c(6)	e(4)	c(4)	**e(0**	c(2)	e(0)	e(0)	e(4)

Table 3.3. FBRA register allocation.

3.2.2 Activity Periods

Tables 3.2 and 3.3, shows that the last pair of coefficients i.e. f(0) and g(0) for the first group of eight samples is available at time instance 38. This means that coefficient computations are overlapped for 5 periods ($38/5 \leq 5$). Table 3.2 also shows the time periods when the intermediate coefficients must remain valid in order to produce the subsequently higher octaves of coefficients. For example consider the case for a first octave coefficient c(0). It remains active from the time instant it has been computed to the time of calculation of the next higher octave of coefficients. In this configuration, it means from time instance 1 to time instance 12. Similarly, e(0), a second octave coefficient remains active until the third octave coefficients (in which it is used) are computed, i.e. from time instance 12 to time instance 38. All the intermediate results, and the associated periods of activity are listed in Table 3.4.

Sample	Available at cycle	Life period
c(0)	1	1 to 12
c(2)	3	3 to 14
c(4)	5	5 to 16
c(6)	7	7 to 18
e(0)	12	12 to 18
e(4)	16	16 to 38

Table 3.4. Activity periods for intermediate results.

The number of registers required in this architecture is directly proportional to the number of levels of DWT decomposition, and is calculated during the construction of the timetable of computations. For the DWT-SA architecture which computes three octaves of DWT decomposition and employs the FRA register allocation method the top row of Table 3.2 indicates the number of registers is 26.

We note that since no variable in Table 3.2 has a negative time index, a periodic interpretation of that table is required. Consider for example, the variable c(-2) in the computation of d(0) and e(0) in cycle 4. The periodic interpretation of Table 3.2 implies that the register which holds the variable c(-2) in cycle 4, also holds the variable c(-2+8) in clock cycle 4+8. Table 3.2 shows that c(6) is held in register R5.

3.2.3 The Control Unit (CU)

The Control Unit of the DWT-SA architecture is shown in Figure 3.5. It schedules the computation of each DWT coefficient as shown in Table 3.1.

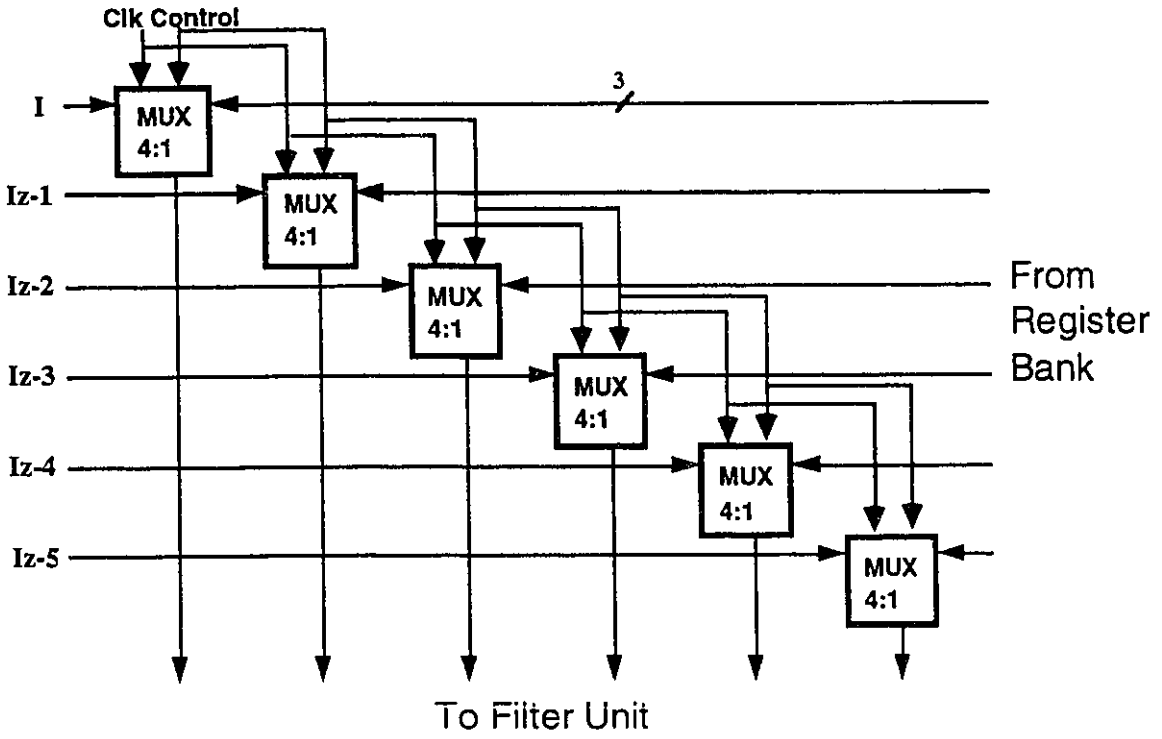


Figure 3.5. The Control Unit (CU) block diagram.

CU is a switch, that passes data from the Input Delay (ID), or the Register Bank (RB) to the Filter Unit (FU). CU is a modular switch with a number of subcomponents equal to the number of taps in the FU. The CU multiplexes data from the ID every second cycle, and from the RB in cycles 4,6, and 8. In cycle 2, CU remains idle, i.e. it does not allow any passage of data. The block diagram of the CU is presented in Figure 3.5.

Proper timing, synchronization as well as enabling and disabling of the CU is ensured by the global CLK signal.

3.3 DWT-SA Architecture

The DWT-SA architecture employs the FRA register allocation scheme. From now on, we will refer only to the FRA register allocation.

The proposed DWT-SA architecture is shown in Figure 3.6. It is obtained by systematic analysis of the FRA register allocation table (Table 3.2) in conjunction with the filter equations (Equations 3.1 and 3.2).

The number of switching inputs in each control subcell is equal to the number of octave computations. The first octave computations are scheduled every second clock cycle and therefore that corresponding switch input is labeled $2k$, where k is any non-negative integer. Moreover, its inputs are supplied directly by the ID. Second octave computations are executed in clock cycles 4 and 8, which is reflected by the label $4k+3$. The third octave computations are scheduled in clock cycle 6, or $8k+5$. Both second and third octave computations use partial results from previous octave computations and therefore use inputs from RB. Table 3.2 determines which register is used as output.

The DWT-SA architecture comprises of the four basic blocks: the Input Delay unit (ID), the Filter Unit (FU), the Register Bank (RB) and the Control Unit (CU).

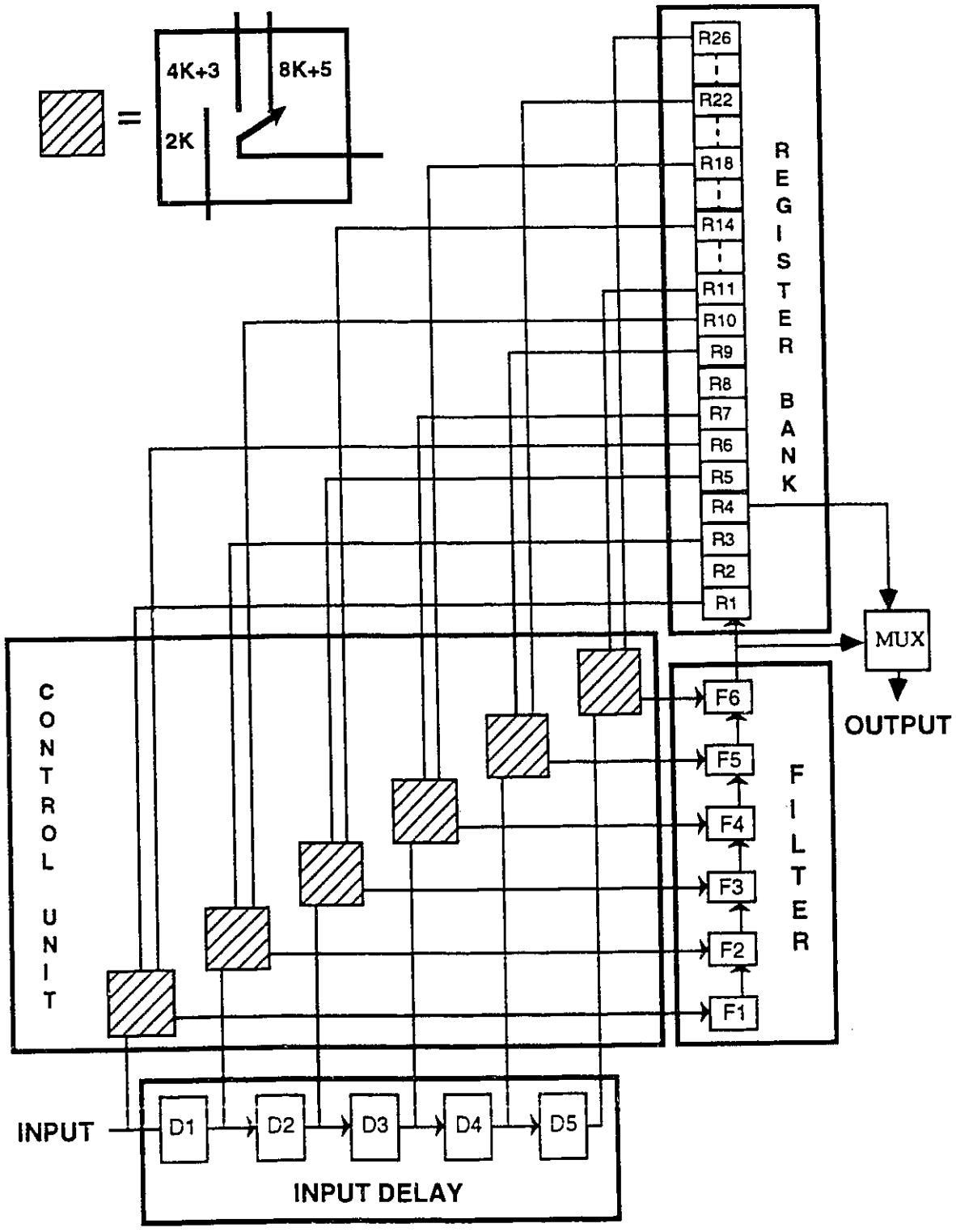


Figure 3.6. DWT-SA architecture.

Delay of the DWT-SA architecture consists of the latency period necessary to fill up the filter for the first time, plus the delay through the registers as described in Table 3.1. First results are thus produced 5 + 38 clock cycles after the first input sample has entered the pipeline. Subsequent coefficients come out of the pipeline every 8 clock cycles. The output coefficients are output from the final filter stage.

3.4 High Speed Multiplier

In order to meet the real-time requirements of application such as video compression a fast multiplier design is required. Several multiplier architectures suitable for VLSI implementation have been proposed in the literature [23]. They are the usual "paper and pencil" algorithm of an array multiplier consisting of *m* shifts and *m-1* additions in a multiplication of two numbers, where *m* is the number of bits in the shorter number. This approach is relatively simple to implement and results in a regular structure. The drawback however, is its large size and low speed. Other architectures, such as Wallace Tree and Dadda structures, reduce the number of addition stages, by employing special circuitry, and are hence very irregular.

The Booth algorithm [24] is a powerful algorithm for signed-number multiplication which has two main features: it treats both positive and negative numbers uniformly and is significantly faster than an array multiplier due to the reduced number of add stages required.

Booth's technique is based on the fact that a consecutive string of 1's can be rewritten as a difference of two numbers, as shown below:

$$2^{i+k} - 2^{i} = 2^{i+k-1} + 2^{i+k-2} + \ldots + 2^{i-1} + 2^{i} \tag{3.3}$$

The multiplier (M1) is scanned from right to left, one bit at a time and if the pattern changes from 1 to 0 or (0 to 1) a multiplicand (M2) is added or (subtracted). The same result can be obtained by examining pairs of bits in M1 in conjunction with the bit to the right of the bit pair

considered. For example, consider the following binary string, consisting of 7 binary digits, where the bold zeros are padded extremities:

$2^0 \quad 2^1 \quad 2^0 \quad 2^1 \quad 2^0 \quad 2^1 \quad 2^0$

0 0 0 1 1 1 1 0 **0**

Four pairs of numbers can be distinguished. The first group on the right, implies that -2^1 * M2 will be added, with the least significant bit in column 2^0. The second pair of numbers signifies that 0 * M2 will be added to the partial product, with the least significant bit in column 2^2. The next group, means that $+2^1$ * M2 will be added to the product, with the least significant bit in column 2^4. Finally, the last group of numbers will add 0 * M2 with the least significant bit in column 2^6. Each bit pair is examined concurrently with the higher-order bit of the next lower pair, and therefore, there are a total of eight possible versions of the multiplicand, as summarized in Table 3.5.

Multiplier bit		Bit on		
2^1	2^0	r. side	Multiplicand multiples	Explanation
i+1	i	i-1	to be added	
0	0	0	0 X	No string
0	0	1	+1 X	End of string
0	1	0	+1 X	Single 1
0	1	1	+2 X	End of string
1	0	0	-2 X	Beginning of string
1	0	1	-1 X	End/Beginning of string
1	1	0	-1 X	Beginning of string
1	1	1	0 X	String of 1s

Table 3.5. Multiplier bit pair recoding scheme.

The standard add-shift method requires 8 additions while the Booth algorithm multiplication requires only 4. By consistently examining the pairs of bits, the total number of additions is always *m/2* where *m* is the word length of the multiplier.

The block diagram of a Booth multiplier employed in the DWT-SA architecture is shown in Figure 3.7. Two computational subcells, namely: Full-Adder and/or Half-Adder are represented by squares whereas the two control subcells: Booth Recoder cell (BRC) and the Shift-and-Complement cell (SAC) are shown implicitly by the modified strings of numbers (M0-M3) and (Q0-Q3) flowing into the multiplication array as the Multiplicand and the Multiplier.

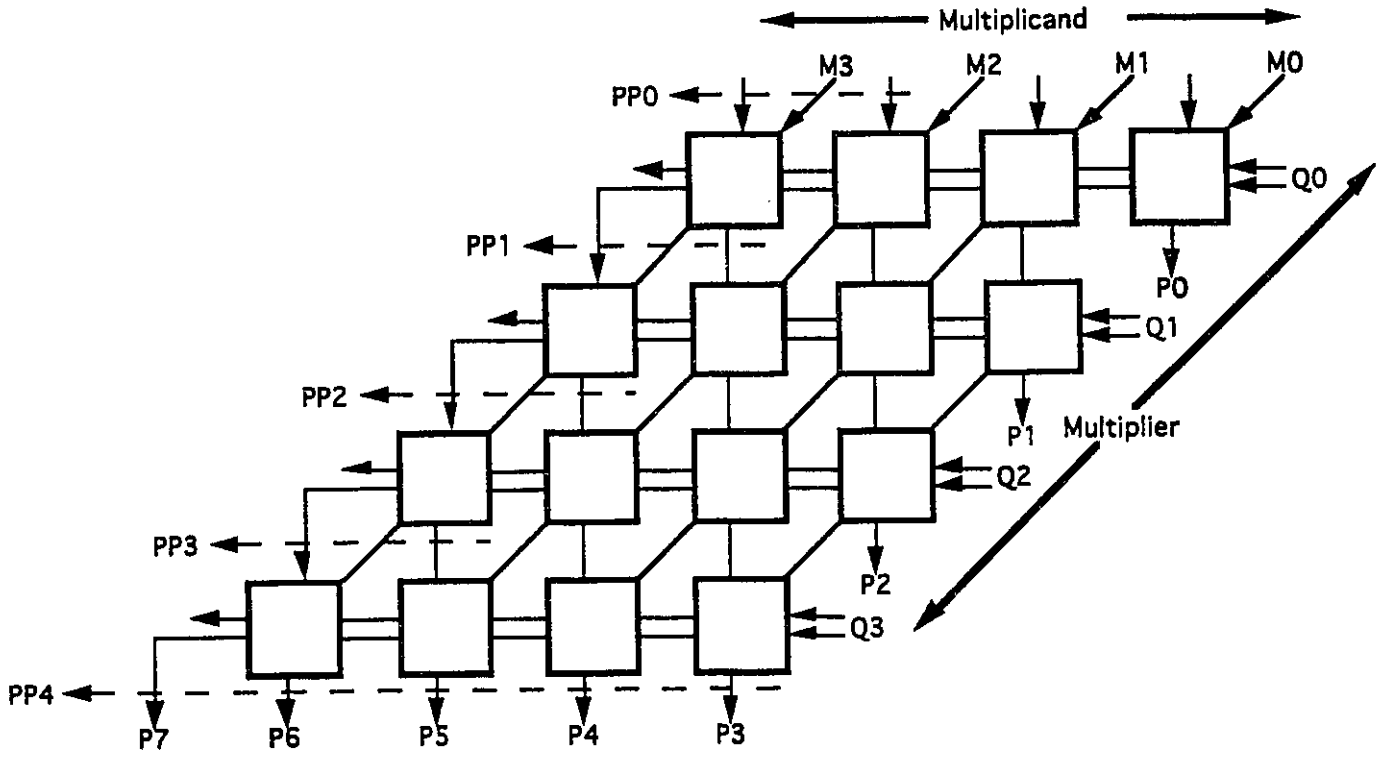


Figure 3.7. Booth Multiplier Architecture.

BRC and SAC subcells execute the algorithm of Table 3.5. The BRC converts three adjacent bits of the Multiplier input into a recoded Booth digit. The outputs of BRC are used by

SAC to control the inputs to the adder stages. SAC selects a bit from the multiplicand input word and feeds it to the adder. The bit in position *m* or *m-1* can be selected. Selection of bit *m-1* has the effect of multiplying the multiplicand by two. The result can also be complemented which converts the operation into a subtraction.

Schematics of all components of this multiplier are included in the Appendix A.

3.5 Conclusion

The proposed DWT-SA architecture uses only one filter in contrast to two parallel computational hardware employed in [15]. The architecture is optimized in hardware by using only a single multiplier and adder set in each filter cell to perform to generate all high-pass and low-pass coefficients. As demonstrated in Chapter 4, the proposed DWT-SA architecture can execute wavelet transform for monochrome video in real-time.

The DWT-SA architecture does not use any external or internal memory modules to store the intermediate results and therefore avoids the delays caused by access, read, write and refresh timing. In addition, there is no need for complex control circuitry to put the intermediate products in and out of the memory as a set of registers controlled by a global clock is employed. This results in a simple and efficient systolic implementation for the computation of 1-D DWT and hence is suitable for VLSI implementation.

The proposed architecture can be used to compute the DWT of any length sequence without dividing it into smaller subsequences. The only limitation of the DWT-SA architecture is the number of decomposition octaves, set at three, which can be easily increased by augmenting the number of temporary registers. We note that this approach does not necessitates any modification to the computation scheduling.

Chapter 4

VLSI Design of the DWT-SA

This chapter presents detailed circuit and gate level implementation of the DWT-SA. Section 4.1 shows the gate level design and simulation results of the FU. In Section 4.2 and 4.3 we describe the gate level design of the CU, and the RB. Simulation results of the entire architecture are shown in Section 4.4.

4.1 FIR Filter

The DWT-SA employs a sixth order non-recursive FIR filter whose architecture is presented in Chapter 3. The filter behaves in a systolic manner by computing partial products of more than one coefficient at a time. The first multiply and accumulate stage computes the first partial product and passes it to the second filter stage where it is added to the second partial product. That repetitive action continues until a complete DWT coefficient is output from the sixth filter stage. The delay through each filter stage is 1 clock cycle. Due to parallel computation of more than one DWT coefficient, the time necessary to complete the computation of the first coefficient is 5 clock cycles. Subsequent coefficients are output every clock cycle. The filter

schematic is presented in Figure 4.1.

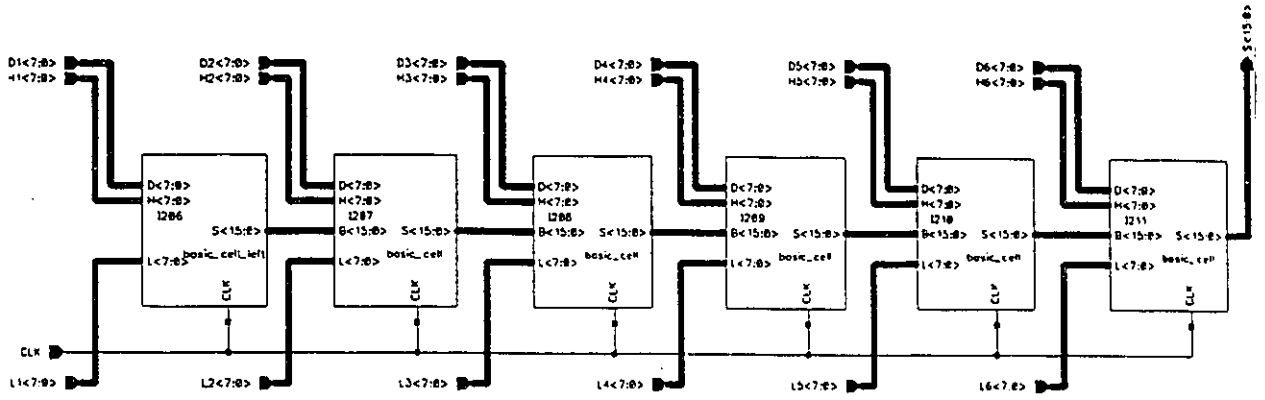


Figure 4.1: 6-stage FIR Filter.

As shown in Figure 4.1, the high-pass and low-pass filter coefficients are loaded into each filter cell separately and before any DWT coefficients are computed. The filter is controlled by a clock and responds to its high and low levels for the computation of the high-pass and the low-pass DWT coefficient respectively.

4.1.1 Filter Cell

Two different filter cells are employed in the design of the FIR filter. "Filter Cell Left" is used for the first stage, whereas "Filter Cell" shown in Figure 4.2, is used in the other five filter stages.

The need for two different filter cells results from the lack of the "carry in" component in the first filter stage. Consequently, the "Filter Cell Left" does not posses an adder present in the other five stages. Both filter cell types contain a multiplier and a multiplexer to select between the high-pass and low-pass coefficients. The filter coefficients and the input data are 8 bit wide and when multiplied, yield a 16 bit partial output of which 8 lowest bits are passed from one cell to the next cell.

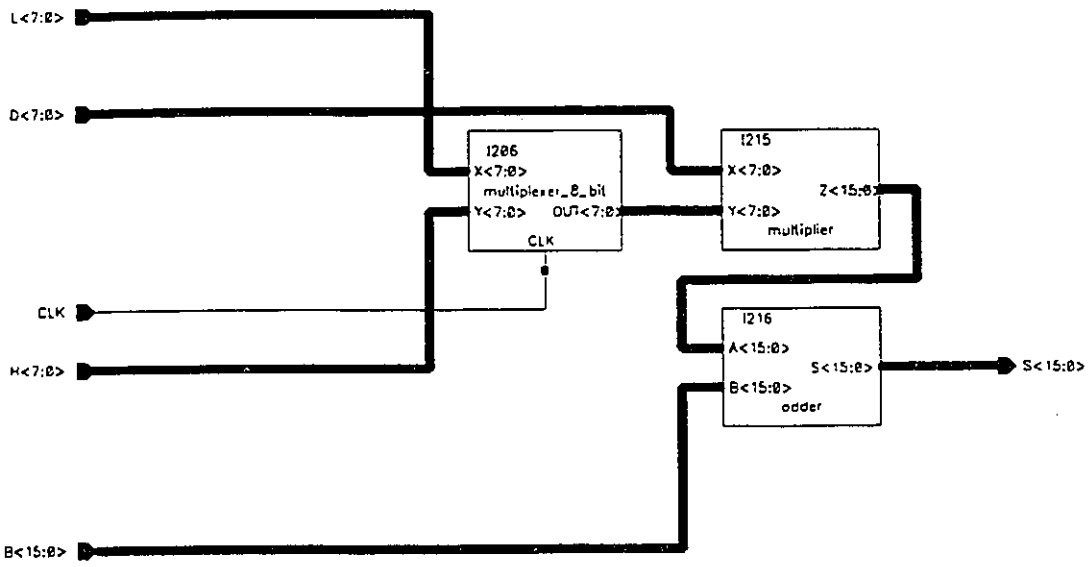


Figure 4.2: Filter Cell.

4.1.2 High Speed Booth Multiplier

The multiplier implemented in the DWT-SA architecture is based on the Booth Algorithm high-speed 8 X 8 integer multiplier whose general architecture is presented in Chapter 3. The multiplier comprises of four different subcells: the Booth Recoder Cell (BRC), the Shift and Complement Cell (SAC), the Full Adder (FA) and the Half Adder (HA) as shown in Figure 4.3. A total of 4 BRCs, 36 SACs, 36 FAs and 9HAs are used in the design. Note that the multiplicand is the X input in Figure 4.3, whereas the multiplier is the Y.

Booth Recoder Cell (BRC)

The Booth Recoder Cell transforms three adjacent bits of one of the inputs (the multiplier or the Y input) into a recoded Booth digit [1]. The recoded Booth digit is the control signal that operates on the second input (the multiplicand or the X input) as it passes through the four add

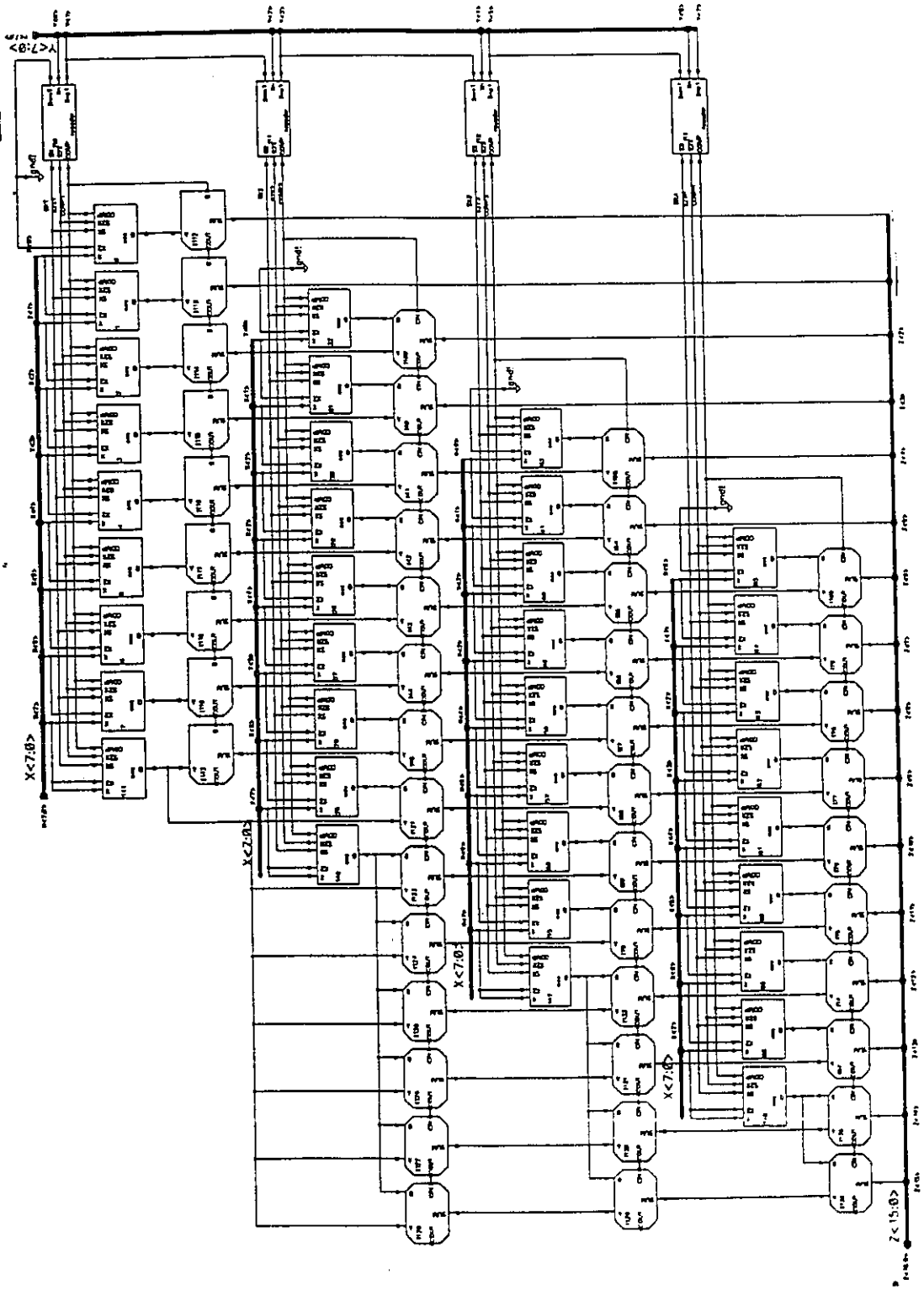


Figure 4.3: Booth Multiplier.

stages. The bit pattern of the Y input controls whether in the second, third or fourth add stages the X input is added to, added two times, subtracted from, or subtracted two times from, the previous add stage. The three output signals from the BRC subcell control the operations: COMP, SX and S2X, execute the control table shown in Figure 3.5. Gate level design of the BRC cell is presented in Figure 4.4. Transistors are minimum length: 1.2 um with the with indicated in the schematics. As an example, 10/5 indicates the P-channel and N-channel width to be 10 and 5 um respectively.

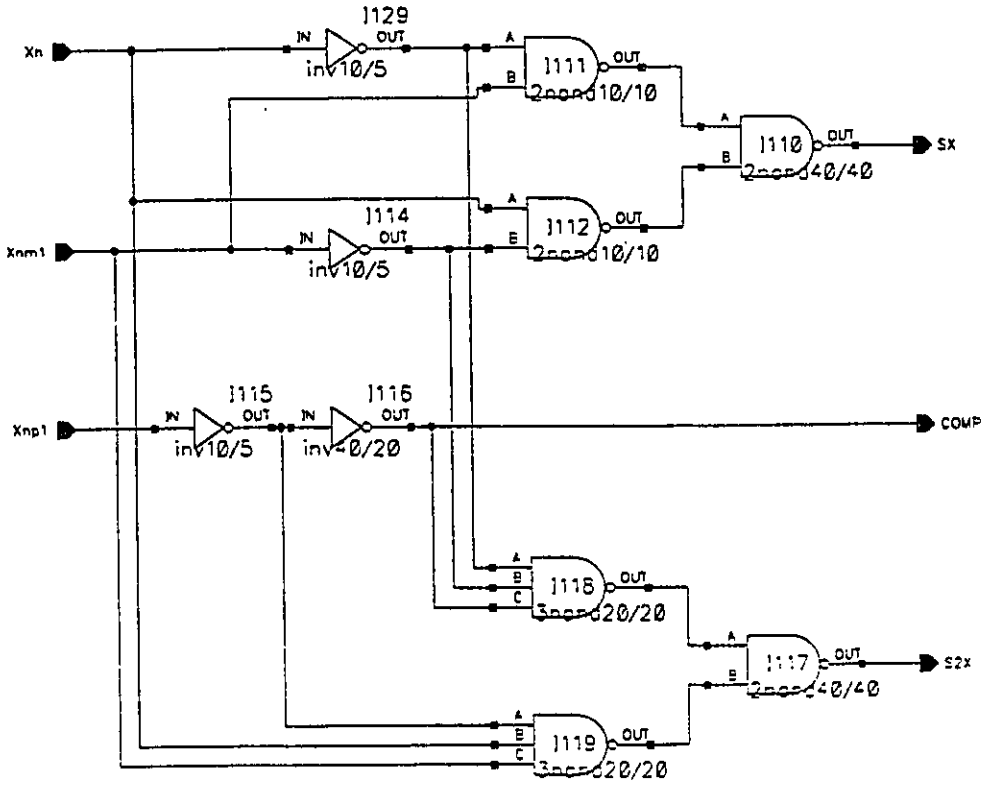


Figure 4.4. Booth Recoder Cell.

Shift and Complement Cell (SAC)

The SAC cell is the second control subcell used in the multiplier. It selects the input from

the multiplicand or the X input and passes it to the three adder stages of the multiplier. The selection process is completely based on the state of control signals from the BRC cell. If the SX signal is selected, all n bits of the multiplicand are passed to the adder stage. If the S2X signal is asserted, then n-1 bits are passed to the adder stages, which has the effect of multiplying the X input by two. Only one of the two: SX or S2X, can be selected at a time.

COMP signal can be activated anytime with either one of the above two control signals, thus changing an add operation to a subtraction. We note that if COMP signal is asserted, the carry bit at the output of the multiplier corresponding to the particular adder stage is set to 1. This has the effect of adding a 1 to the least-significant bit position when the partial product is complemented. The process of complementing and adding one to a number implements a subtraction when the number is represented in 2's complement form. Gate level design of the SAC cell is presented in Figure 4.5.

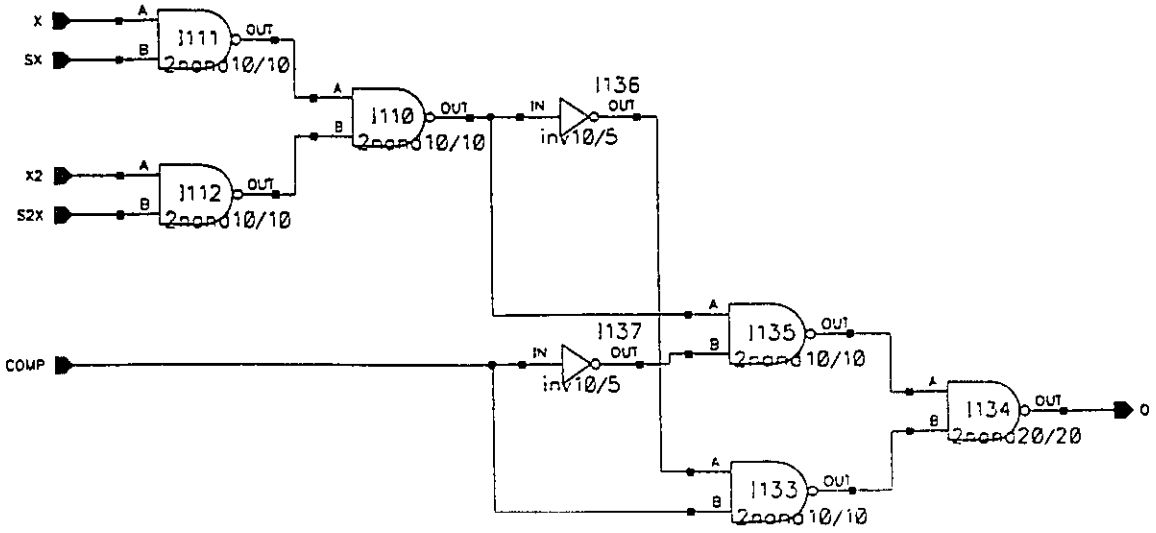


Figure 4.5: Shift and Complement Cell.

Full Adder Cell (FA) and Half Adder Cell (HA)

The Full Adder Cell and the Half Adder Cell are variations of those presented in [25]. Because both of these cells are part of the critical path, modifications were performed in order to decrease the carry out delay. Gate level schematics of the FA and the HA are presented in Figures 4.6. and 4.7.

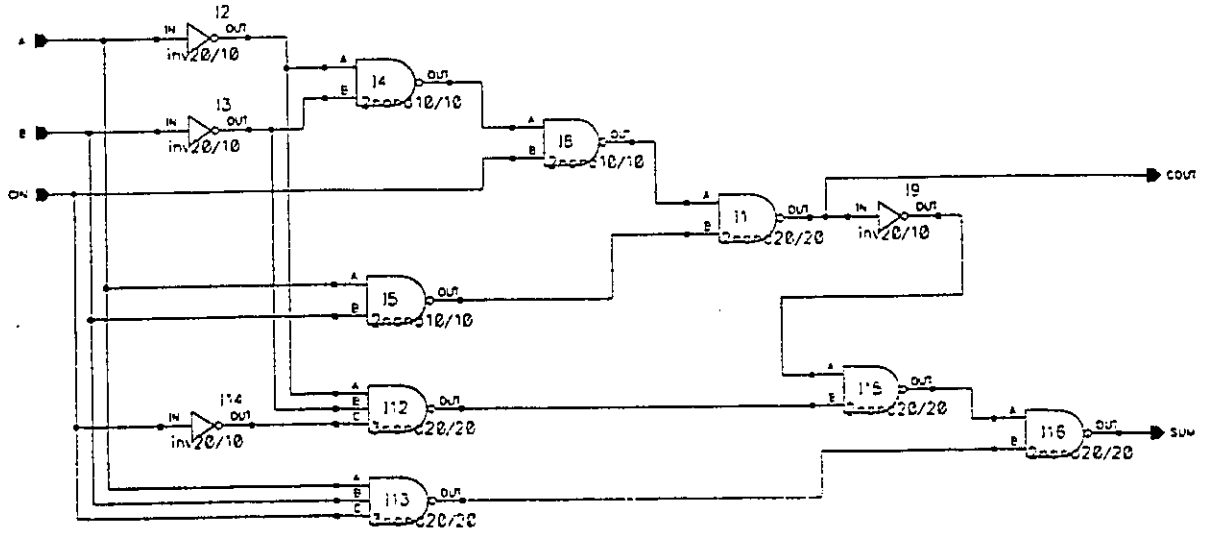


Figure 4.6: Full Adder.

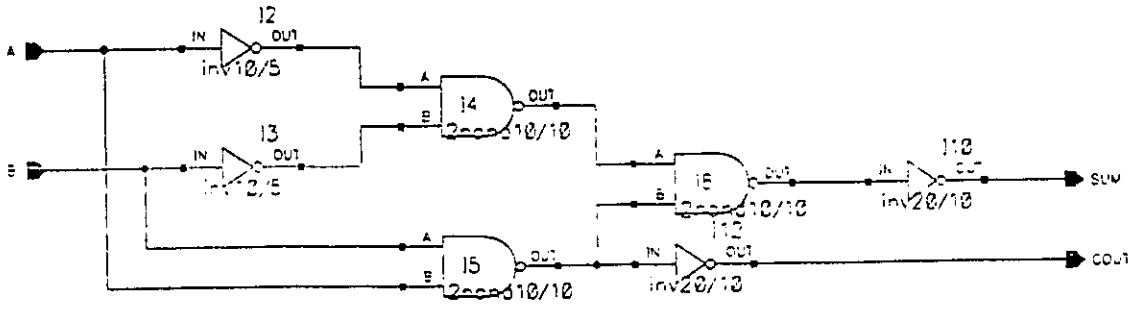


Figure 4.7: Half Adder

4.1.3 Filter Multiplexer

The multiplexer selects between the high-pass and low-pass filter coefficients and passes it to the multiplier. It is a standard 8-bit logic design controlled by the phase of the clock. Its 8-bit block configuration is shown in Figure 4.8, whereas the 1-bit gate design is presented in figure 4.9.

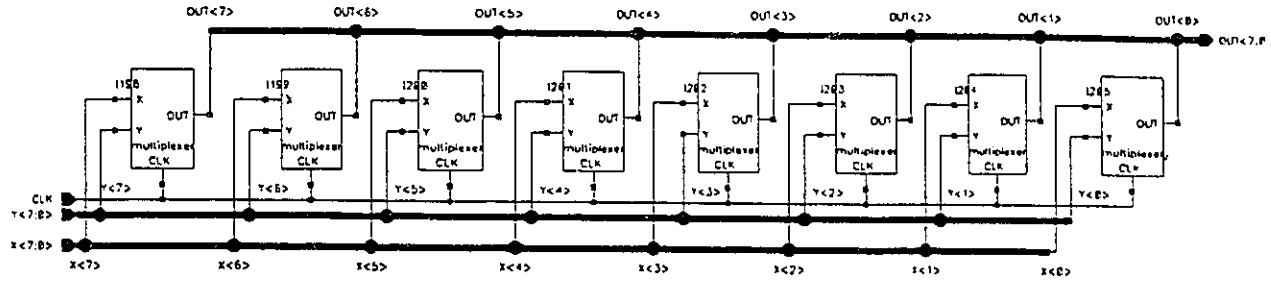


Figure 4.8: 8-bit Filter Multiplexer.

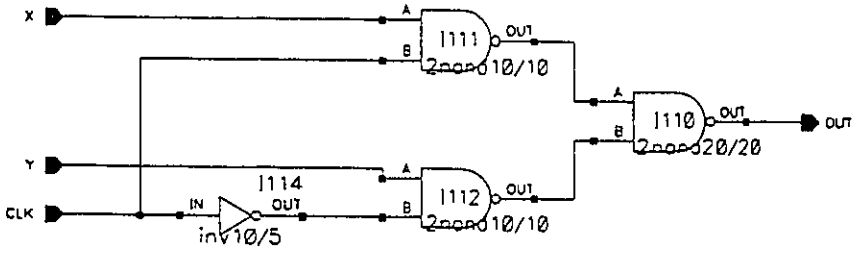


Figure 4.9: 1-bit Multiplexer.

4.2 Control Unit

The architectural details of the CU were presented in Section 3.2.3. The following presents the gate level implementation of CU in the DWT-SA architecture.

The top level schematic of CU is a combination of six word level switches that multiplex

data going to the FU, as shown in Figure 4.11. The switches are enabled and controlled by a Master Control (MC) circuit. The inputs labeled TOP come from the input pipeline and serve in the computation of the first octave DWT coefficients. Signals labeled MID(dle) and BOT(tom) are outputs from the RB and serve in the computation of the second and third octave DWT coefficients, respectively. Signal GR(ound) is a VSS connection employed during the k+2 clock cycles when the multiplier is in its idle mode.

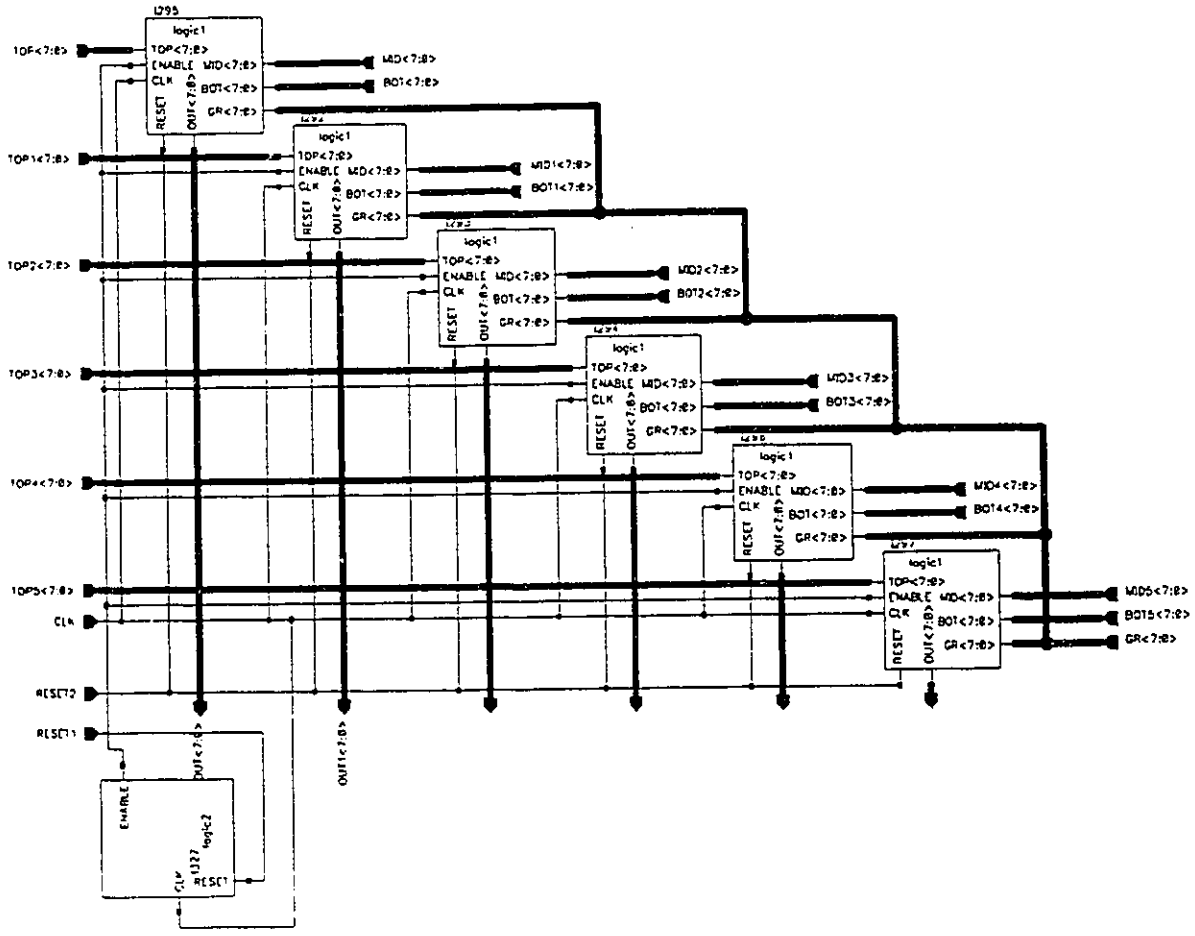


Figure 4.11: Control Unit.

4.2.1 Switch

The six word-level switches execute data multiplexing operation shown in Table 3.1. Each switch comprises a 3-bit counter, logic gates, and a demultiplexer as shown in Figure 4.12. Data to the FU is selected from different inputs depending on the clock cycle.

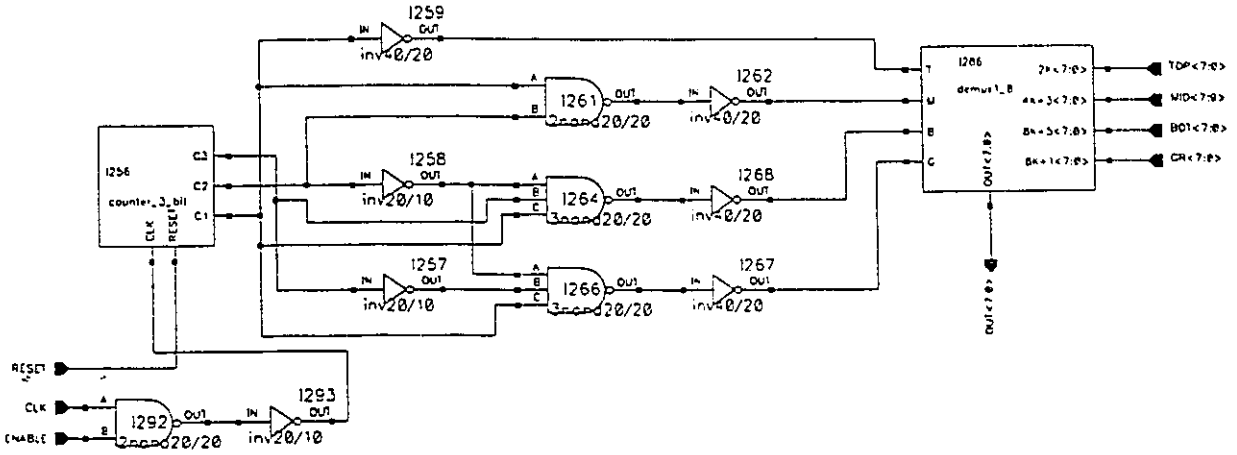


Figure 4.12: The Switch.

Demultiplexer

The demultiplexer takes the four different inputs, combines them with appropriate control signals: T(op), M(iddle), B(ottom) and G(round) generated by the counter and the control logic, and passes the output to the FU. The gate level circuit of the demultiplexer is shown in Figure 4.13.

3-Bit Counter

A standard, 3-bit counter based on 3 master slave (MS) T flip-flops is employed in the DWT-SA. The counter is enabled by the rising edge of the clock cycle and remains active (counts from 0 to 7) until it becomes disabled by the reset signal. Figure 4.14 shows the implementation of the 3-bit counter, whereas Figure 4.15 shows the Master-Slave flip-flop used in the counter.

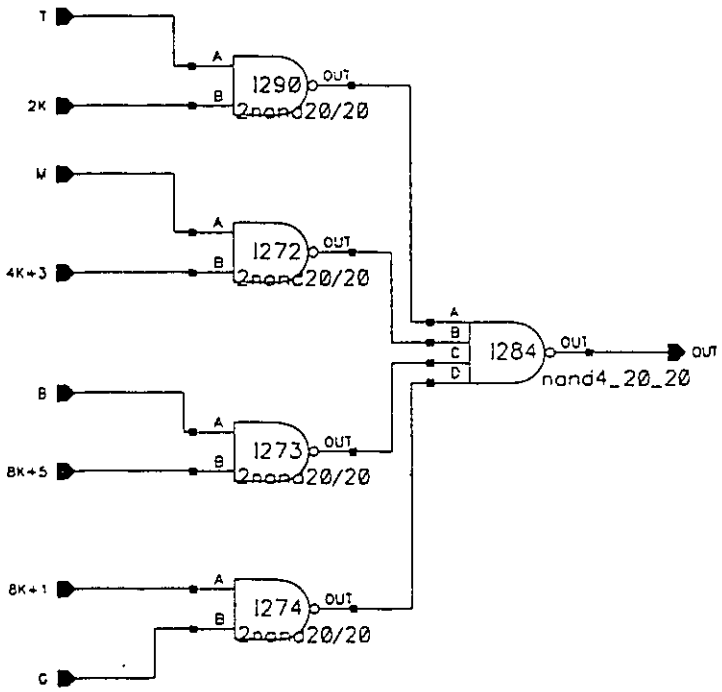


Figure 4.13: Bit-level demultiplexer.

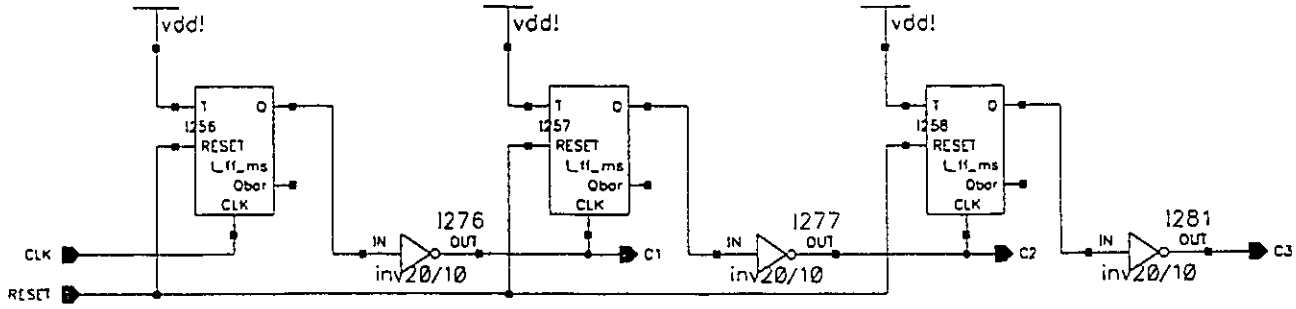


Figure 4.14: 3-Bit Counter.

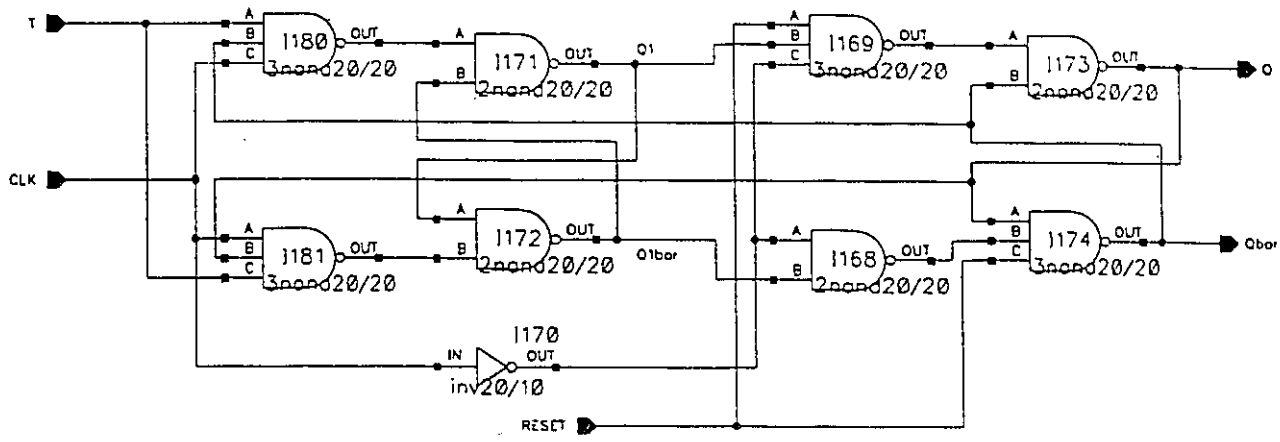


Figure 4.15: Master-Slave T Flip-Flop.

4.2.2 Master Control

The Master Control (MC) circuit ensures that the CU does not start operation until valid data samples are present at its inputs. It takes five clock cycles to fill up the input delay pipeline with data and thus, the MC enables the CU on the rising edge of the 6th clock cycle. It remains on until the entire chip is disabled. The internal structure of the MC consists of a three-bit counter and logic circuitry as shown in Figure 4.16.

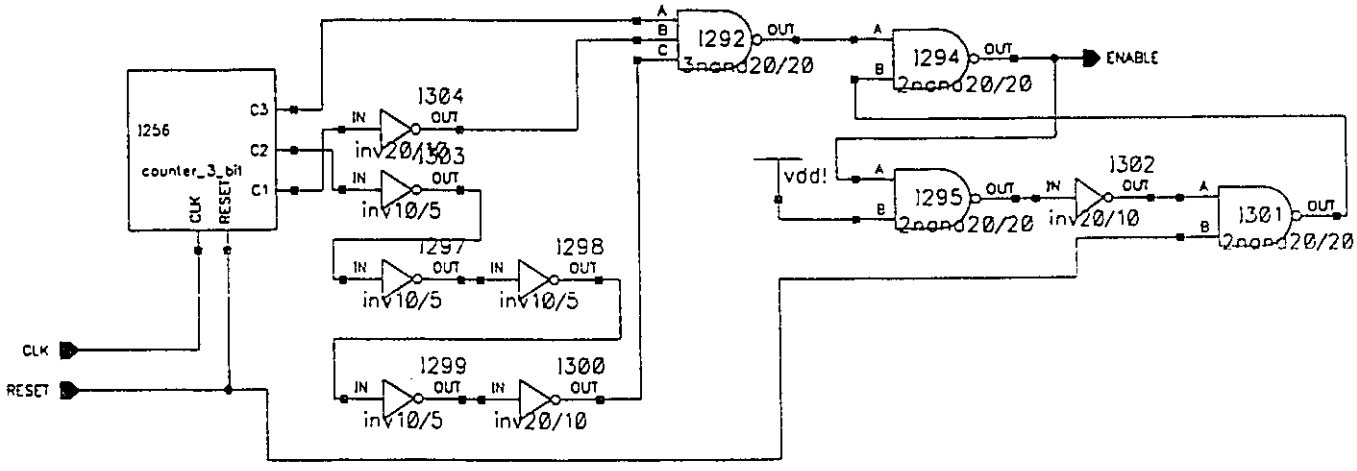


Figure 4.16: Master Control.

4.3 Storage

As presented in Chapter 3, two storage units are required in the DWT-SA architecture. First, is the ID unit, consisting of five 8-bit registers connected in series, whose outputs are the delayed input samples to the FU. The second storage unit is the RB whose function is to store the intermediate octave coefficients. The RB consists of 26 8-bit registers connected in series. D-type Master-Slave flip-flops shown in Figure 4.17 are used to implement the above storage units.

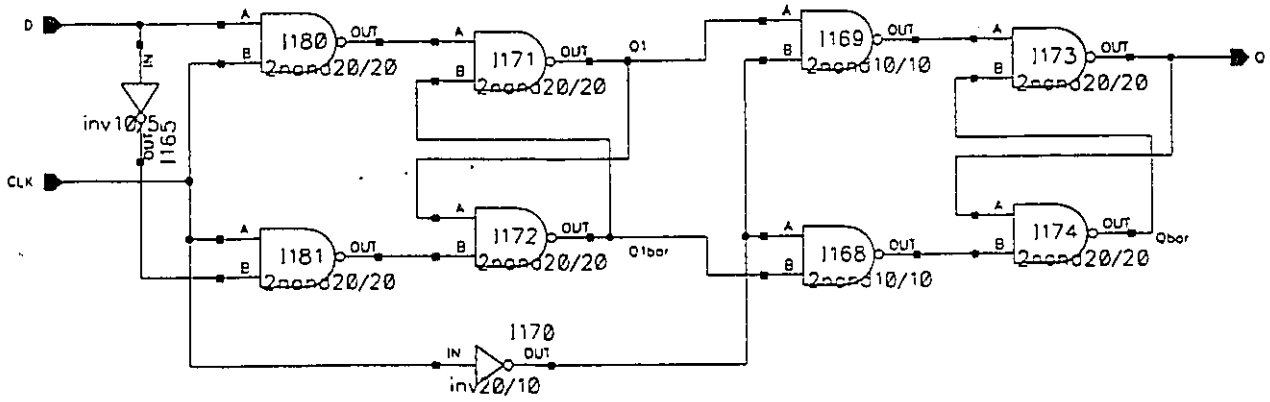


Figure 4.17: D type Master-Slave Flip-Flop.

4.4 DWT-SA Simulation Results

The DWT-SA architecture was simulated to validate its functionality and performance. Three different level simulations were performed:

- Analog gate level subcircuit simulations.
- Digital circuit level simulations.
- Digital chip level simulations.

The analog simulations were executed using the Hspice simulation tool, running under the

Opus 4.2 design platform. During analog simulations, all subcircuit transistors were sized to yield optimal speed and minimal chip area. An example of the type of work performed is presented in Section 4.4.1. Once the gate level analog simulations of all subcircuits were completed, digital simulations were performed on groups of subcircuits forming a more complex functional circuit. Larger blocks of circuits were assembled progressively and verified until the entire DWT-SA architecture was simulated. The digital simulator used was Verilog logic simulator running under Opus 4.2. The results from two digital simulations: the Booth multiplier's and the complete DWT-SA architecture's are presented in sections 4.4.2 and 4.4.3 respectively.

4.4.1 Filter Analog Simulation Results

A sample of analog simulation result is presented in Figure 4.19. The example represents the analog behavior of a Shift And Complement (SAC) cell which forms an integral part of the multiplier. In figure 4.19, the output Q is shown as the top wave. The input signals below Q correspond to COMP, S2X, 2X, SX and X respectively.

The analog waveforms in Figure 4.19, provide two pieces of important information needed to verify the design of a VLSI subcircuit. First, they verify the proper function of a subcircuit by showing the switching action in each gate. Secondly they provide time delays through each gate. These last parameters are later extracted and used in digital simulations of larger circuits.

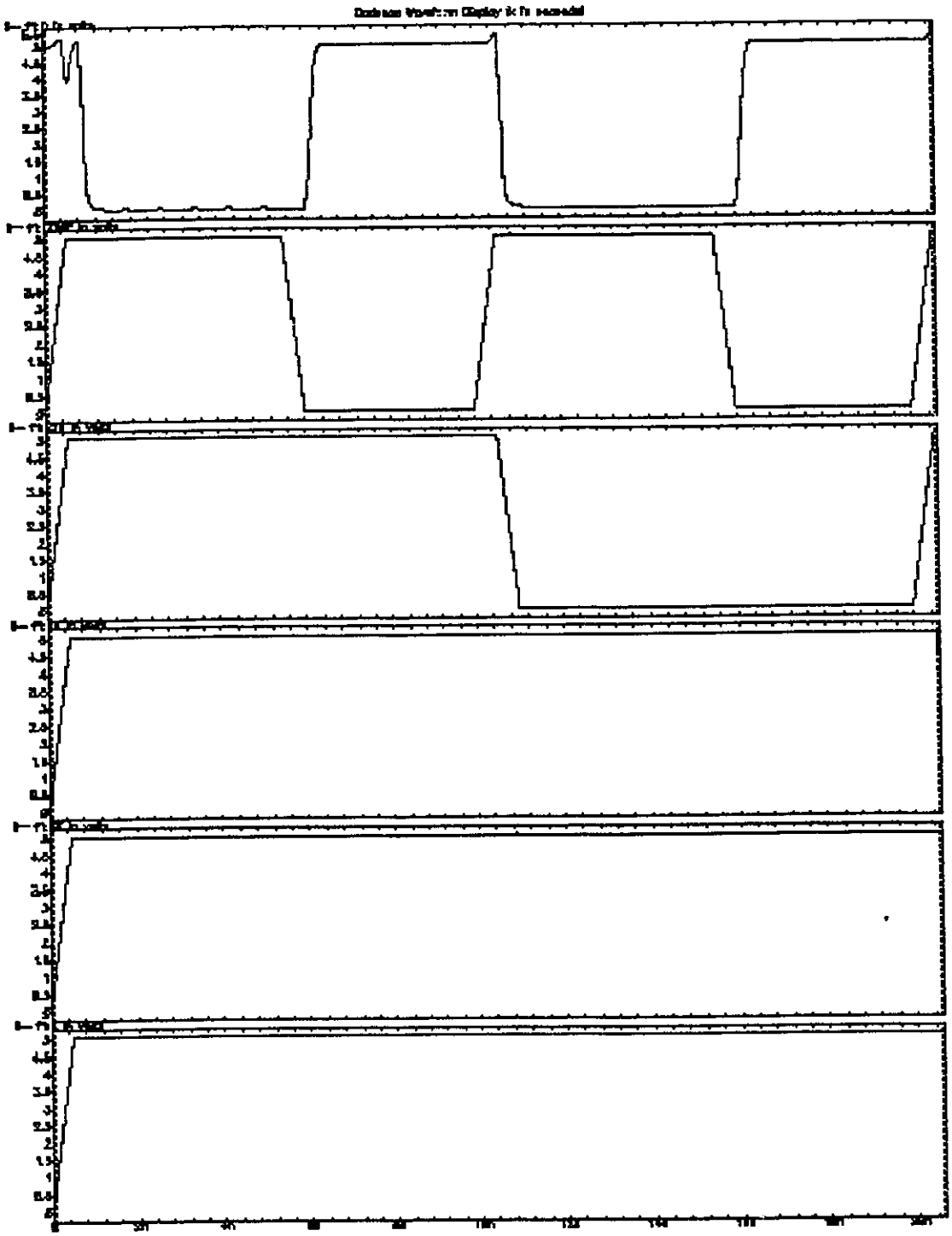


Figure 4.18: Sample analog simulations on the SAC cell.

4.4.2 Filter Digital Simulation Results

To illustrate the usefulness of the digital simulations in verification of a digital circuit, we present in Figure 4.19a and 4.19b, the results of the digital simulation of the filter. Lines 1 trough 6 in Figure 4.19.a represent the six inputs stages I(nT) through I(nT-5) as defined in Figure 3.1. Lines 7 and 8 are the CLK and the output from the filter respectively. The figure also shows that it takes 5 cycles for all filter stages to receive their respective data samples. During the sixth clock cycle, filter results (the high-pass coefficient in the first half of the clock cycle, and the low-pass coefficient in the second half of the clock cycle) are computed. Due to the systolic nature of the multiplier, more than one filter coefficient calculations are executed at the same time, and thus after the sixth clock cycle, coefficients are computed at every clock cycle.

For clarity, lines 7 and 8 in Figure 4.19a are enlarged, in Figure 4.19b and show the correct computation performed by the filter. Filter's high-pass and low-pass coefficients, the input samples as well as the computation results during the sixth clock cycle, obtained in the simulation of Figure 4.19 are summarized in Table 4.1. The results coincide with the results computed using Equations 3.1a and 3.1b.

Sample Input	High-pass Filter coefficient	Low-pass Filter coefficient
I(nT) = 2	H = 1	L = -1
I(nT-1) = 1	H = 2	L = -2
I(nT-2) = 2	H = 1	L = -1
I(nT-3) = 1	H = 2	L = -2
I(nT-4) = 2	H = 1	L = -1
I(nT-5) = 1	H = 2	L = -2
High-pass Result	12 or 000C	
Low-pass Result	-12 or FFF4	

Table 4.1: Sample Filter Simulation Results.

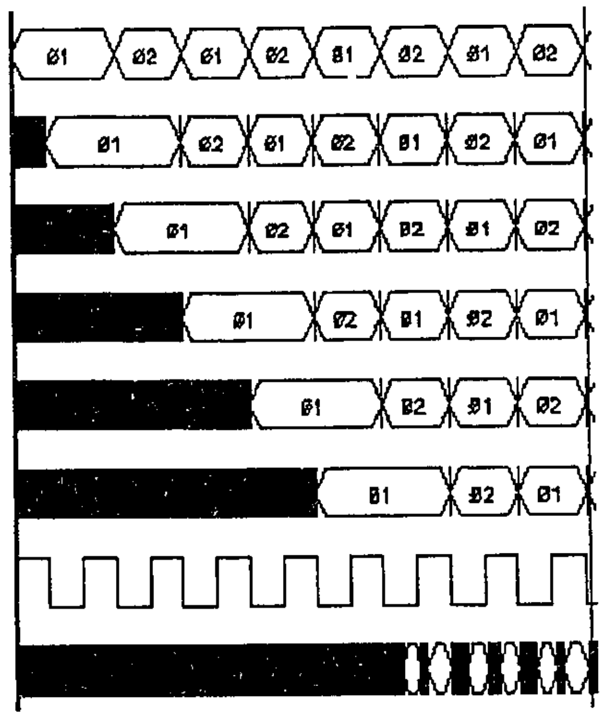


Figure 4.19a: Filter Digital Simulation Results.

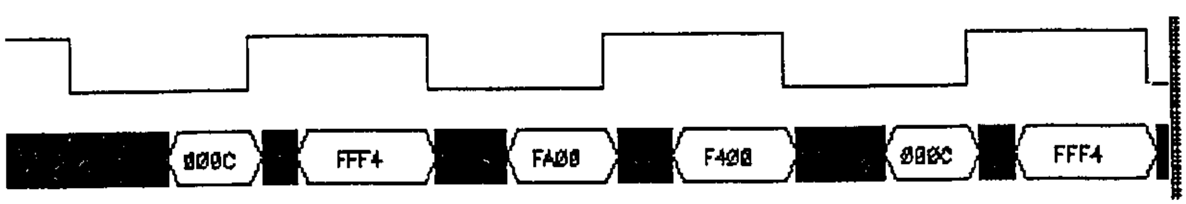


Figure 4.19b: Filter Digital Simulation Results, Partial Blow-Up.

4.4.3 DWT-SA Digital Simulation Results

Figure 4.20 shows the simulated operation at the clock speed of 20 MHz of the entire DWT-SA architecture during the first thirty eight cycles. As stated earlier, the operation of the DWT-SA architecture is periodic with period equal to 8 cycles, and has an initial pipeline delay of

38 cycles. In order to illustrate the entire operation of the DWT-SA architecture, digital simulation results after the first 38 clock cycles are depicted in Figure 4.20.

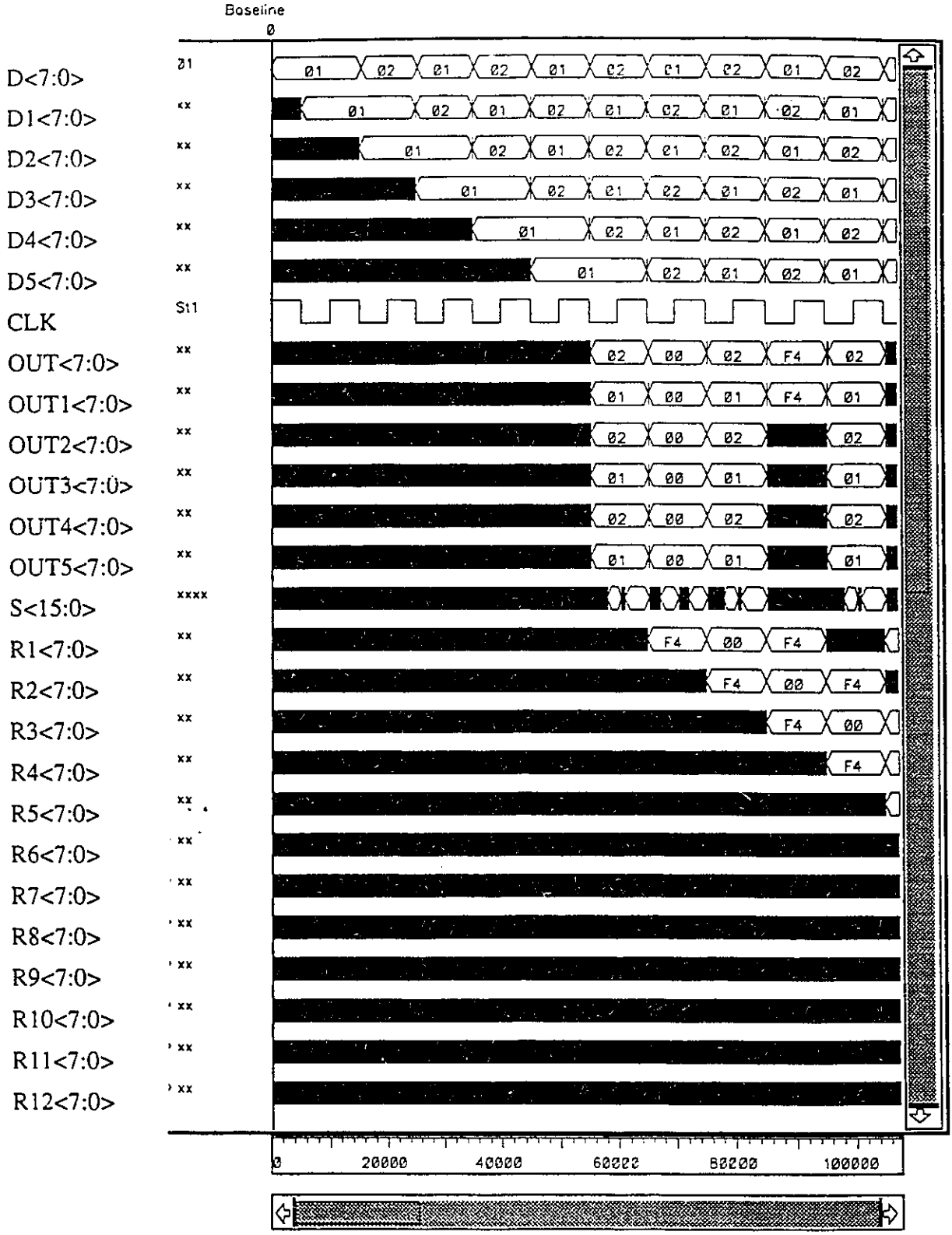


Figure 4.20: DWT-SA architecture digital simulations.

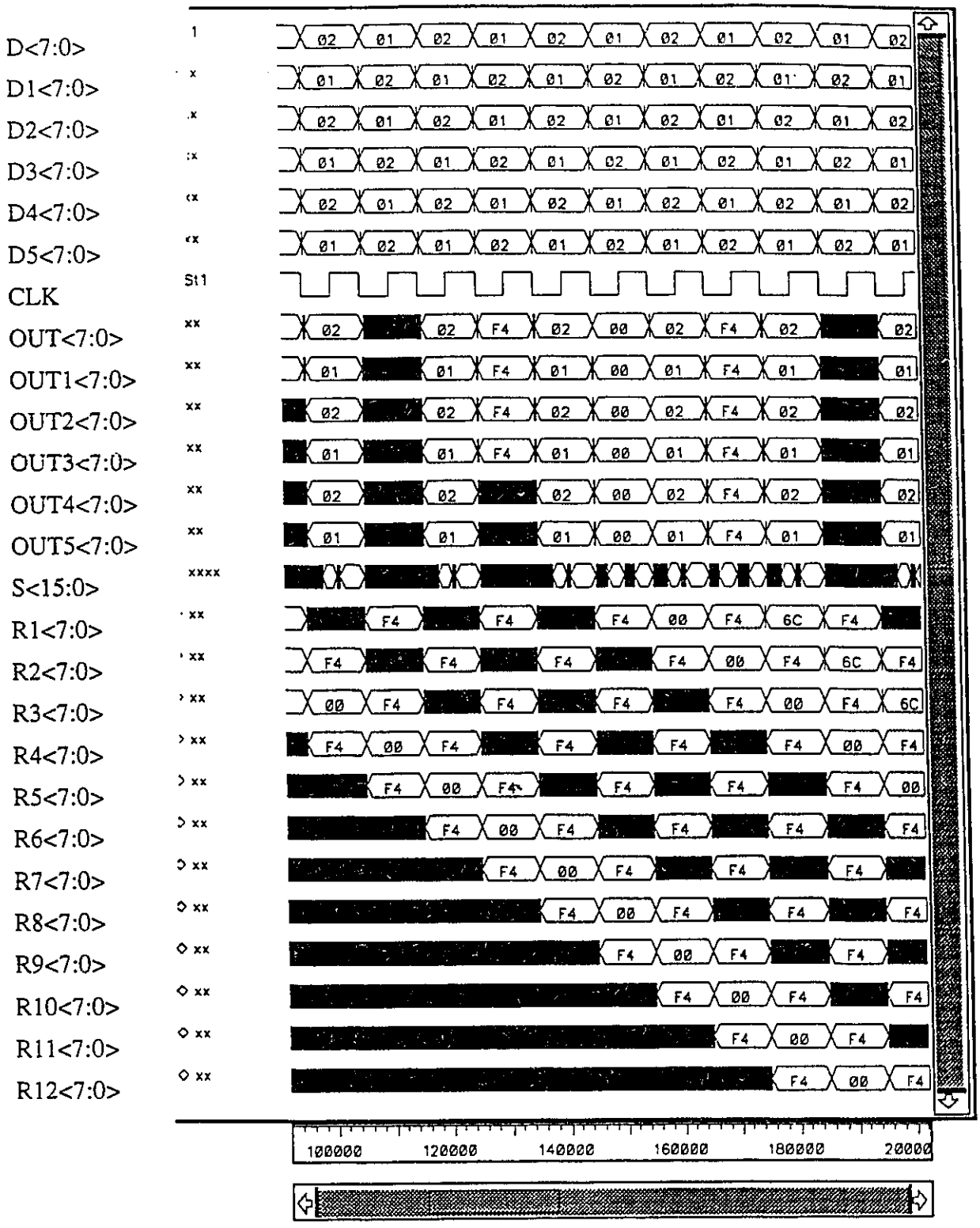


Figure 4.20: DWT-SA architecture digital simulations, (continued).

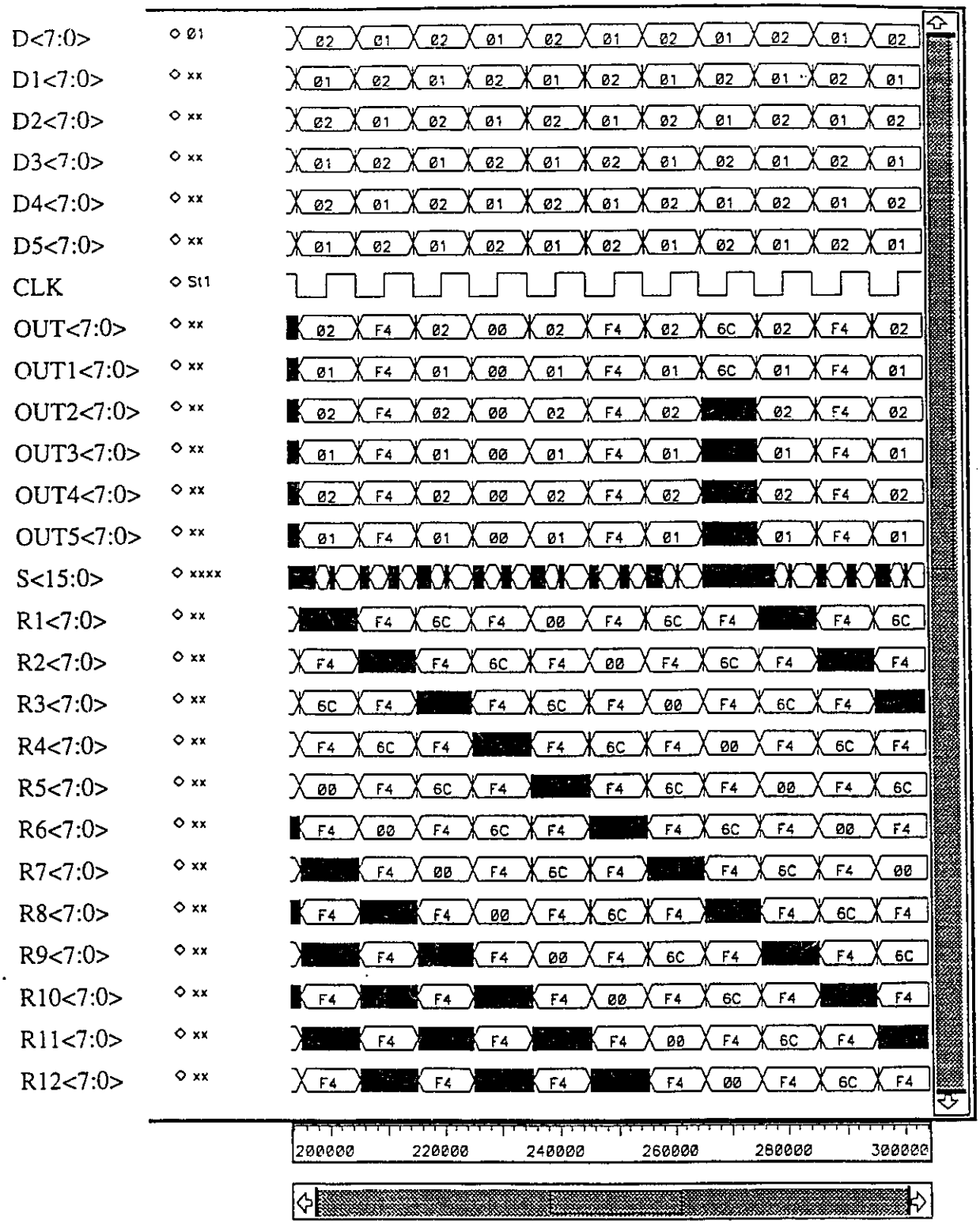


Figure 4.20: DWT-SA architecture digital simulations, (continued).

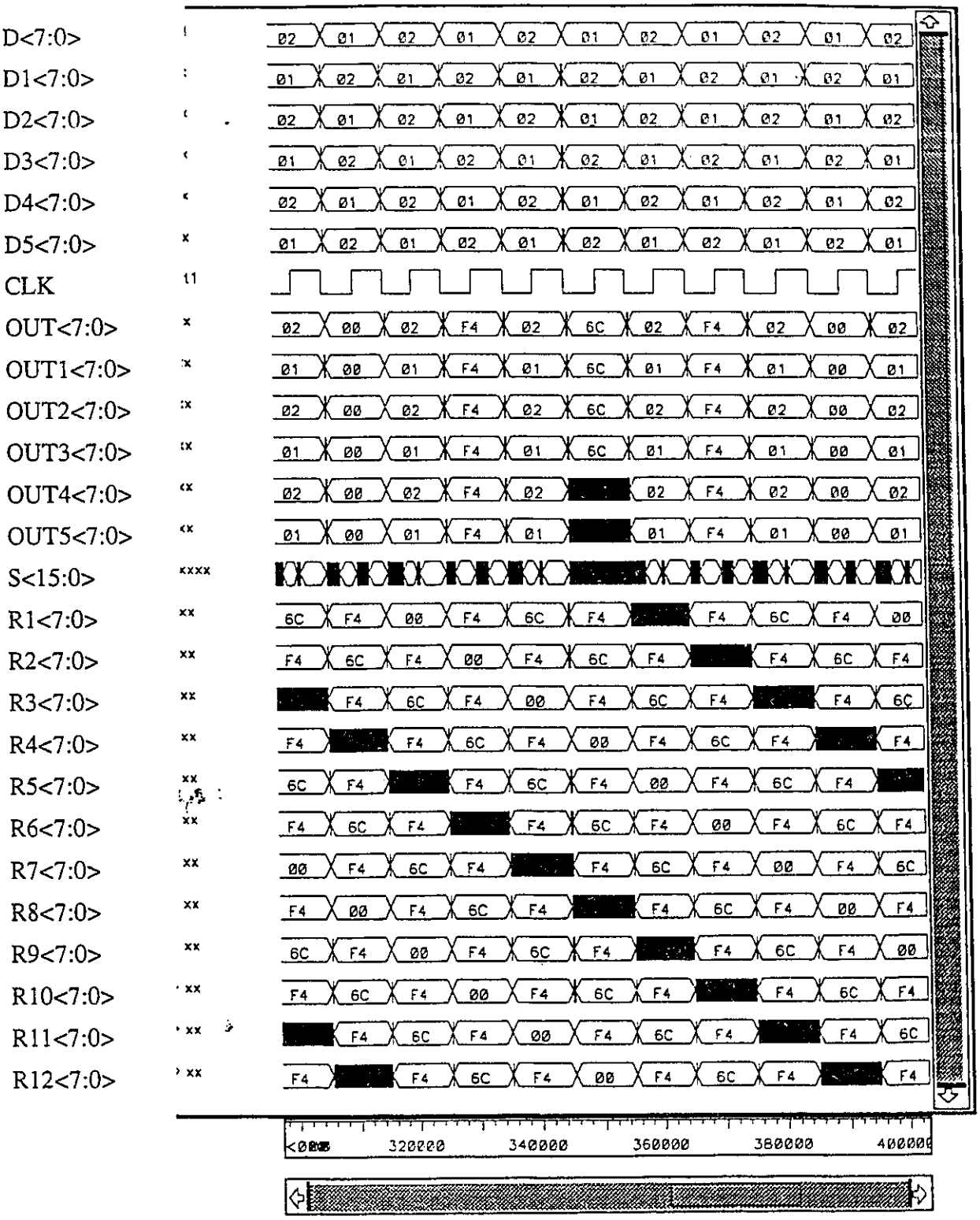


Figure 4.20: DWT-SA architecture digital simulations, (continued).

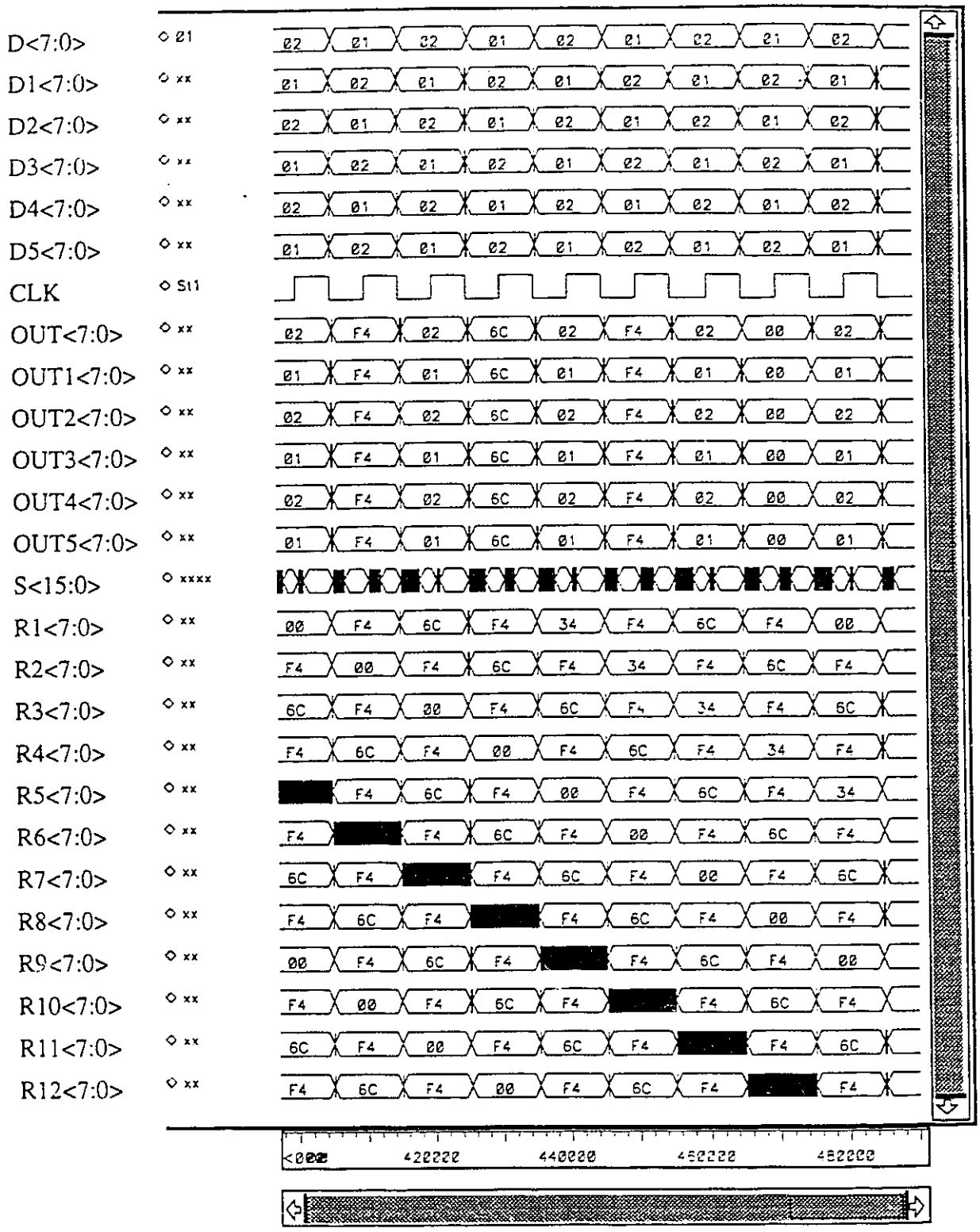


Figure 4.20: DWT-SA architecture digital simulations, (continued).

The signals in Figure 4.20 are the input sequence, it's delayed versions, clock, intermediate outputs of the six tap filter, and finally the output of the filter (or output of the DWT-SA architecture). All subsequent signals are the contents of the 26 registers.

As mentioned earlier, six cycles are necessary to fill up the input delay circuitry and the filter, with data. Only then, computations of coefficients begin. As shown in Figure 4.20, the first coefficient calculation takes place at time 500 ns. It is the first cycle of coefficient calculation. Cycle 38 corresponds to time instance 4300 ns.

Two outputs are generated per cycle, with high pass coefficient in the first half cycle, and the low-pass coefficient in the second half. Such order is necessary, because low pass coefficients are required for subsequent calculations. High pass coefficients are thus generated first and output from the chip. Low pass coefficients are stored in registers and subsequently used.

In order to facilitate reading of Figure 4.20, clock cycle numbers n and time instances t at which all final and intermediary results for the first octave are computed are shown in Table 4.2. Due to periodical nature of computations, subsequent frames coefficients are computed at time instances $t + 8$. Final DWT coefficients are shown in **bold** while the intermediate results are underlined.

High-pass coefficient	Cycle of computation	Low-pass coefficient	Cycle of computation
1st octave		1st octave	
b(0)	n=1; t=500ns	c(0)	n=1; t=500ns
b(2)	n=3; t=700ns	c(2)	n=3; t=700ns
b(4)	n=5; t=900ns	c(4)	n=5; t=900ns
b(6)	n=7; t=1100ns	c(6)	n=7; t=1100ns
2nd octave		2nd octave	
d(0)	n=12; t=1600ns	e(0)	n=12; t=1600ns
d(4)	n=16; t=2000ns	e(4)	n=16; t=2000ns
3rd octave		3rd octave	
f(0)	n=38; t=4300ns	**g(0)**	n=38; t=4300ns

Table 4.2. Table of coefficient calculations for the DWT-SA architecture.

4.5 DWT-SA Precision and Arithmetic Range.

The purpose of this thesis is to demonstrate that a 1-D or 2-D DWT can be executed efficiently in real-time using dedicated hardware. An 8-bit wide data input and filter coefficients are used in the design and the arithmetic range supported by the DWT-SA is from -127 to +127. The architecture functions properly as long as at every stage in the internal filter pipeline, there is no arithmetic overflow. In case of overflow, the intermediate result will be clamped at either one of the two extreme values.

To solve the problem of limited value range, the DWT-SA architecture can be extended to 16-bit width for the input data and filter coefficients. The change required, is simple and consists of extending the bus width throughout the DWT-SA architecture from 8 to 16 bit and designing a multiplier operating with wider operands and producing 32-bit results. That last change is also easy to implement due to a very modular and extendible multiplier design. The resulting DWT-SA

16-bit architecture would require 4 times more transistors, would occupy 4 times more silicon area consume about 4 times more in power and be about 40% slower due to longer time necessary for the multiplication operation. (Simulation results show that critical path delay through the multiplier grows less than linearly as the number of adder stages is increased). The range of values supported by the 16-bit DWT-SA is from -32767 to +32768, which is more than satisfactory for most data compression applications.

4.5 Conclusion

Gate level design of each cell in the DWT-SA architecture is presented in Chapter 4. By subdividing the chipset into four distinguishable components, the hierarchical methodology used in the design of the DWT-SA is presented. Subcircuit analog simulation results, subcircuit and circuit level digital simulation results as well as the complete digital simulation results for the overall DWT-SA chipset are presented.

Chapter 5

DWT-SA Applications

In this chapter, two system architectures for computing a 2-D DWT using the 1-D DWT-SA architecture proposed in Chapter 3 are presented. Both architectures are based on connecting the DWT-SA chipset(s) to the standard backplane VMEbus. The 2-D architectures are extendible to any image size. The architectures have a potential for real time computation of 2-D DWT by adding extra pipelining and extensive parallel processing to systolic operation of the 1-D modules.

5.1 2-D Wavelet Decomposition

The 2-D DWT decomposition is widely used in image processing applications for achieving compression. It is an extension of the one dimensional DWT operation to a two dimensional array of data and is executed as follows. First, the DWT computation is executed on input data in one direction, i.e. rows (columns) followed by a matrix transposition operation. Subsequently, the same DWT operation is performed on the columns (rows) of data[1]. The

corresponding data flow is shown in Figure 5.1.

Figure 5.1. 2-D DWT decomposition architecture.

Matrix Transposition

The matrix transposition operation can mathematically be stated as:

$$X^T{}_{ij} = X_{ji} \quad \text{for } i, j = 1,2,\cdots,N$$

For a 4 X 4 matrix, the operation is shown in Figure 5.2.

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} & X_{14} \\ X_{21} & X_{22} & X_{23} & X_{24} \\ X_{31} & X_{32} & X_{33} & X_{34} \\ X_{41} & X_{42} & X_{43} & X_{44} \end{bmatrix} \xrightarrow{Transposition} \begin{bmatrix} X_{11} & X_{21} & X_{31} & X_{41} \\ X_{12} & X_{22} & X_{32} & X_{42} \\ X_{13} & X_{23} & X_{33} & X_{43} \\ X_{14} & X_{24} & X_{34} & X_{44} \end{bmatrix}$$

Figure 5.2. Transposition of a matrix.

Several matrix transposition architectures have been proposed in literature[31,32,33]. However, these architectures lack the modularity and cascadability necessary to achieve real time

2-D DWT operation using the 1-D transformation presented in chapter 3. The one architecture modular enough to adequately perform matrix transposition required by the 2-D DWT architecture is presented in [34]. It possesses the necessary serial and parallel loading capability as well as cascadability to perform transposition of a matrix of any size.

It consists of three basic components: transposer module (**TM**), input module (**IM**), and output module (**OM**).

TM is a $N \times N$ matrix of N^2 basic cells interconnected as N rows of cells with each row consisting of N column of cells as shown in Figure 5.3. It has two modes of interconnection: the A-mode and the B-mode.

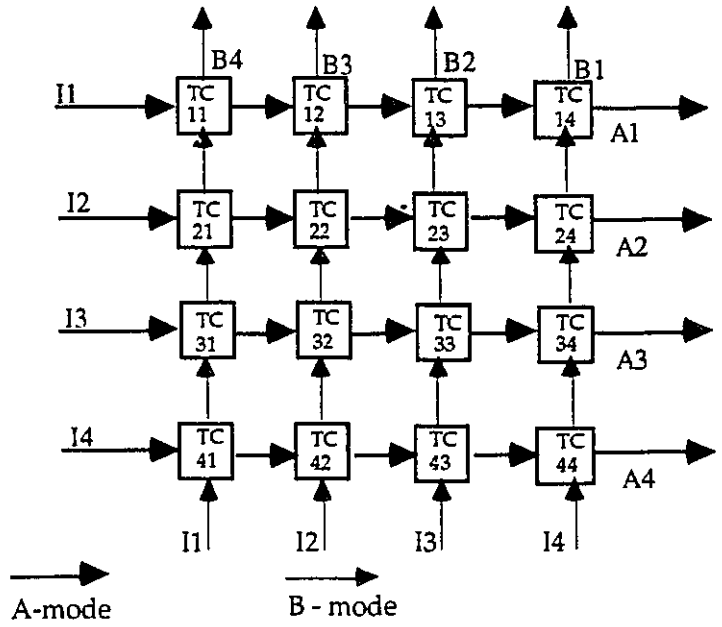


Figure 5.3. Transposer module.

Cells are connected from left to right in the A-mode and from bottom to top in the B-

mode. A row (column) of data is loaded in and out of the module in each clock cycle. In the A-mode a row (column), initially the first row (column), of data of a matrix is loaded in parallel into the cells TC11-TC41 at every clock cycle. Meanwhile, the second row (column) of data is prepared to be loaded into the **TM** by the **IM**.

In the next clock cycle the second row is loaded into the transposer cells **TC** TC11-TC41 while the first row (column) of data moves to the cells TC12-TC42. The procedure continues and at the end of four clock cycles all the rows (columns) of data are loaded into **TM**.

IM: A series of multiplexers necessary for selection of serial or parallel input. It is composed of N flip-flops and N 2-to-1 multiplexers.

OM: A series of multiplexers necessary for selection of A mode or B mode sequence. It is composed of N flip-flops and N 2 to 1 multiplexers.

The transposer operates in a systolic manner as follows. First it accepts data in a parallel or serial mode in the IM (serial input necessitates realigning of input data). Secondly, it passes the aligned input data to the TM. Finally, the OM propagates the transposed matrix data out of the transposer. Note that the TM works in parallel only, and therefore the IM should work at a clock rate that is equal to N times that of the TM where N is the one dimensional size of the matrix.

5.2 Interconnection standards and DWT-SA

Several commercial interconnection standards have evolved in the computer field. These standards are intended to facilitate the interconnection of equipment manufactured by a wide variety of companies. Many commercial standards are "de facto standards", due to their widespread popularity. One such widely accepted interconnection standard is the VMEbus which we have

selected as the method of choice for connecting and using the DWT-SA chipset.

5.2.1 VMEbus overview

The VMEbus is a sophisticated backplane bus intended for high-performance microprocessor systems. It has several advanced features that make it suitable for use in both single-processor and multiprocessor systems.

There are several options available for configuring a system based on the VMEbus. The bus may have 8, 16, or 32 data lines and may use 16, 24, or 32-bit addressing.

The signals on the VMEbus are divided into four groups, as follows:

1. Data transfer bus (DTB).
2. DTB arbitration.
3. Priority Interrupt.
4. Utilities.

The DTB comprises all lines used for transferring data during read and write operations. They include the address and data lines and the control lines needed to carry timing signals and information such as the operand size and the direction of the transfer. All VMEbus data transfer lines are summarized in Table 5.1.

Data transfer over the VMEbus can take place in 8-bit, 16-bit, or 32-bit quantities. The number of address bits used may be 16, 24, or 32. Read and write requests are differentiated by means of the WRITE* line, which is activated for write operations only. The timing of a data transfer is controlled by asynchronous handshake signals. The master activates the Address signals, AS*, DS0*, and DS1*, and the slave responds by activating the Data-Acknowledgment signal, DTACK*.

Designation	Function	Driver	Number
A01 - A31	Address	3ST	31
AM0 - AM5	Address modifier	3ST	6
D00 - D31	Data	3ST	32
WRITE*	Write operation	3ST	1
AS*	Address strobe	3ST - HC	1
DS0*, DS1*	Data strobe	3ST - HC	2
LWORD*	Long word	3ST	1
DTACK*	Data acknowledge	OC	1
BERR*	Buss error	OC	1
Total			76

Table 5.1. VMEbus data transfer lines.

Irrespective of the address, data are always placed on the data lines such that the least-significant data bit is in line D00. A byte transfer always uses lines D07 to D00 and a word transfer uses lines D15 to D00. Hence, a 32-bit device, such as a memory board, must incorporate appropriate multiplexers to shift byte or word data to the low-order data lines, to enable byte or word transfers to take place.

Shifting data to the low-order section of the bus is needed to simplify the connection of 8 and 16-bit devices. Recall that in the proposed DWT architecture, the coefficients as well as input data are all 8 bit, and the output data is 16 bit. By using WMEbus, reading in the input samples as well as writing out the output samples is greatly simplified.

5.3 2-D DWT Architectures

There are two methods for achieving 2-D DWT. The first method called the **simple 2-D DWT-SA** uses only one DWT-SA chipset tc compute both of the 1-D DWT transformations in

Figure 5.1. It therefore requires that the first 1-D DWT results be stored temporarily in memory, and once the transposition has been completed be fed back to the DWT-SA. Due to sharing of resources between the first and second 1-D DWT computations, that scheme excludes the possibility of computing 2-D DWT in real-time as defined in Chapter 3.

The second method called the **extended 2-D DWT-SA**, uses two 1-D DWT dedicated hardware to compute the 1-D DWT in Figure 5.1. This scheme speeds up the computation of the 2-D DWT.

5.3.1 Simple 2-D DWT Architecture

In the simple 2-D DWT-SA architecture both row and column DWT computations are performed by the same DWT-SA chipset. Rows coefficients are computed and stored in memory, subsequently are realigned in software, and fed back to the chipset for column computation.

The DWT-SA chipset is a simple, dedicated engine for computing DWT coefficients. It is comprised of the following input and output lines:

- Signals FH0 - FH7 and FL0 - FL7 are the two 8-bit busses respectively use for loading the high-pass and low-pass filter coefficients. The busses are employed to input all six high-pass and six low-pass filter coefficients, a pair per clock cycle. This approach was chosen to minimize the number of pins required. Six clock cycles are required to set all filter coefficients.
- An 8-bit data input bus (D0 - D7) supplies the input data samples.
- A 16-bit (O0 - O15) tristate output bus feeds the computed DWT coefficients when the R/W signal is low.

- A RST line is provided to reset the DWT-SA chipset.

With the exception of the CLK and RST lines, all other signals in the 2-D DWT-SA simple architecture, share the 32 bit data bus as follows: 8 bit high-pass filter coefficients are connected to data lines D00 to D07, whereas the low-pass filter coefficient bits are connected to lines D07 - D15. That arrangements permits simultaneous loading of a pair of filter coefficients. Data sample bits 0 to 7 are connected to lower 8 bits D00 - D07 of the data bus, and the 16 output bits to lines D00 - D15.

To ensure proper functioning of the proposed architecture interfaced to the VMEbus, a clock synchronized with the data output signals of the architecture is necessary. Since the DTB buts is the medium on which data is fed into the DWT architecture as well output from, it must be shared in a way that allows for bi-directional flow. The way to achieve this functionality is to allow for data to be stored and retrieved in every cycle, as shown in Figure 5.4. Here, read and write operations follow one another, with the read to DWT-SA operation occupying the first half of the R/W cycle and the write from DWT-SA operation using the second half of the R/W cycle. The scheme is made possible because of the fact that only one (high-pass coefficient) of two coefficients computed by the filter is output from the architecture; the other, as explained earlier is stored in a local register for higher octave processing.

It should be noted that once every eight cycles, two highest octave coefficients (one high-pass and one low-pass) are produced and must be output. This implies that the data bus has to be available for two consecutive write operations, situation that could create conflicts on the data bus. The solution to the problem can be found by reexamining the schedule for one complete set of calculations in Table 3.1. It is shown that one out of eight clock cycles, two DWT coefficients are computed (cycle 6). Similarly, in one out of eight cycles, no DWT computations take place. This suggests that one of the computed output coefficients (the high-pass coefficient) can be output

(written to DTB) in a manner similar to all the other DWT coefficients. The second output coefficient can be stored in the Register Bank and be output during the idle clock cycle which takes place in cycle 2, 4 clock cycles later. Register Allocation in Table 3.2 shows that in the time period from cycle 6 to cycle 2, the coefficient will have traveled to register RB4, and thus must be written to DTB from that location. It is for that purpose that in Figure 3.6, the output of the filter and the output of the RB4 are multiplexed. In clock cycle 2 the DWT-SA output is taken from RB4, whereas in all other time instances it is taken from the filter output. This approach removes the possible conflicts and bus contentions and consequently results in an orderly read-write DWT-SA - DTB transfer as shown in Figure 5.4.

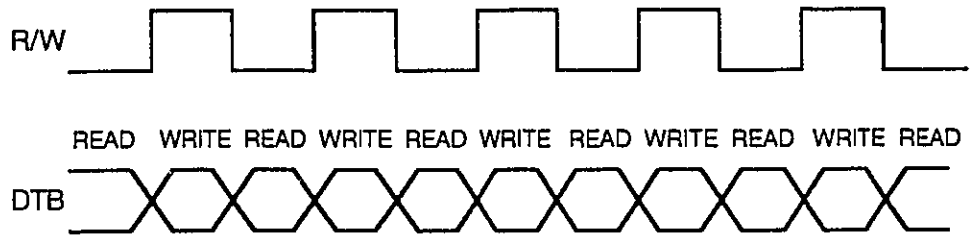


Figure 5.4. DWT Architecture - DTB data transfer.

The designed chipset can be considered as dedicated hardware accelerator, that computes DWT coefficients, but does not realign data that has passed through it. These functions have to be taken care of by other system components. A possible scheme is presented in Figure 5.5.

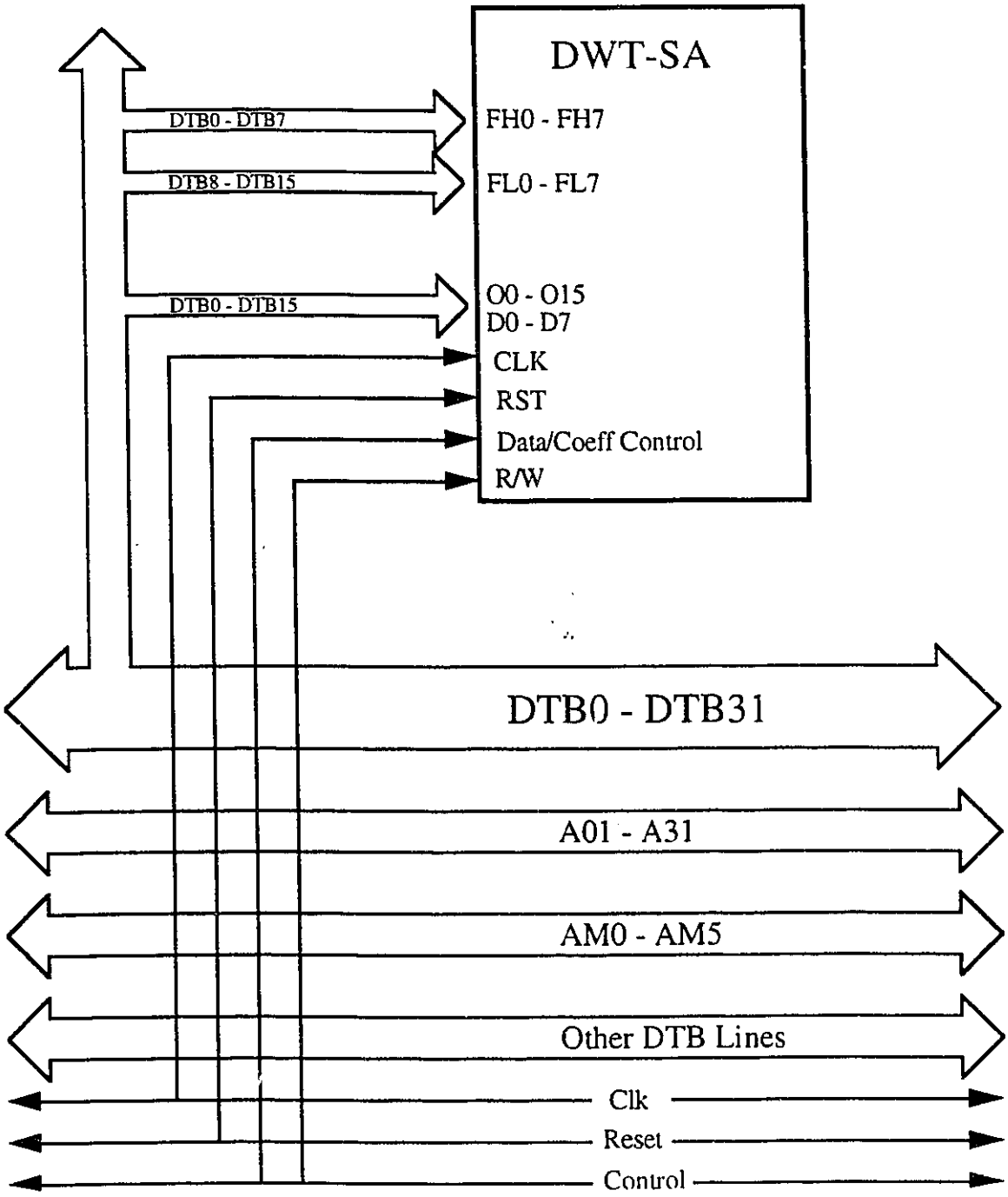


Figure 5.5. VME Interface for the 2-D DWT-SA simple architecture.

5.4 Extended 2-D DWT Architecture

The extended 2-D DWT-SA architecture is proposed in order to exploit fully the parallelism present in the 2-D DWT transform, and achieve real-time operation. In comparison to the simple 2-D DWT-SA architecture presented earlier, the extended 2-D DWT-SA architecture operates with two DWT-SA chipsets, one of which performs the 1-D DWT on the image rows, and the other executes the 1-D DWT on the image columns. The architecture for the extended 2-D DWT-SA is shown in Figure 5.6.

The two DWT-SA chipsets are separated from the VMEbus by their respective microcontrollers and the transposer unit, which have the following tasks:

- Both microcontrollers receive filter coefficients for their respective DWT-SA chipsets and assure that they are properly set up.
- Microcontroller 1 receives data input from the VMEbus DTB and passes the data to its DWT-SA 1 chip for processing. It also receives the results and sets them in sequence before they are fed into the transpose module. Only after a row of input data has been processed completely and all its coefficients have been calculated, can they be realigned and subsequently passed onto the matrix transposer.
- The matrix transposer receives input one row at a time and after N such inputs it produces first transposed output column.
- Microcontroller 2 receives the transposed data and feeds it to the second DWT-SA 2 chipset for the stage of columns processing. It also receives the final results and sets them in sequence before they are output to the VMEbus DTB.

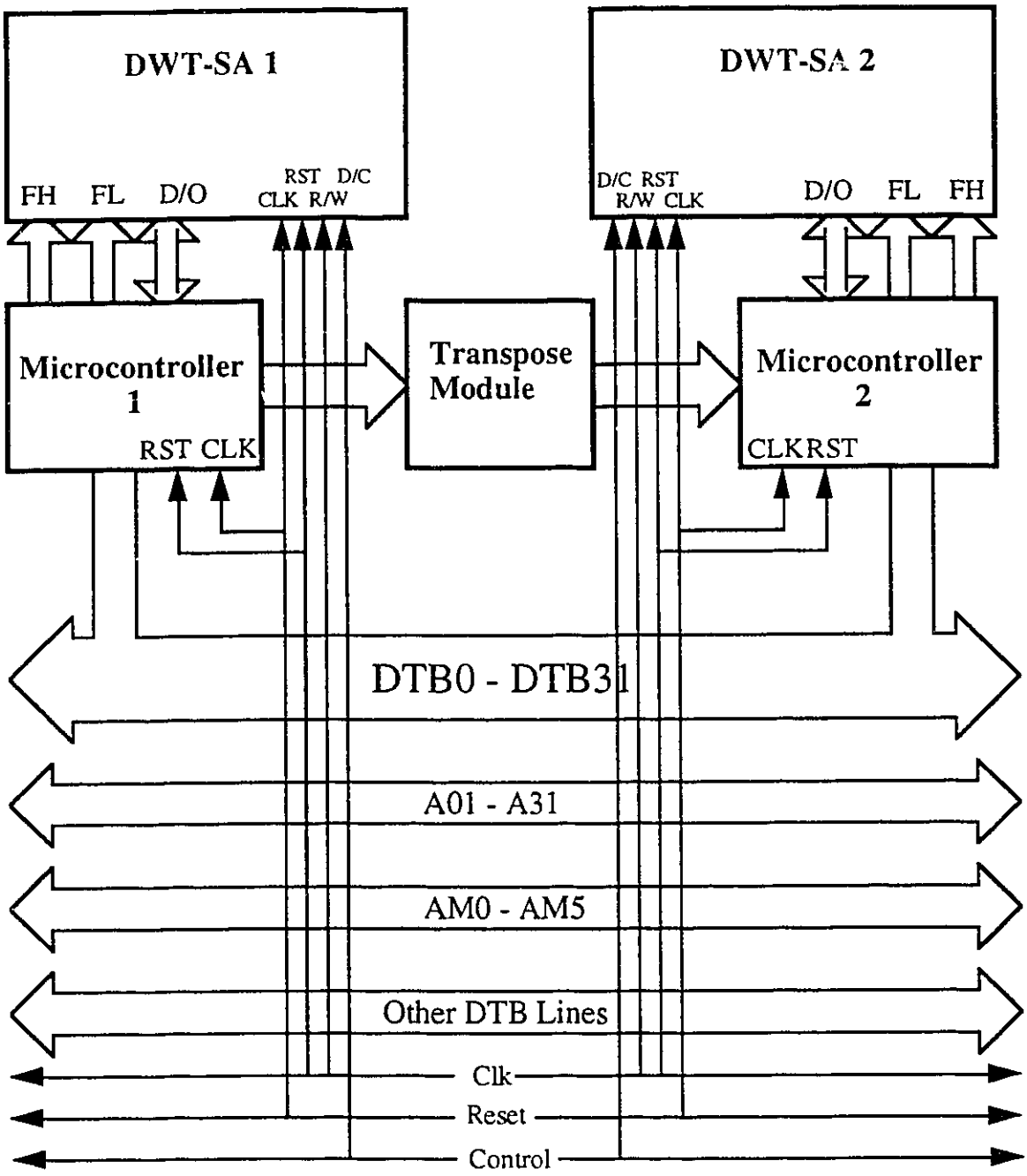


Figure 5.6. Extended 2-D DWT-SA architecture.

5.4 Extended 2-D DWT Performance Estimates

Full size 512 X 512 pixel video input requires complete DWT coefficient calculation every 33 ms, which corresponds to a standard rate of 30 images per second.

A typical image is composed of 512 rows, which in turn contain 512 pixels each, the total number of DWT coefficients is 262144, corresponding to one DWT coefficient for every pixel. Considering that the input to the architecture is sequential, on a row by row and pixel by pixel basis, it takes 38 + 512 clock cycles to compute a complete set of DWT coefficients for each row in the input image. As explained previously 38 cycles is the initial pipeline delay through the 1-D DWT architecture. At the simulated and achievable clock rate of 20 MHz, 550 clock cycles (512+38) correspond to 27.5 μs. Since there are 512 such calculations to be performed in order to complete 1-D DWT computations for all rows, the time required to complete the task is 14.08 ms.

As complete rows of DWT coefficients became available they are realigned by the control circuitry and subsequently fed into the matrix transposer module. This approach results in extracting additional parallelism from the architecture and suggests that realigning is performed at no additional cost to the overall delay through the 2-D architecture. Equally negligible delay is associated with the matrix transposition operation. That is due to the fact that by the time the last row of coefficients is being calculated, N-1 rows of previously computed coefficients are already in the matrix transposer module waiting only for the last row of coefficients to be input. Thus the delay through the matrix transposer equals 512 + 2 cycles i.e. 514 cycles. The two additional cycles account for the delay trough the input and output modules of the transposer. As a result, it takes 14.08 ms + 0.0257 ms = 14.1 ms to get all rows of 2-D input data trough the first 1-D DWT computation, coefficient alignment as well as matrix transposition. At this point sequential data will be processed by the second 1-D DWT architecture, but now on column by column basis. It can be concluded, that 14.08 ms later, all 2-D DWT coefficients for the 2-D input stream are computed, for a total of 28.18 ms.

The second 1-D DWT computation is performed on data temporarily stored in the matrix transposer. This suggests that the moment that operation begins, the first 1-D module is free to start computing the DWT coefficients of the following input frame. As a results partial computations of two sequential frames are executed at the same time. Consequently, the time required for computing one frame of DWT coefficients is just the time required for computing the first 1-D transformation i.e. 14.1 ms. In this manner, the overall hardware utilization and efficiency is close to 100%.

It can be concluded that full frame 512 X 512 monochrome video can be processed in less than half the time required for real time computation.

In the case of color video, three color planes instead of one have to be computed in the time allowed of 33 ms. That corresponds to 11 ms time allocation for each color plane. In order for this architecture to successfully process color video in real time, it would have to function at a faster clock rate. Moving to smaller than 1,2 μm technology is required because the speed of computation is directly related to the process technology. 30% Speed improvement of up to 30%, can be achieved with a process technology between 0.6μm - 0.8μm. In 1995, such technology is becoming a standard used by most VLSI manufacturers, suggesting that in the very near future, a 2-D DWT architecture just presented will be able to successfully process full motion, standard size video input signals.

5.6 Conclusion

In this chapter two-dimensional DWT computation using the DWT-SA architecture is introduced. Two 2-D DWT architectures, simple and extended 2-DWT architectures using one and two DWT-SA chipsets respectively are presented. Performance estimates for the 2-D DWT-SA extended architecture are presented.

Chapter 6

Summary and Future Work

6.1 Summary

In this thesis, a new simple and efficient VLSI architecture (DWT-SA) for computing the Discrete Wavelet Transform has been presented. The proposed architecture is systolic in nature, modular and extendible to 1-D or 2-D DWT transform of any size. The DWT-SA has been designed, simulated and implemented in silicon.

The following are the features of the DWT-SA architecture:

- It has an efficient (close to 100%) hardware utilization.
- It works with data streams of arbitrary size.
- The design is cascadable, for computation of one, two or three dimensional DWT.
- It requires a minimum interface circuitry on the chip for purposes of interconnecting to a standard communication bus.

The design of DWT-SA, is based on a computation schedule derived from the DWT Equations 3.2a - 3.2n, which are the result of applying the Pyramid Algorithn for N=8 (Fig. 2.2), to the six stage filter equations 3.1a - 31.b. The high-pass and the low-pass filter coefficients can be computed using a six stage multiply and accumulate digital filter, where partial results are calculated at the each stage separately and then progressively passed to the next higher stage for accumulation.

Minimum number of intermediate registers are used in the design to store different octave outputs before they are used in the next octave computations. A high-speed multiplier based on the Booth Algorithm is employed in the filter.

The DWT-SA design has been implemented using CMOS 1.2 um technology.

Two system architectures for computing a 2-D DWT using the 1-D DWT-SA architecture proposed in Chapter 3 are presented. Both architectures are based on connecting the DWT-SA chipset(s) to the standard backplane VMEbus. The 2-D architectures are extendible to any image size and have a potential for real time computation of 2-D DWT.

6.2 Future Research Work

The DWT-SA architecture, proposed in this thesis is a proof of concept. Its intention is to demonstrate that a 1-D or 2-D DWT can be executed efficiently in real-time in simple, dedicated hardware. The architecture is constructed using 8-bit wide data and filter coefficients. Due to this constraint, the arithmetic range supported by the DWT-SA ranges -127 to +127. The architecture functions properly as long as at every stage in the internal pipeline as well as in the filter, the intermediate result is not greater than +127 or smaller than -127. Violation of this rule results in an output clamped at either one of the two extreme values leading to an imprecise result.

Future research work consists of extending the DWT-SA architecture to a 16-bit

architecture. The change required, is simple and consists of extending the bus width throughout the DWT-SA architecture from 8 to 16 bit and designing a multiplier operating with wider operands and producing a 32-bit result. That last change is also easy to implement due to a very modular and extendible multiplier design. The resulting DWT-SA 16-bit architecture will require 4 times more transistors, would occupy 4 times more silicon area consume about 4 times more in power and be about 40% slower due to longer delay necessary for the multiplication operation. (Simulation results show that critical path delay through the multiplier grows less than linearly as the number of adder stages is increased). The range of values supported by the 16-bit DWT-SA is from -32767 to +32767, which is more than satisfactory for most data compression applications.

Another possibility for future work consists of porting the DWT-SA simulations and layout onto a newer process technology, for example 0.5μm. That improvement would almost certainly result in real-time execution of DWT on color video.

Finally, a completely new design methodology based on algorithm synthesis (by using the Synopsis tool) can be explored. Due to the systematic data flow through the architecture which can be expressed by a set of equations, the Synopsis top-down approach has the capacity to dramatically cut the design, simulation and verification time required.

References

[1] N. Jayant, "High Quality Networking of Audio-Visual Information", *IEEE Communications Magazine,* pp. 84-95, September 1993.

[2] E. A. Fox, "Advances in Interactive Digital Multimedia Systems", *IEEE Computer Magazine,* pp. 9-21, October 1991.

[3] P. H. Ang, P. A. Ruetz and D. Auld, "Video Compression Makes Big Gains", *IEEE Spectrum Magazine,* pp. 16-19, October 1991.

[4] A. K. Jain, "Image Data Compression: A Review", *Proceedings of the IEEE,* Vol. 69, No. 3, pp. 349- 389, March 1981.

[5] K. R. Rao and P. Yip, *Discrete Cosine Transform - Algorithms, Advantages, Applications,* Academic Press: San Diego, 1990.

[6] N. M. Nasrabadi and R. A. King, "Image Coding Using Vector Quantization: A Review", *IEEE Transactions on Communications,* Vol. 36, No. 8, pp. 957-971, August 1988.

[7] H. G. Musmann, P. Pirsch and H. J. Grallert, "Advances in Picture Coding", *Proceedings of the IEEE,* Vol. 73, No. 4, pp. 523-548, April 1985.

[8] A. N. Netravali and B. G. Haskell, *Digital Pictures,* Plenum: New York, 1989.

[9] FBI Systems Technology Unit: "Gray Scale Fingerprint Image Compression Status Report, Nov. 4 1991.

[10] Brower, B., et al.: The National Imagery Transmission Format Standard Bandwidth Compression Study, Phase II, Final Report, Eastman Kodak, Federal Systems Division (1991).

[11] Brower, B., et al.: The National Imagery Transmission Format Standard Bandwidth Compression Study, Phase III, Part 2, Preliminary Report, Eastman Kodak, Federal Systems Division (1993).

[12] Knowles G., "VLSI Architecture for the Discrete Wavelet Transform," *Elec. Letters,* Vol. 26, No. 15, pp. 1184-1185, Jul. 1990.

[13] C. Chakrabarti, M. Vishwanath and R. M. Owens, "Architectures for Wavelet Transforms", *Proc. IEEE VLSI Signal Processing Workshop.* pp. 507-515, 1993.

[14] M. Vishwanath, "Discrete Wavelet Transform in VLSI", *Proc. IEEE Int. Conf. Appl. Specific Array Processors.* pp. 218-229, 1992.

[15] K. K. Parhi and T. Nishitani, "VLSI Architectures for Discrete Wavelet Transforms", IEEE Trans. VLSI Systems, vol. pp. 191-202, June 1993.

[16] K.K. Parhi, "A systematic approach for design of digit -serial signal processing architectures", *IEEE Trans. Circuits Syst.*, vol. 38, pp. 358-375, Apr. 1991.

[17] K.K. Parhi, "Systematic synthesis of DSP data converters using life-time analysis and forward-backward register allocation", *IEEE Trans. Circuits Syst-II*, vol. 39, pp. 423-440, July 1992.

[18] K. K. Parhi, "Video data format converters using minimum number of registers", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 38, pp. 255-267, June 1992.

[19] Yi Kang, "Low-power design of wavelet processors", IEEE Transactions on SPIE. vol. 2308, pp. 1800-1806, 1993.

[20] Aware Wavelet Transform Processor (WTP) Preliminary, Aware Inc., Cambridge, MA.

[21] W. B. Pennebaker and J. L. Mitchell, JPEG Still Image Data Compression Standard, Van Nostrand Reinhold: New York (1993)

[22] M. Vishwanath, "The Recursive Pyramid Algorithm for the Discrete Wavelet Transform", To appear in the IEEE Trans. on Signal Processing, March, 1994.

[23] Gerben J. Hekstra, *Multiplier Architectures for VLSI Implementation,* Technische Universiteit Delft, Technisch Rapport no. 90-104, November 1990.

[24] Andrew D. Booth, "A Signed Binary Multiplication Technique", *Quarterly Journal*

of Mechanics and Applied Mathematics, Vol. 4, pp.236-240, 1951.

[25] Neil Weste and Kamran Eshragian, *Principles of CMOS VLSI Design*, Addison-Wesley, June 1988.

[26] Joseph J.F. Cavanagh, *Digital Computer Arithmetic,* Mc. Graw-Hill, 1984.

[27] W. B. Kleijn, "Encoding speech using Prototype Waveforms", *IEEE trans. on Speech and Audio Processing,* Vol. 1, No. 4, pp. 386-399, October 1993.

[28] H. Szu, B. Telfer and S. Kadambe, "Neural network adaptive wavelets for signal representation and classification," *Journal of Optical engg.,* Vol. 31, pp. 1907-1916, September 1992.

[29] S. Kadambe and P. Srinivasan, "Application of adaptive wavelets for speech coding," *Proceedings of the IEEE-SP International Symposium on Time-Frequency and Time-Scale Anaysis,* Philadelphia, Pennsylvania, October 25-28 1994.

[30] Anil K. Jain, "Fundamentals of Digital Image Processing", Prentice-Hall Inc., 1989.

[31] Bowen, B.A and Brown, W.R: 'Systems design. vol. II" (Prentice-Hall Inc., 1985)

[32] Eklundh, J.O.: "A fast computer method for matrix transposing", IEEE Trans., 1972, C-21, (7), pp.801-803

[33] O'Leary, D.P.:"Systolic arrays for matrix transpose and other reorderings', IEEE Trans., 1987, C-36, (1), pp. 117-122

[34] S. Panchanathan, " Universal architecture for matrix transposition" ,IEE PROCEEDINGS-E Vol. 139 No. 5 sep. 1992

Appendix A

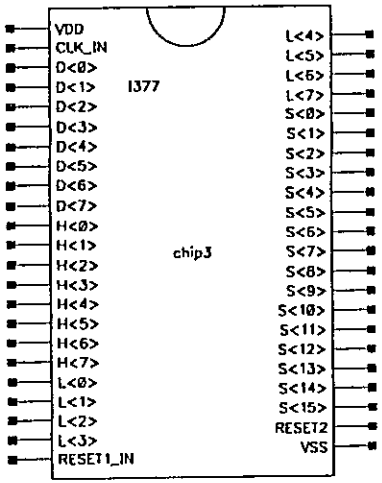
Architecture Schematics

The following pages contain a complete hierarchical set of schematics for the proposed DWT-SA architecture.

CAGE NO.	DWG NO.	SH	REV
Alex	Grzeszczak		Ø.1

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED



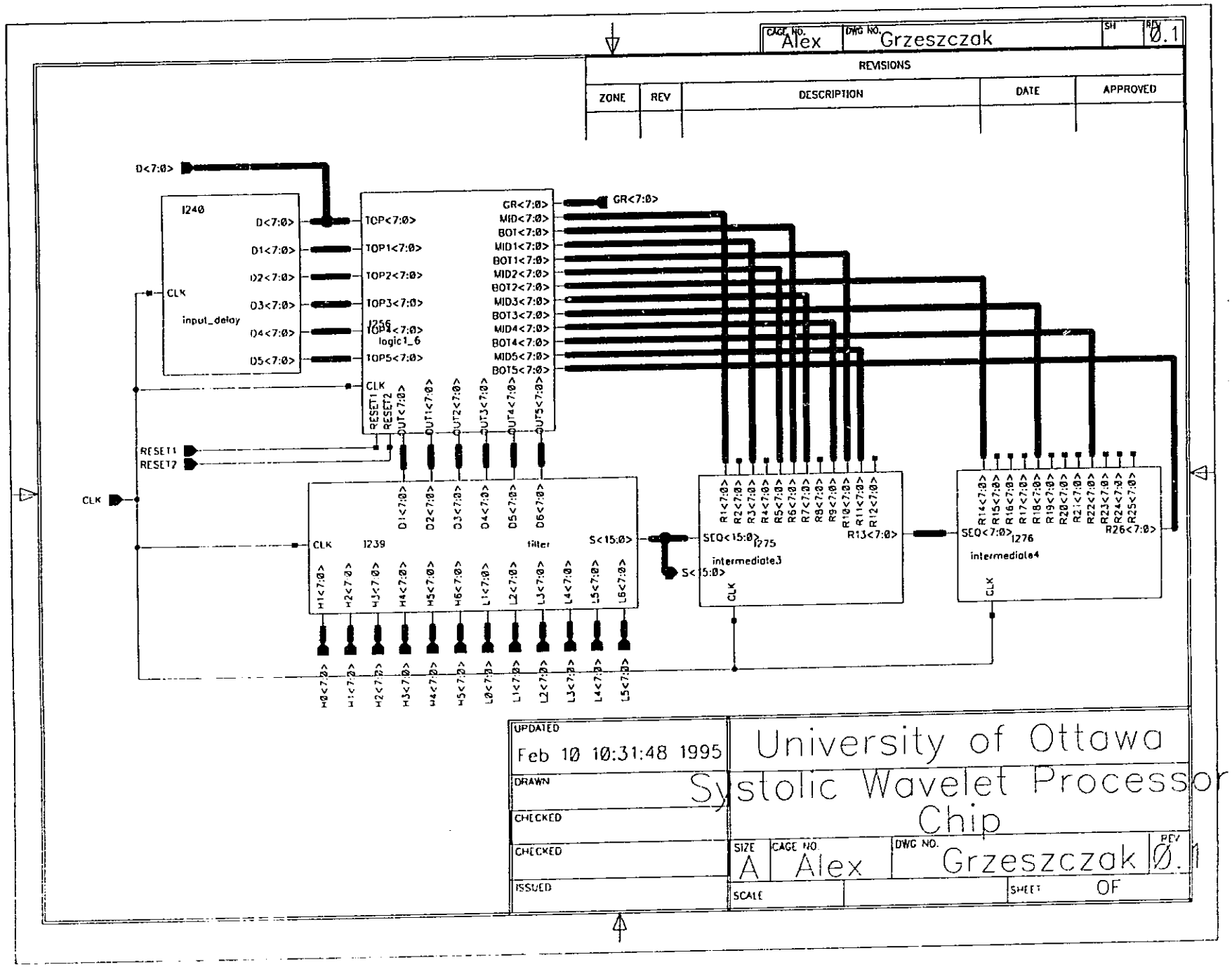
UPDATED	Mar 14 14:26:05 1995
DRAWN	
CHECKED	
CHECKED	
ISSUED	

University of Ottawa

Systolic Wavelet Processor Chip

SIZE	CAGE NO.	DWG NO.	REV
A	Alex	Grzeszczak	Ø.1

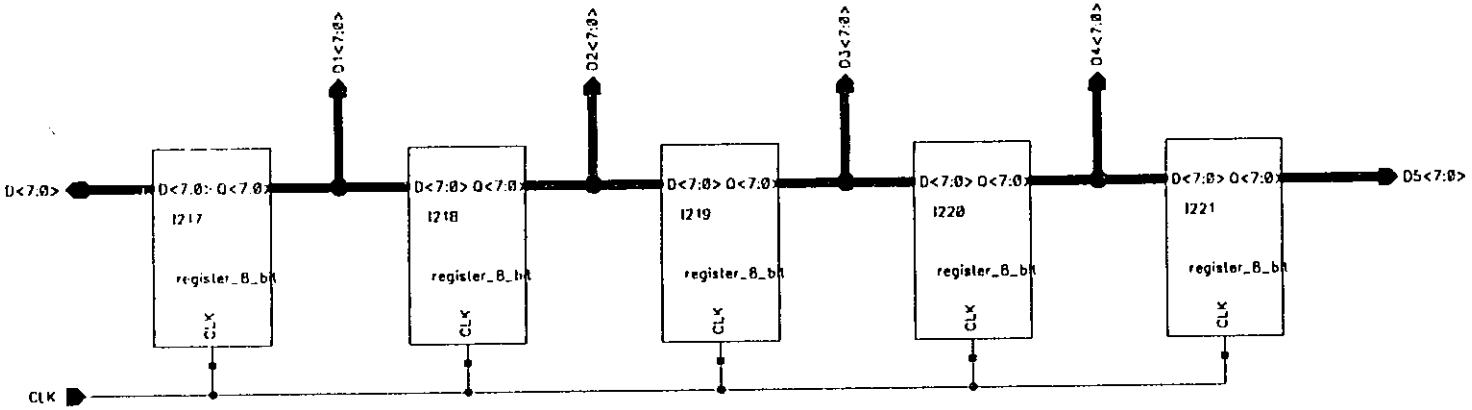
SCALE | SHEET OF


CAGE NO. Alex
DWG NO. Grzeszczak
SH
REV Ø.1
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
I240
input_delay
I256
logic1_6
I239
filter
I275
intermediate3
I276
intermediate4
UPDATED
Feb 10 10:31:48 1995
DRAWN
CHECKED
CHECKED
ISSUED
University of Ottawa
Systolic Wavelet Processor Chip
SIZE A
CAGE NO. Alex
DWG NO. Grzeszczak
REV Ø.1
SCALE
SHEET OF

CAGE NO.	DWG NO.	SH	REV
Alex	Grzeszczak		Ø.1

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED



UPDATED	Jan 18 10:37:48 1995
DRAWN	
CHECKED	
CHECKED	
ISSUED	

University of Ottawa

Systolic Wavelet Processor Input Delay

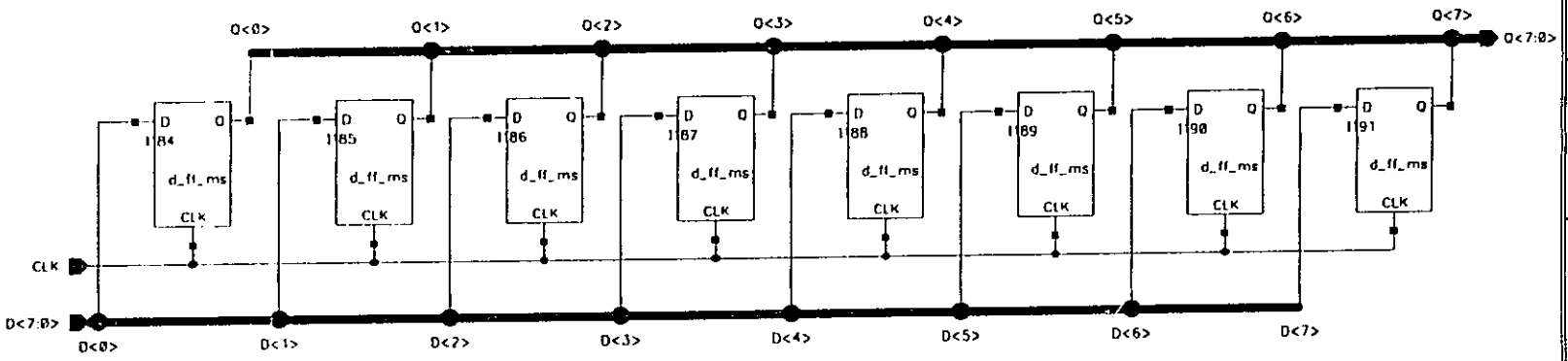
SIZE	CAGE NO.	DWG NO.	REV
A	Alex	Grzeszczak	Ø.1

SCALE | SHEET OF

CAGE NO.	DWG NO.	SH	REV
Alex	Grzeszczak		Ø.1

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED



UPDATED	Jan 18 11:29:16 1995
DRAWN	
CHECKED	
CHECKED	
ISSUED	

University of Ottawa

Systolic Wavelet Processor

Register (8 Bit)

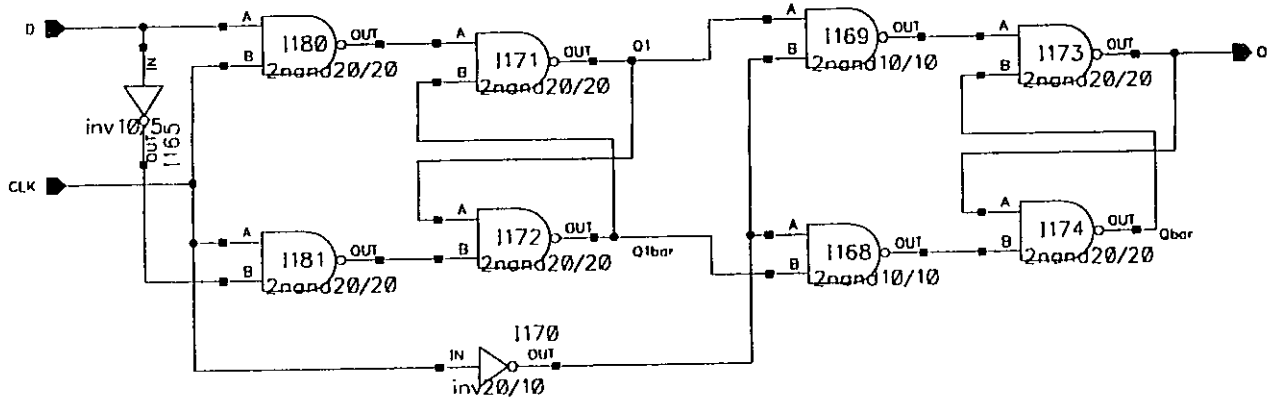
SIZE	CAGE NO.	DWG NO.	REV
A	Alex	Grzeszczak	Ø.1

SCALE | SHEET OF

CAGE NO.	DWG NO.	SH	REV
Alex	Grzeszczak		Ø.3

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED



UPDATED	Jan 18 10:27:19 1995
DRAWN	
CHECKED	
CHECKED	
ISSUED	

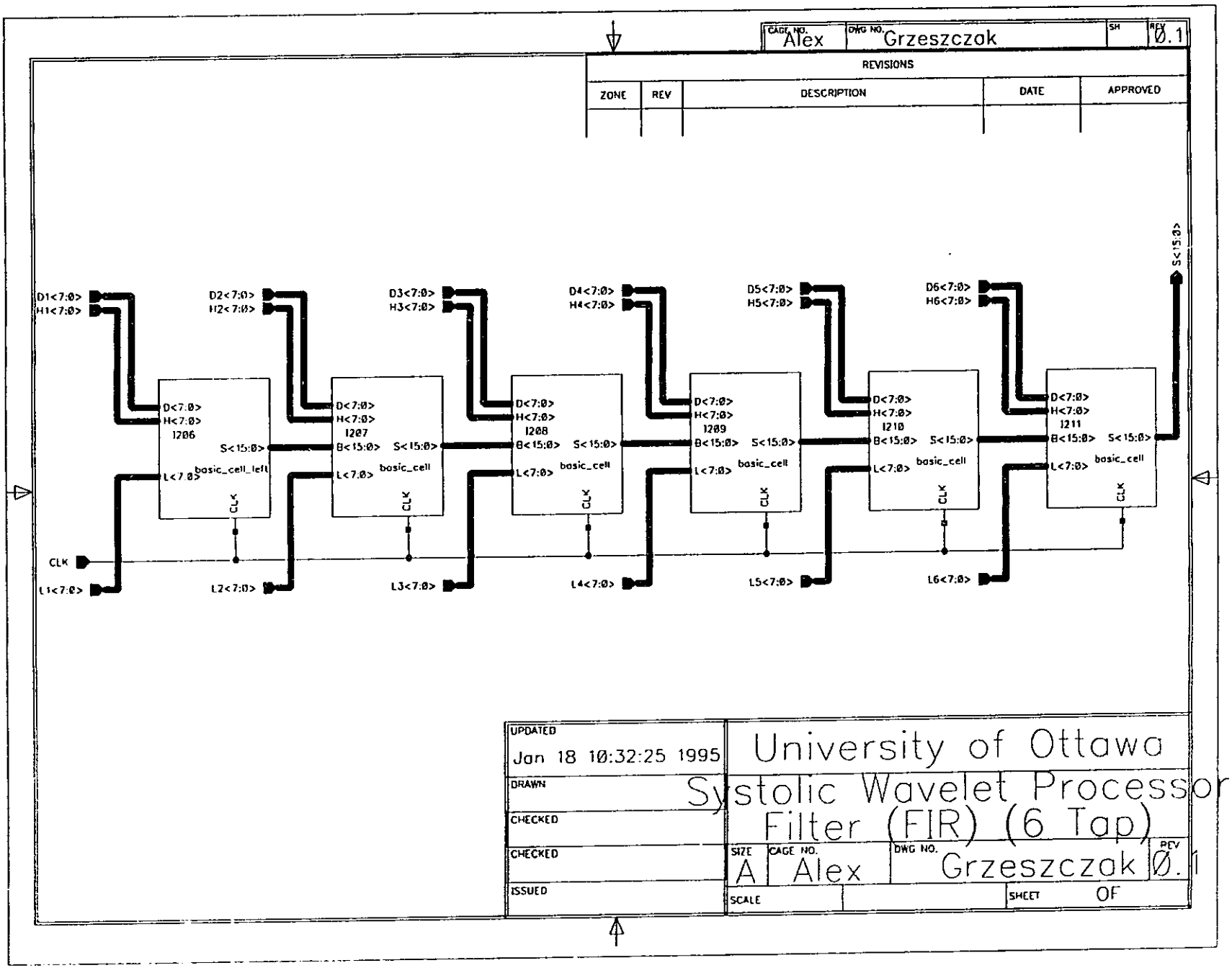
University of Ottawa

Systolic Wavelet Processor

D Flip Flop (Master Slave)

SIZE	CAGE NO.	DWG NO.	REV
A	Alex	Grzeszczak	Ø.3

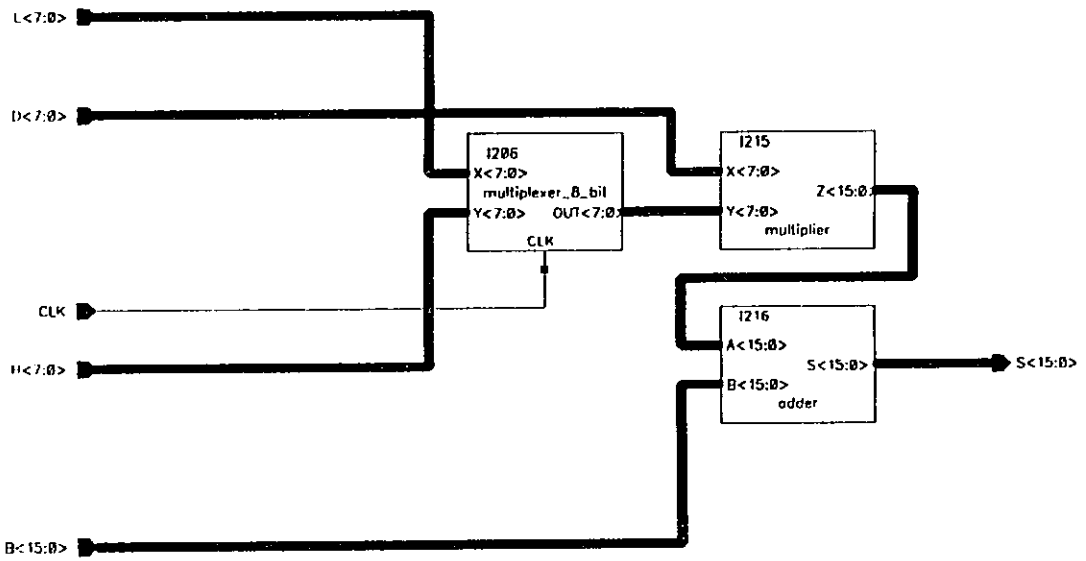
SCALE | SHEET OF


CAGE NO. Alex
DWG NO. Grzeszczak
SH
REV Ø.1
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
D1<7:0>
H1<7:0>
D2<7:0>
H2<7:0>
D3<7:0>
H3<7:0>
D4<7:0>
H4<7:0>
D5<7:0>
H5<7:0>
D6<7:0>
H6<7:0>
S<15:0>
D<7:0>
H<7:0>
I206
B<15:0>
S<15:0>
L<7:0>
basic_cell_left
CLK
I207
basic_cell
I208
I209
I210
I211
CLK
L1<7:0>
L2<7:0>
L3<7:0>
L4<7:0>
L5<7:0>
L6<7:0>
UPDATED
Jan 18 10:32:25 1995
DRAWN
CHECKED
CHECKED
ISSUED
University of Ottawa
Systolic Wavelet Processor
Filter (FIR) (6 Tap)
SIZE A
CAGE NO. Alex
DWG NO. Grzeszczak
REV Ø.1
SCALE
SHEET OF

CAGE NO. Alex	DWG NO. Grzeszczak	SH	REV Ø.1

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED

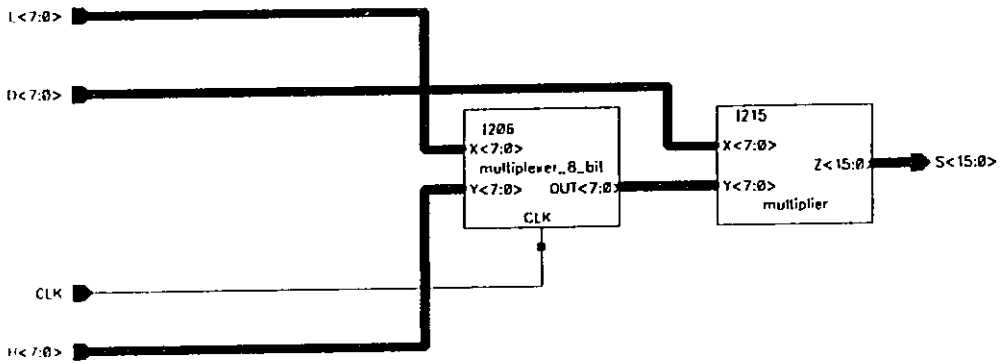


UPDATED Jan 20 11:53:46 1995	University of Ottawa
DRAWN	Systolic Wavelet Processor
CHECKED	Basic Filter Cell
CHECKED	SIZE A | CAGE NO. Alex | DWG NO. Grzeszczak | REV Ø.1
ISSUED	SCALE | SHEET OF

CAGE NO.	DWG NO.	SH	REV
Alex	Grzeszczak		Ø.1

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED

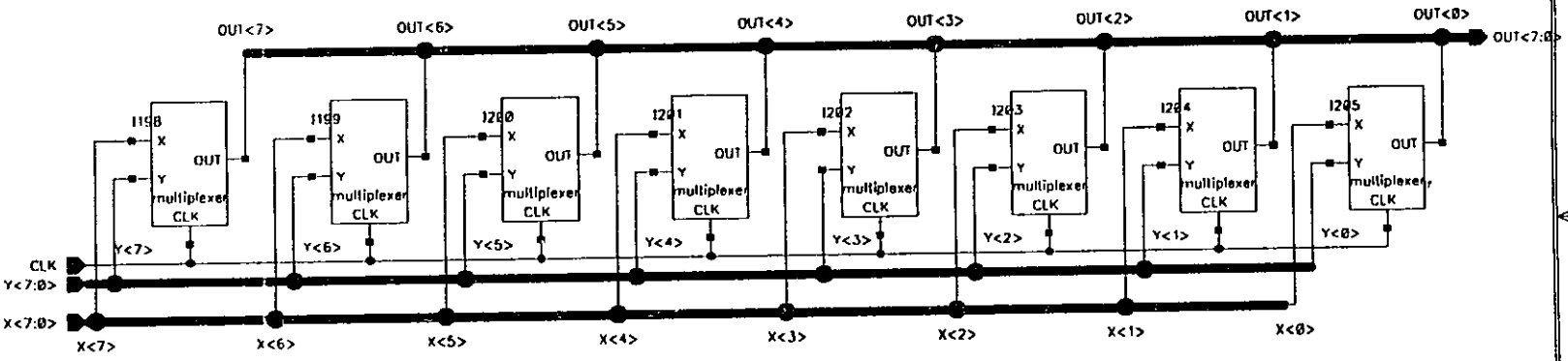


UPDATED	Jan 20 11:52:48 1995	University of Ottawa	
DRAWN		Systolic Wavelet Processor	
CHECKED		Basic Filter Cell (Left)	
CHECKED		SIZE A / CAGE NO. Alex / DWG NO. Grzeszczak / REV Ø.1	
ISSUED		SCALE / SHEET OF	

CAGE NO.	DWG NO.	SH	REV
Alex	Grzeszczak		Ø.1

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED



UPDATED	Jan 18 11:25:47 1995
DRAWN	
CHECKED	
CHECKED	
ISSUED	

University of Ottawa

Systolic Wavelet Processor

Multiplexer (8 Bit)

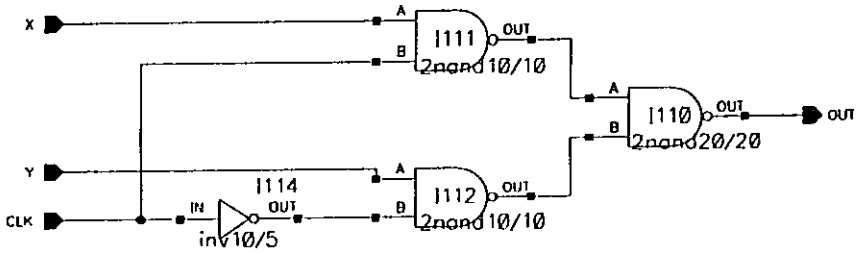
SIZE	CAGE NO.	DWG NO.	REV
A	Alex	Grzeszczak	Ø.1

SCALE | SHEET OF

CAGE NO.	DWG NO.	SH	REV
Alex	Grzeszczak		Ø.1

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED



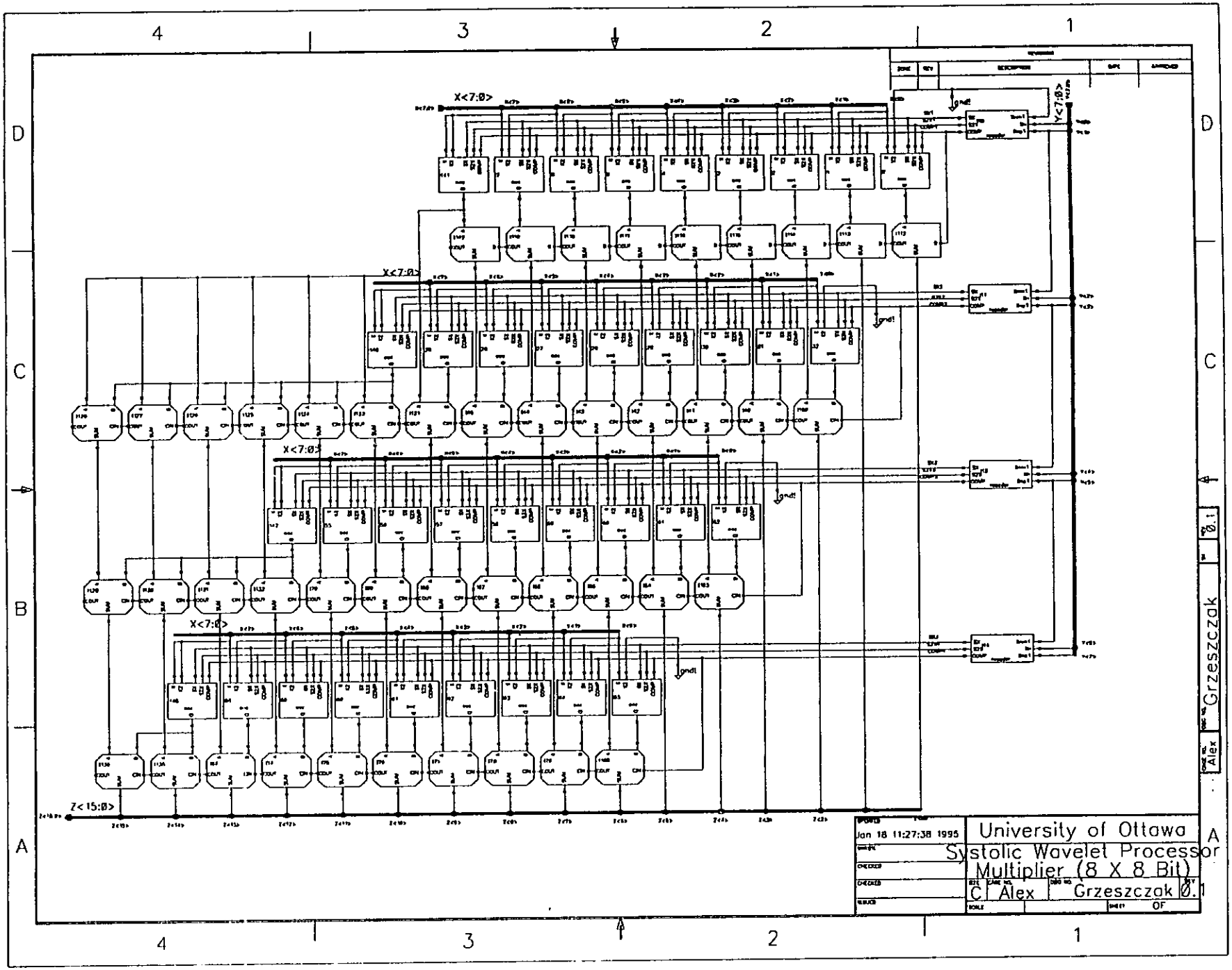
UPDATED	Jan 18 11:24:36 1995
DRAWN	
CHECKED	
CHECKED	
ISSUED	

University of Ottawa

Systolic Wavelet Processor Multiplexer

SIZE	CAGE NO.	DWG NO.	REV
A	Alex	Grzeszczak	Ø.1

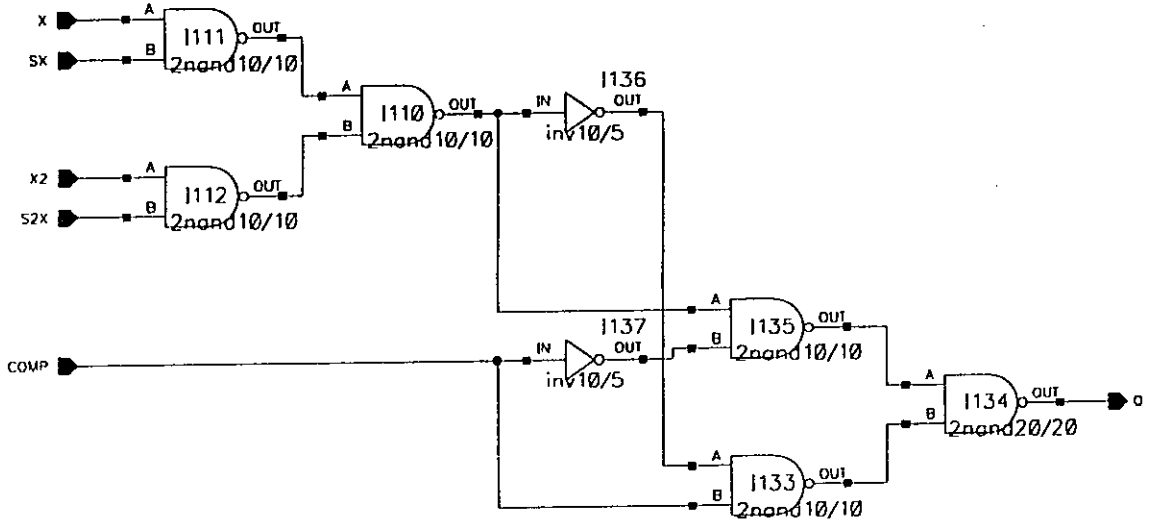
SCALE | SHEET OF


X<7:0>
X<7:0>
X<7:0>
X<7:0>
Z<15:0>
gnd!
University of Ottawa
Systolic Wavelet Processor
Multiplier (8 X 8 Bit)
Jan 18 11:27:38 1995
Alex
Grzeszczak Ø.1

CAGE NO.	DWG NO.	SH	REV
Alex	Grzeszczak		Ø.3

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED

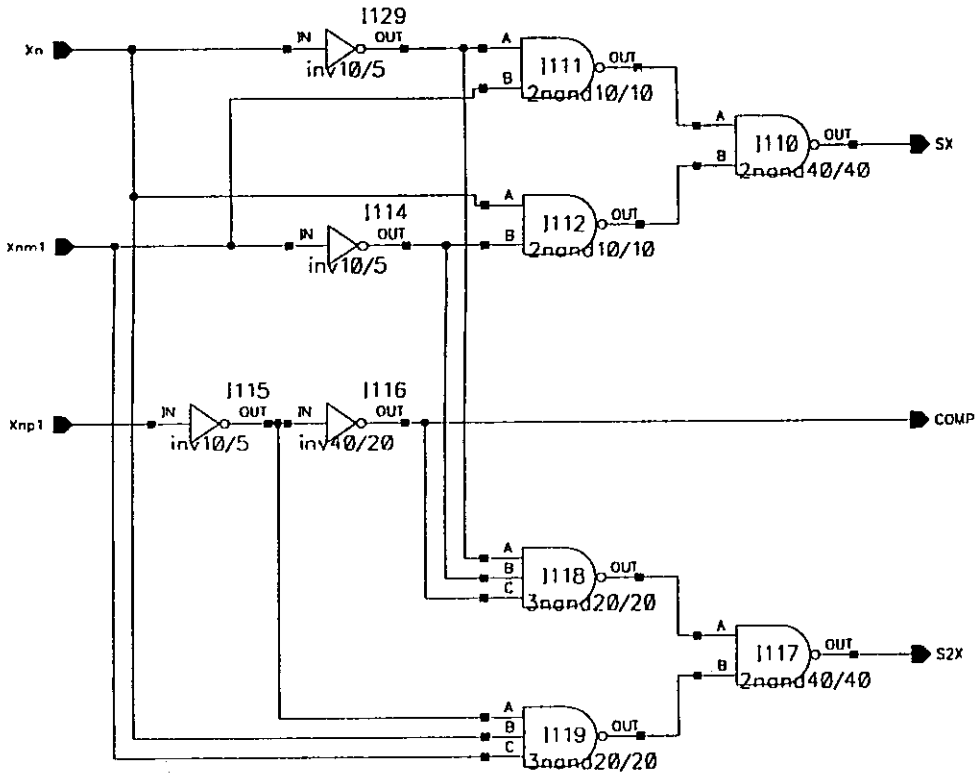


UPDATED	Jan 18 11:38:54 1995	University of Ottawa	
DRAWN		Systolic Wavelet Processor	
CHECKED		Shift And Complement	
CHECKED		SIZE A / CAGE NO. Alex / DWG NO. Grzeszczak	REV Ø.3
ISSUED		SCALE	SHEET OF

CAGE NO.	DWG NO.	SH	RV
Alex	Grzeszczak		Ø.1

REVISIONS

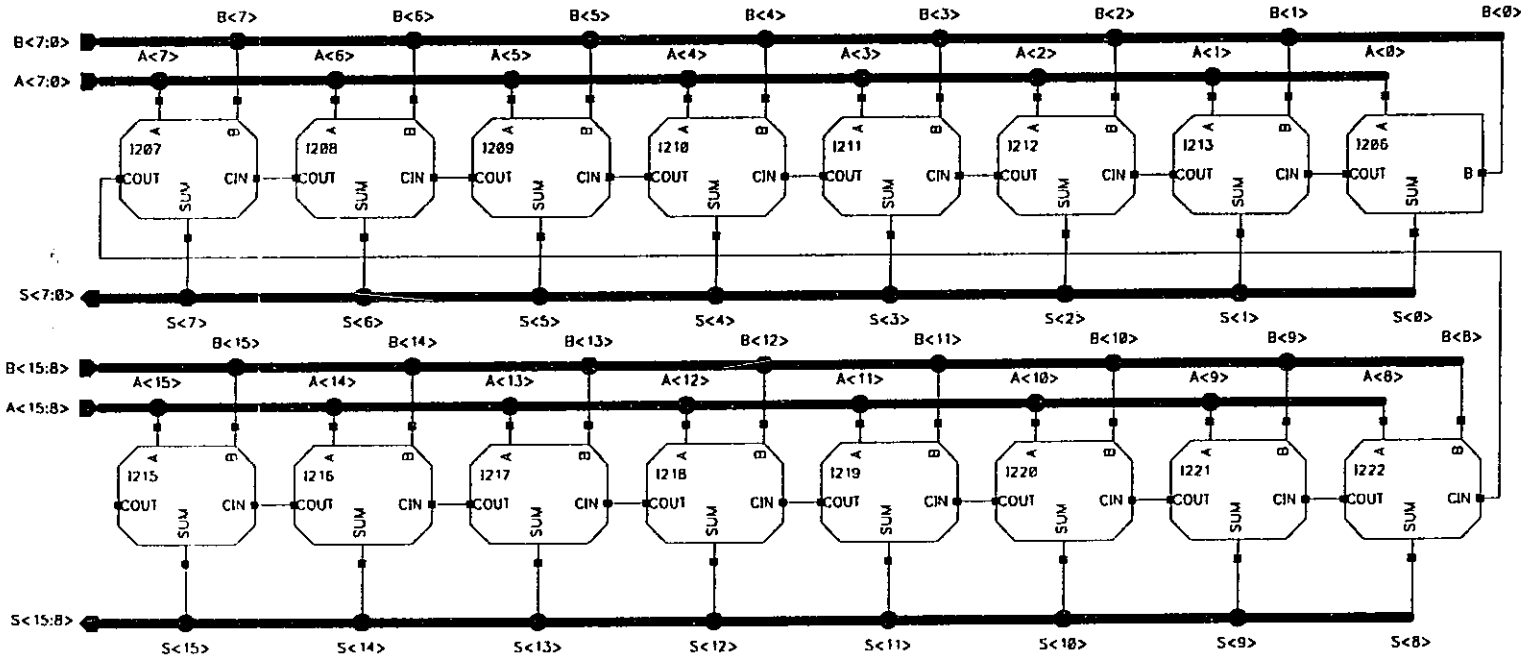
ZONE	REV	DESCRIPTION	DATE	APPROVED



UPDATED	Jan 18 11:28:29 1995	University of Ottawa		
DRAWN		Systolic Wavelet Processor		
CHECKED		Booth Recoder Cell		
CHECKED		SIZE A	CAGE NO. Alex	DWG NO. Grzeszczak
ISSUED		SCALE		SHEET OF

CAGE NO.	DWG NO.	SH	REV
Alex	Grzeszczak		Ø.1

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPROVED

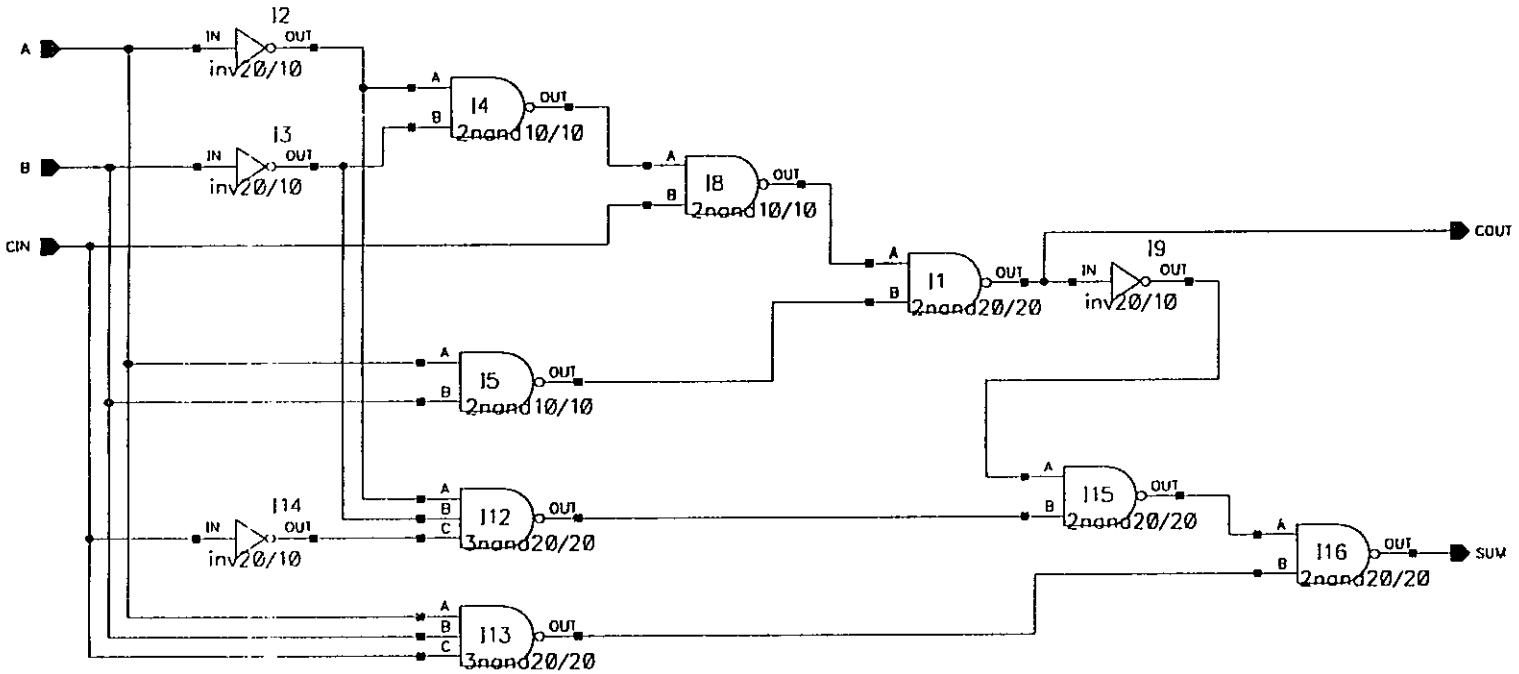


UPDATED	Jan 18 10:05:09 1995	University of Ottawa	
DRAWN		Systolic Wavelet Processor Adder	
CHECKED			
CHECKED		SIZE A / CAGE NO. Alex / DWG NO. Grzeszczak / REV Ø.1	
ISSUED		SCALE / SHEET OF	

CAGE NO.	DWG NO.	SH	REV
Alex	Grzeszczak		Ø.1

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED



UPDATED	Jan 18 10:33:24 1995
DRAWN	
CHECKED	
CHECKED	
ISSUED	

University of Ottawa

Systolic Wavelet Processor

Full Adder

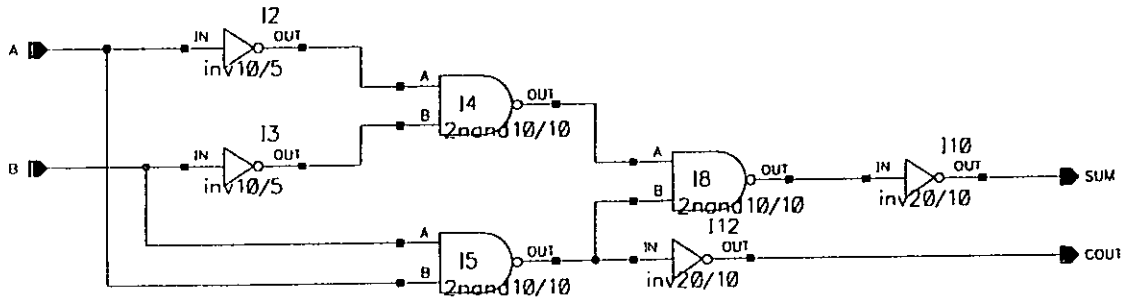
SIZE	CAGE NO.	DWG NO.	REV
A	Alex	Grzeszczak	Ø.1

SCALE		SHEET	OF

CAGE NO.	DWG NO.	SH	REV
Alex	Grzeszczak		Ø.1

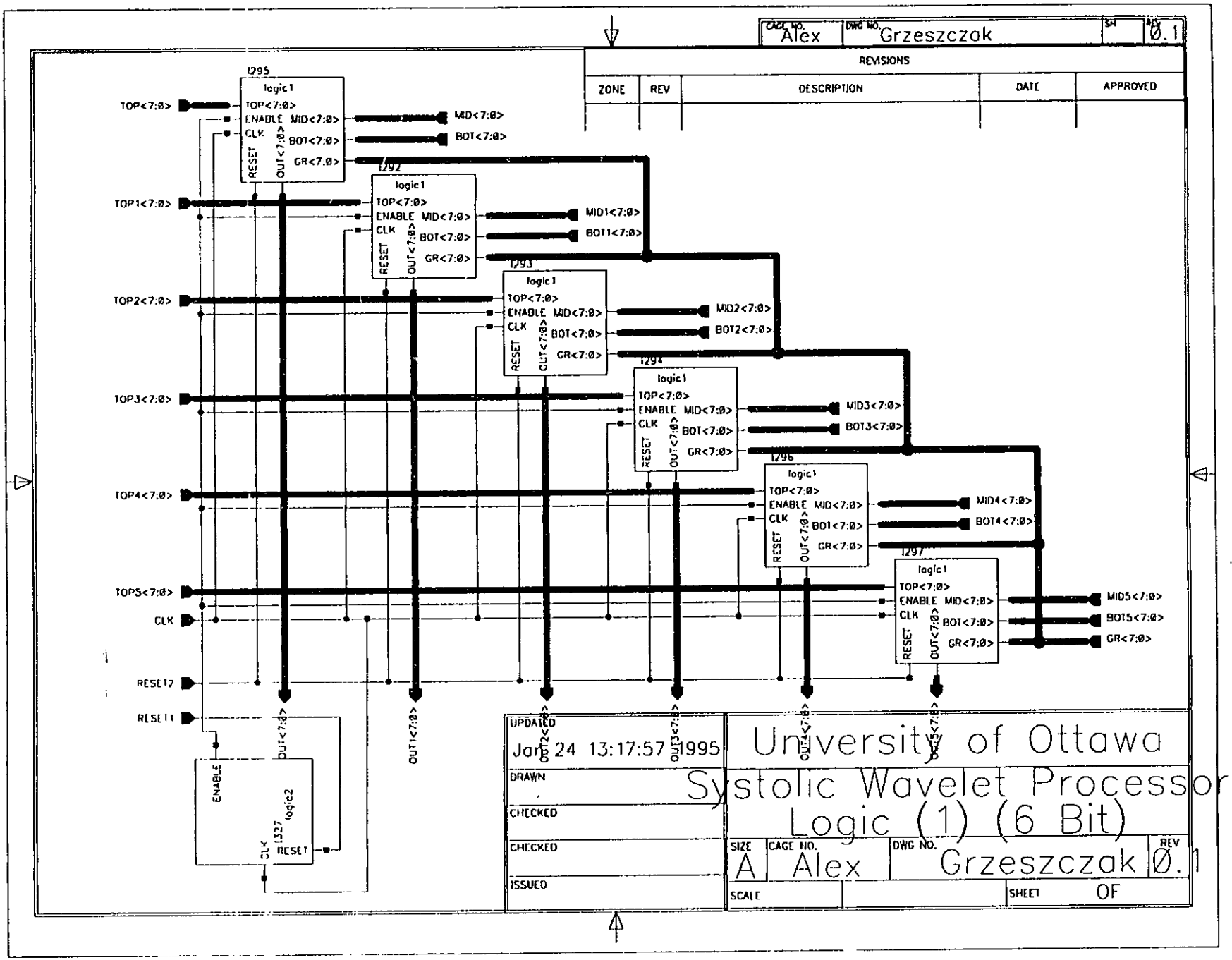
REVISIONS

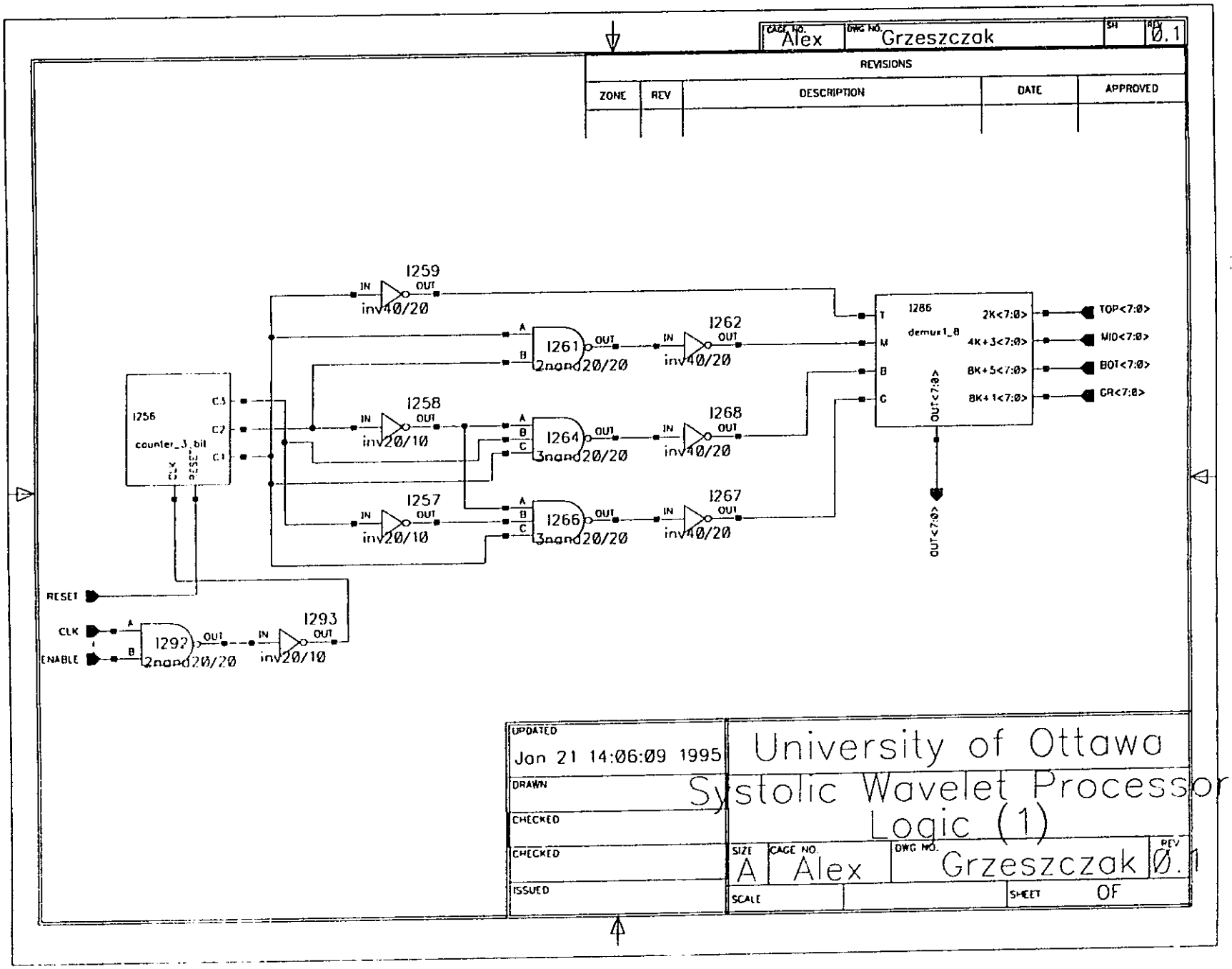
ZONE	REV	DESCRIPTION	DATE	APPROVED



UPDATED	Jan 18 10:34:08 1995	University of Ottawa	
DRAWN	Systolic Wavelet Processor		
CHECKED	Half Adder		
CHECKED			
ISSUED			

SIZE	CAGE NO.	DWG NO.	REV
A	Alex	Grzeszczak	Ø.1
SCALE		SHEET	OF

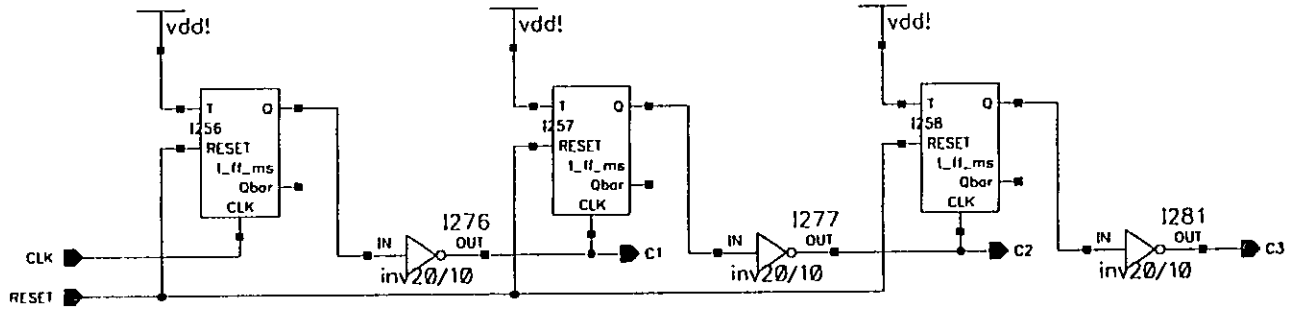

Alex
Grzeszczak
Ø.1
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
logic1
TOP<7:0>
ENABLE
CLK
RESET
OUT<7:0>
MID<7:0>
BOT<7:0>
GR<7:0>
MID1<7:0>
BOT1<7:0>
MID2<7:0>
BOT2<7:0>
MID3<7:0>
BOT3<7:0>
MID4<7:0>
BOT4<7:0>
MID5<7:0>
BOT5<7:0>
GR<7:0>
TOP1<7:0>
TOP2<7:0>
TOP3<7:0>
TOP4<7:0>
TOP5<7:0>
CLK
RESET2
RESET1
logic2
UPDATED
Jan 24 13:17:57 1995
DRAWN
CHECKED
CHECKED
ISSUED
University of Ottawa
Systolic Wavelet Processor
Logic (1) (6 Bit)
SIZE
A
CAGE NO.
Alex
DWG NO.
Grzeszczak
REV
Ø.1
SCALE
SHEET
OF


CAGE NO. Alex
DWG NO. Grzeszczak
SH
REV Ø.1
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
1259
IN
OUT
inv40/20
1261
A
B
2nand20/20
1262
inv40/20
1256
counter_3_bit
C3
C2
C1
CLK
RESET
1258
inv20/10
1264
A
B
C
3nand20/20
1268
inv40/20
1257
inv20/10
1266
3nand20/20
1267
inv40/20
1286
demux1_8
T
M
B
C
OUT<7:0>
2K<7:0>
4K+3<7:0>
8K+5<7:0>
8K+1<7:0>
TOP<7:0>
MID<7:0>
BOT<7:0>
GR<7:0>
OUT<7:0>
RESET
CLK
ENABLE
1292
2nand20/20
1293
inv20/10
UPDATED
Jan 21 14:06:09 1995
DRAWN
CHECKED
CHECKED
ISSUED
University of Ottawa
Systolic Wavelet Processor
Logic (1)
SIZE A
CAGE NO. Alex
DWG NO. Grzeszczak
REV Ø.1
SCALE
SHEET OF

CAGE NO.	DWG NO.	SH	REV
Alex	Grzeszczak		Ø.1

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED



UPDATED	Jan 18 10:10:00 1995
DRAWN	
CHECKED	
CHECKED	
ISSUED	

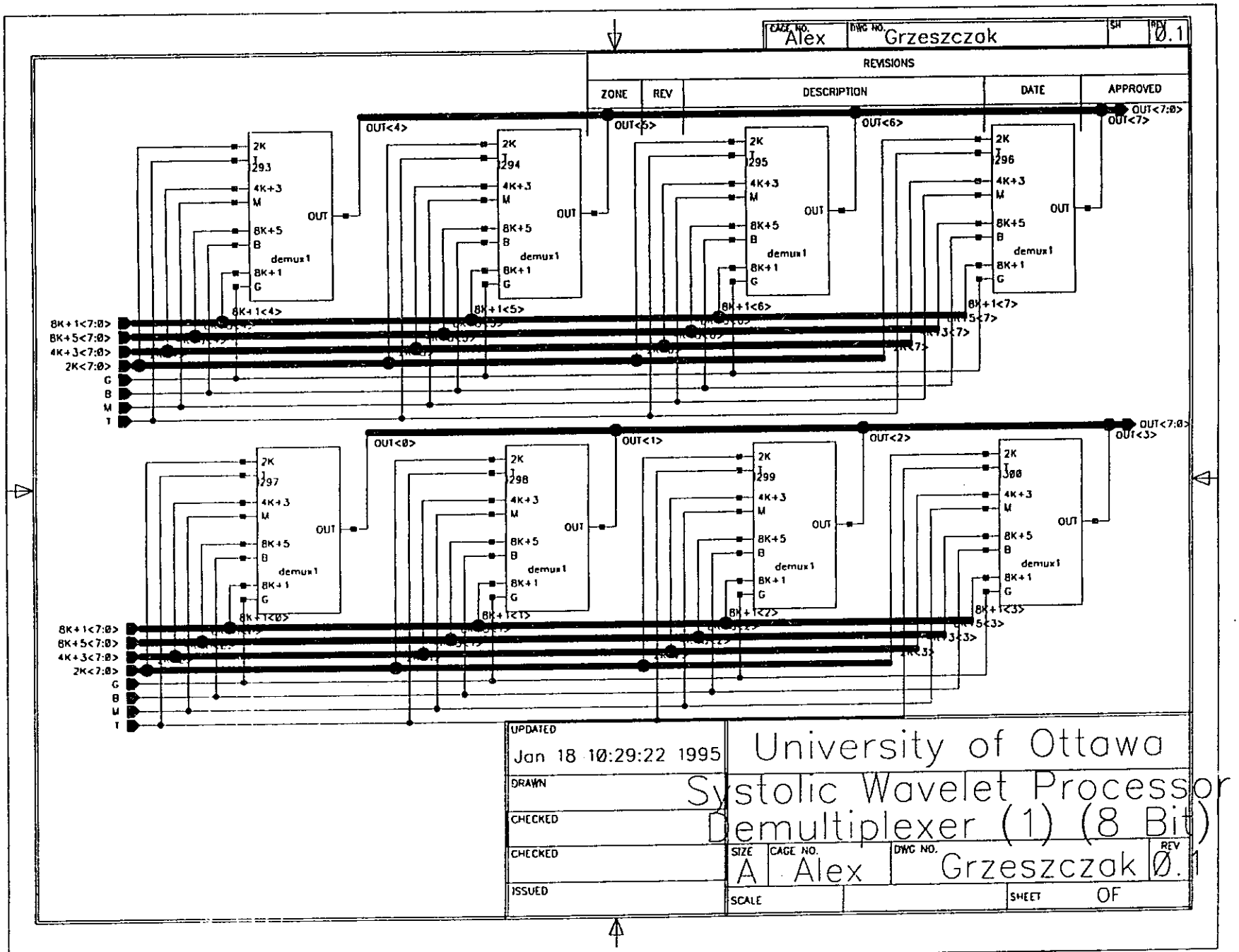
University of Ottawa

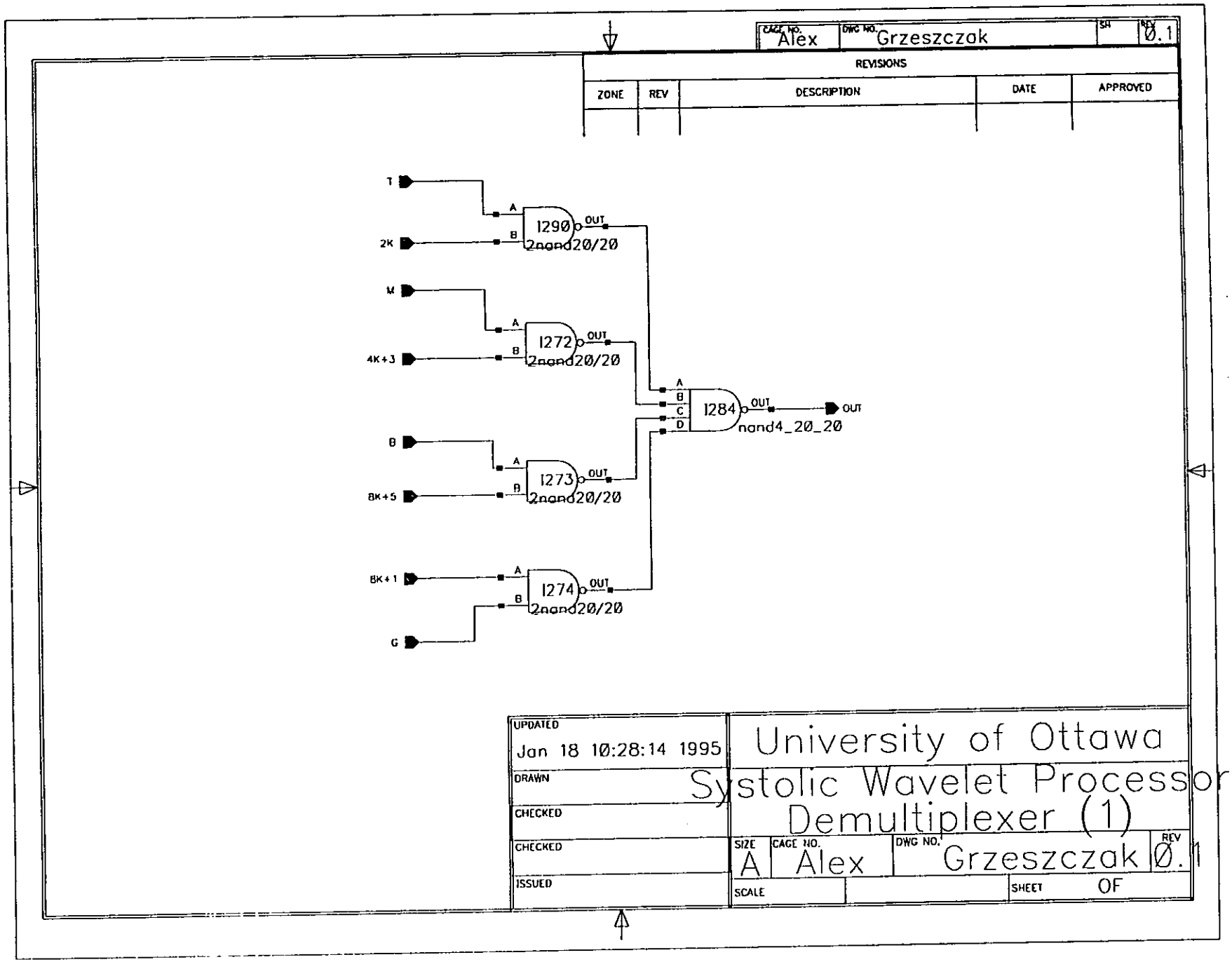
Systolic Wavelet Processor

Counter (3 Bit)

SIZE	CAGE NO.	DWG NO.	REV
A	Alex	Grzeszczak	Ø.1

SCALE		SHEET	OF

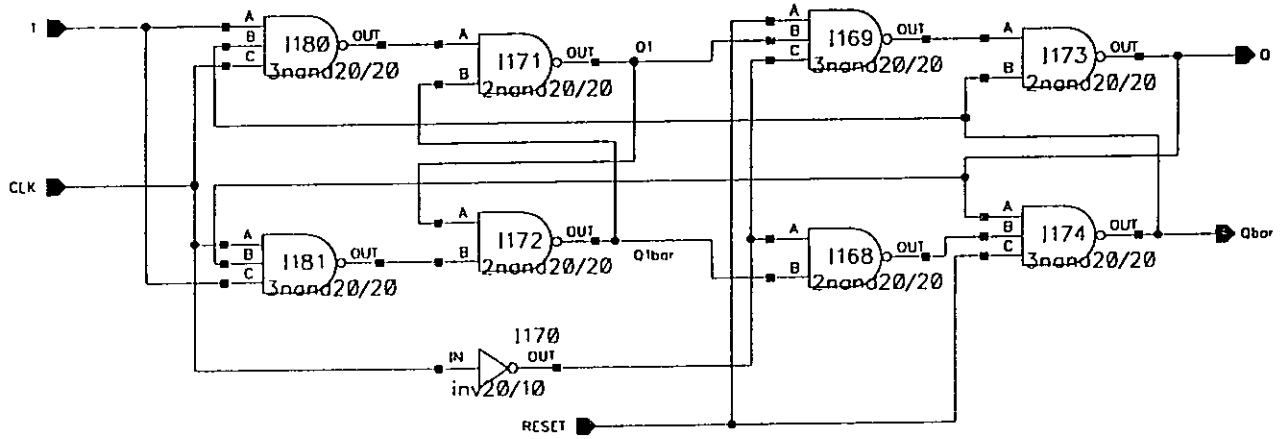

CAGE NO. Alex
DWG NO. Grzeszczak
SH
REV Ø.1
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
OUT<4>
OUT<5>
OUT<6>
OUT<7:0>
OUT<7>
2K
I
293
294
295
296
4K+3
M
8K+5
B
8K+1
G
OUT
demux1
8K+1<4>
8K+1<5>
8K+1<6>
8K+1<7>
8K+1<7:0>
8K+5<7:0>
4K+3<7:0>
2K<7:0>
G
B
M
T
OUT<0>
OUT<1>
OUT<2>
OUT<3>
297
298
299
300
8K+1<0>
8K+1<1>
8K+1<2>
8K+1<3>
8K+1<7:0>
8K+5<7:0>
4K+3<7:0>
2K<7:0>
UPDATED
Jan 18 10:29:22 1995
DRAWN
CHECKED
CHECKED
ISSUED
University of Ottawa
Systolic Wavelet Processor
Demultiplexer (1) (8 Bit)
SIZE A
CAGE NO. Alex
DWG NO. Grzeszczak
REV Ø.1
SCALE
SHEET OF


CAGE NO. Alex
DWG NO. Grzeszczak
SH
REV Ø.1
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
T
2K
M
4K+3
B
8K+5
8K+1
G
A
B
OUT
1290
2nand20/20
1272
2nand20/20
1273
2nand20/20
1274
2nand20/20
C
D
1284
nand4_20_20
OUT
UPDATED
Jan 18 10:28:14 1995
DRAWN
CHECKED
CHECKED
ISSUED
University of Ottawa
Systolic Wavelet Processor
Demultiplexer (1)
SIZE A
CAGE NO. Alex
DWG NO. Grzeszczak
REV Ø.1
SCALE
SHEET OF

CAGE NO.	DWG NO.	SH	REV
Alex	Grzeszczak		Ø.3

REVISIONS

ZONE	REV	DESCRIPTION	DATE	APPROVED



UPDATED	Jan 18 11:39:48 1995
DRAWN	
CHECKED	
CHECKED	
ISSUED	

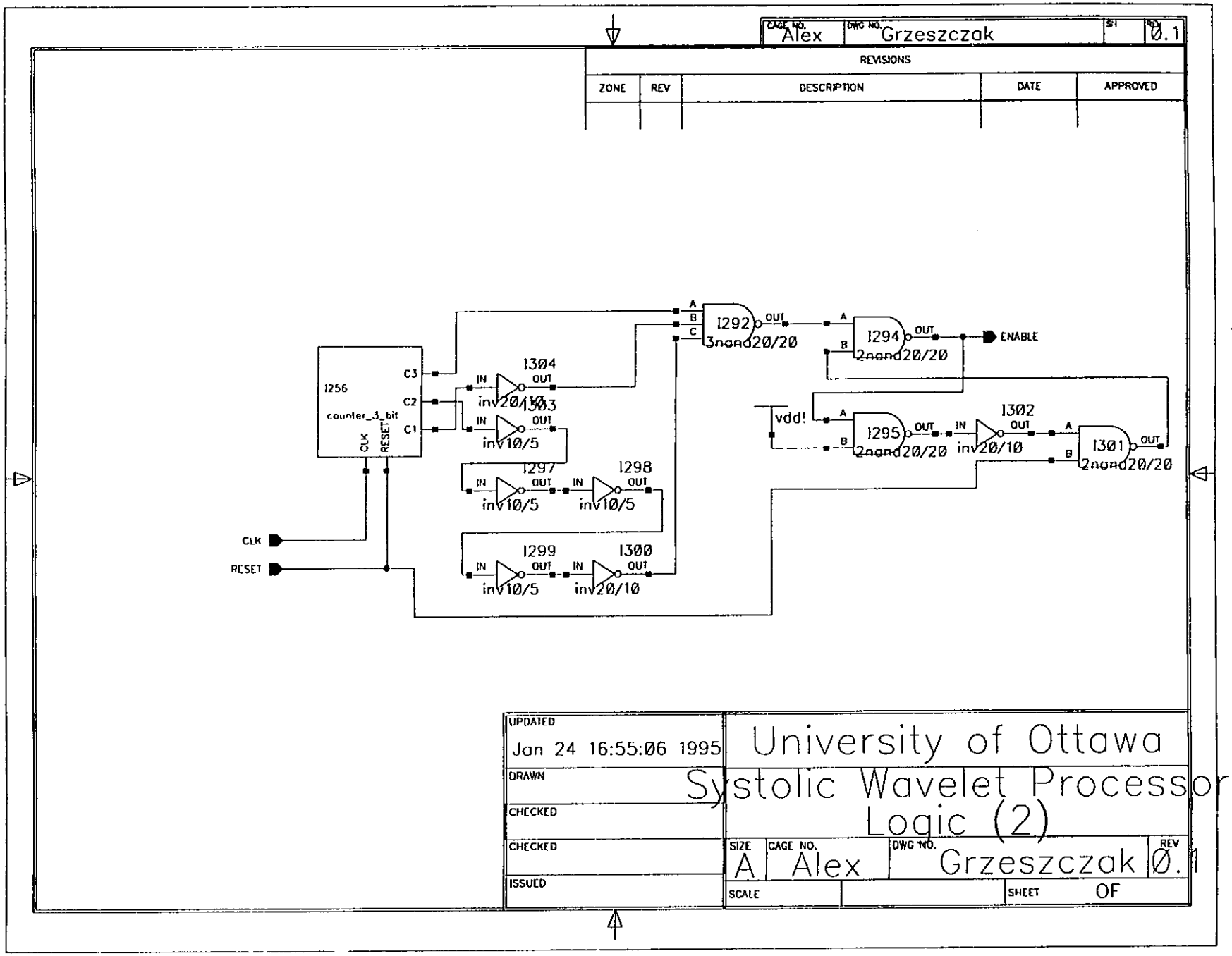
University of Ottawa

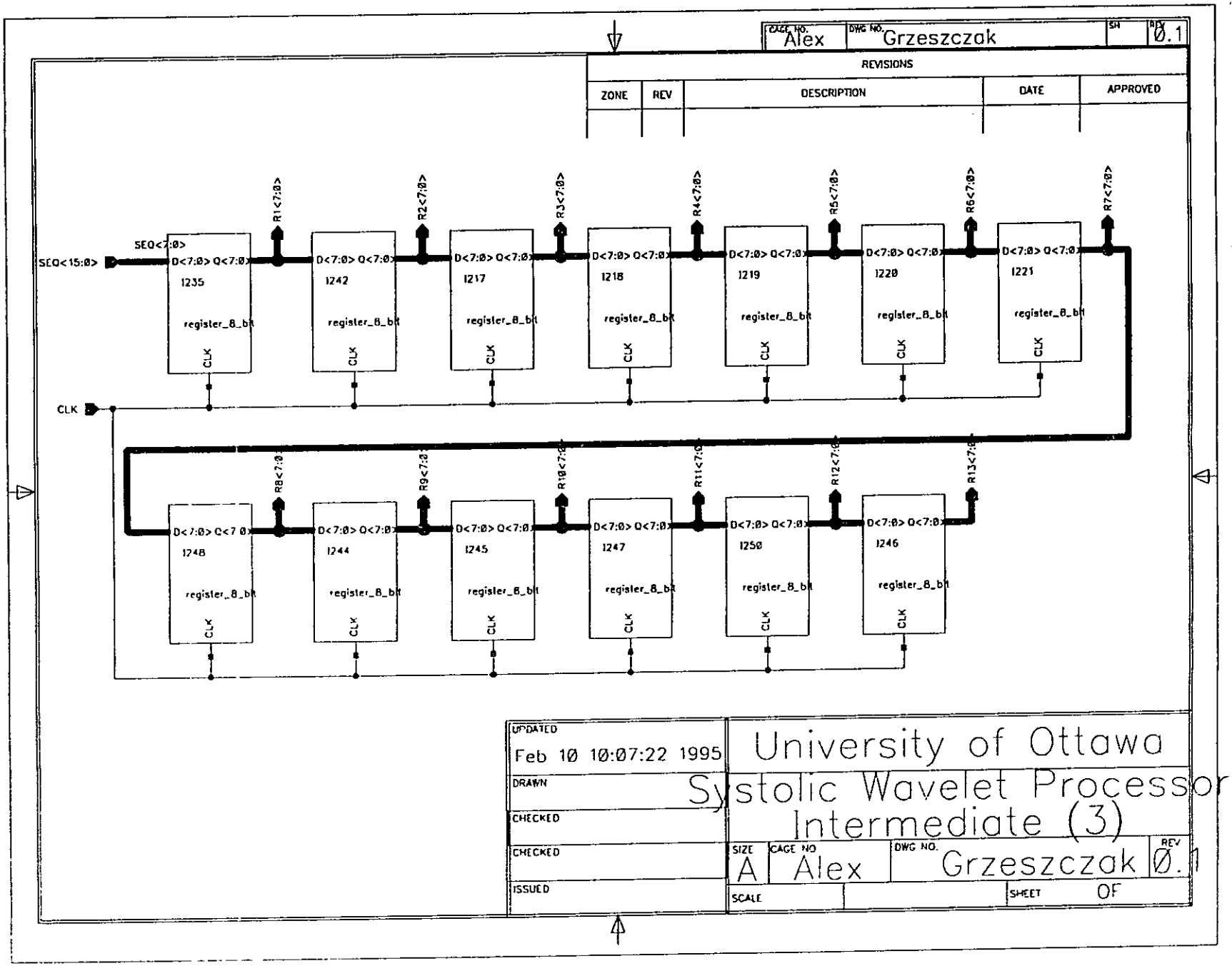
Systolic Wavelet Processor

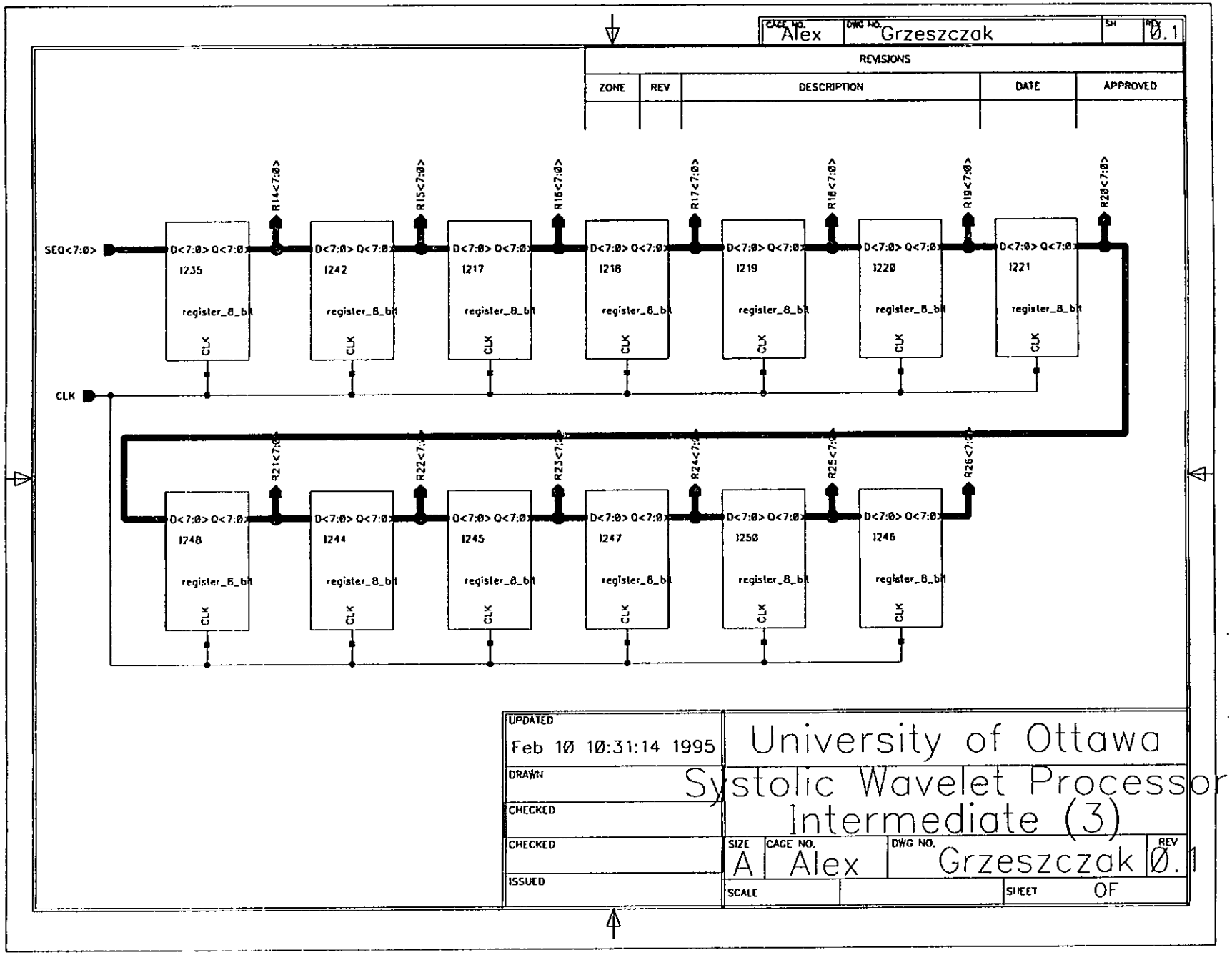
T Flip Flop (Master Slave)

SIZE	CAGE NO.	DWG NO.	REV
A	Alex	Grzeszczak	Ø.3

SCALE | SHEET OF


CAGE NO. Alex
DWG NO. Grzeszczak
SIZE
REV Ø.1
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
1256
counter_3_bit
CLK
RESET
C3
C2
C1
CLK
RESET
1304
inv20/10
1303
inv10/5
1297
inv10/5
1298
inv10/5
1299
inv10/5
1300
inv20/10
1292
3nand20/20
1294
2nand20/20
ENABLE
vdd!
1295
2nand20/20
1302
inv20/10
1301
2nand20/20
UPDATED
Jan 24 16:55:06 1995
DRAWN
CHECKED
CHECKED
ISSUED
University of Ottawa
Systolic Wavelet Processor
Logic (2)
SIZE A
CAGE NO. Alex
DWG NO. Grzeszczak
REV Ø.1
SCALE
SHEET OF


CAGE NO. Alex
DWG NO. Grzeszczak
SH
REV Ø.1
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
SEQ<15:0>
SEQ<7:0>
CLK
R1<7:0>
R2<7:0>
R3<7:0>
R4<7:0>
R5<7:0>
R6<7:0>
R7<7:0>
R8<7:0>
R9<7:0>
R10<7:0>
R11<7:0>
R12<7:0>
R13<7:0>
D<7:0> Q<7:0>
1235
1242
1217
1218
1219
1220
1221
1248
1244
1245
1247
1250
1246
register_8_bit
CLK
UPDATED
Feb 10 10:07:22 1995
DRAWN
CHECKED
CHECKED
ISSUED
University of Ottawa
Systolic Wavelet Processor
Intermediate (3)
SIZE A
CAGE NO Alex
DWG NO. Grzeszczak
REV Ø.1
SCALE
SHEET OF


CAGE NO. Alex
DWG NO. Grzeszczak
SH
REV Ø.1
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
SEQ<7:0>
CLK
D<7:0> Q<7:0>
1235
1242
1217
1218
1219
1220
1221
1248
1244
1245
1247
1250
1246
register_8_bit
CLK
R14<7:0>
R15<7:0>
R16<7:0>
R17<7:0>
R18<7:0>
R19<7:0>
R20<7:0>
R21<7:0>
R22<7:0>
R23<7:0>
R24<7:0>
R25<7:0>
R26<7:0>
UPDATED
Feb 10 10:31:14 1995
DRAWN
CHECKED
CHECKED
ISSUED
University of Ottawa
Systolic Wavelet Processor
Intermediate (3)
SIZE A
CAGE NO. Alex
DWG NO. Grzeszczak
REV Ø.1
SCALE
SHEET OF