

DE MONTFORT UNIVERSITY  
SCHOOL OF ENGINEERING AND MANUFACTURE



# FINAL YEAR PROJECT REPORT

---

**DIGIT-SERIAL IIR FILTER  
IMPLEMENTATION ON FPGA**

Author: Kyriakos Deliparaschos

Hand in: Dr Amar Aggoun

## **Acknowledgements**

I would like to thank my supervisor and project leader Dr A. Aggoun of Electronics and Electrical Engineering Department at De Montfort University, Gateway, Leicester, for supplying me with this project and for his kind help and guidance.

I would also like to acknowledge my appreciation of the efforts of Nick Bessis and everybody else who did not hesitate to answer my questions and queries.

Finally, the love and encouragement of my parents and my girlfriend is sincerely appreciated.

## Abstract

This project is based on the implementation of a **digit-serial IIR filter**, on FPGA by either using VHDL or ECAD programs (*Viewlogic*).

The application of the digit-serial structures to the design of IIR filters introduces delay elements in the feed back loop of the IIR filter. This offers the possibility of pipelining the feed back loop inherent in the IIR filters. The digit serial structure is based on the feed forward of the carry digit, which allows sub digit pipelining to increase the throughput rate of the IIR filters.

The implementation of the digital filter was split into its fundamental elements according to its block diagram. All the elements of the filter were designed, simulated and tested to prove their functionality. Furthermore a 1<sup>st</sup> order digit-serial IIR filter ( $n=4$ ,  $M=32$ ) was composed and simulated to prove that is functioning satisfactorily.

Finally the last should be downloaded onto the FPGA and tested. The FPGA chip, which was available at the time of this project, was located on a general use board intended for less complex designs. Due to this fact the 1<sup>st</sup> order digit-serial filter was not downloaded, but the 16x16 bit digit-serial multiplier with digit-serial adder and parallel-in to serial-out register was downloaded instead and tested.

# Contents

	<b>Page</b>
<b>Acknowledgements</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>List of Contents</b>	<b>iii</b>
<b>List of Figures</b>	<b>v</b>
<b>List of Tables</b>	<b>vii</b>
<b>Introduction</b>	<b>1</b>
➤ Brief Introduction to Digital Filters	1
➤ Scope of the project	6
<b>Chapter 1 Previous Work Done</b>	<b>7</b>
1.1 Introduction	7
1.2 Scattered Look-Ahead technique	7
1.3 Signed Digit number representation	8
<b>Chapter 2 Radix-2<sup>n</sup> Arithmetic</b>	<b>11</b>
2.1 Introduction	11
2.2 The Radix-2 <sup>n</sup> Approach	12
2.3 New Radix-2 <sup>n</sup> Vector Inner Product Algorithms	13
2.4 Radix-2 <sup>n</sup> Arithmetic Cell	14
<b>Chapter 3 Bit-level Pipelined Digit-Serial IIR Filter</b>	<b>16</b>
3.1 Introduction	16
3.2 Design Methodology of Digit-Serial IIR Filter	17
3.3 Pipelined Digit-Serial IIR Filter	22
3.4 Bit-level Pipelined Digit-Serial Multiplier	22
3.5 Truncation of the Output	24
3.6 Systolic Digit-Serial IIR Filter Structure	26
3.7 Accumulation of the Recursive and the Nonrecursive Computations	27
3.8 Pipelining of the Digit-Serial IIR Filter	29
3.9 Pipelining Of the Digit-Serial Adder	30
3.10 Pipelining of the Digit-Serial IIR Filter	30

3.11	Design of the Bit-level Pipelined Digit-Serial Adder	31
<b>Chapter 4</b>	<b>Implementation of 1<sup>st</sup> order Digit-Serial IIR Filter</b>	<b>34</b>
4.1	Introduction	34
4.2	Full Adder Design and Simulation	34
4.3	AND gated Full Adder Design	42
4.4	Carry Save Adder Design	43
4.5	4-bit Register Design	45
4.6	Digit-Serial Multiplier Design and Simulation	47
4.7	Carry Ripple Adder Design	52
4.8	16x16 bit Digit-Serial Multiplier with Digit-Serial Adder Design and Simulation	53
4.9	16x16 bit Digit-Serial Multiplier with Digit-Serial Adder Design with Shift Registers Design and Simulation	63
	4.9.1 Parallel-in to Serial-out Shift Register Design	63
	4.9.2 Serial-in to Parallel-out Shift Register Design	65
4.10	1 <sup>st</sup> order Digit-Serial IIR Filter Design and Simulation	69
	4.10.1 Carry Save Adders Array Design	70
	4.10.2 Bit-level Pipelined Digit-Serial Adder Design and Simulation	72
<b>Chapter 5</b>	<b>Implementation and Testing of 16x16 bit Digit-Serial Multiplier on FPGA</b>	<b>80</b>
5.1	Introduction	80
5.2	An Overview for <i>Xilinx</i> FPGAs	80
5.3	Programming or Configuring the Device	81
5.4	Downloading of 16x16 bit Digit-Serial Multiplier on FPGA and Testing	82
<b>Chapter 6</b>	<b>Conclusion</b>	<b>86</b>
<b>Chapter 7</b>	<b>Recommendations for Further Work</b>	<b>89</b>
<b>References</b>		<b>90</b>
<b>Appendix A</b>	<b>FPGA Board Schematic</b>	<b>93</b>
<b>Appendix B</b>	<b>Xilinx Family Architecture Comparison</b>	<b>95</b>
<b>Appendix C</b>	<b>Photographs of the Project</b>	<b>97</b>
<b>Appendix D</b>	<b>Gantt Chart</b>	<b>100</b>

- Figure 29:** Digit-Serial multiplier symbol
- Figure 30:** Digit-Serial multiplier simulation results
- Figure 31:** CRA schematic
- Figure 32:** CRA symbol
- Figure 33:** 16x16 bit Digit-Serial multiplier
- Figure 34:** Digit-Serial Adder block diagram
- Figure 35:** 16x16 bit Digit-Serial multiplier and Digit-Serial Adder
- Figure 36:** Simulation result for the 16x16 bit Digit-Serial multiplier
- Figure 37:** 16x16 bit digit-serial multiplier with digit-serial adder symbol
- Figure 38:** Parallel-in to Serial-out shift register block diagram
- Figure 39:** Serial-in to Parallel-out shift register block diagram
- Figure 40:** 16x16 bit Digit-Serial multiplier with shift registers
- Figure 41:** Simulation results for 16x16 bit digit-serial multiplier with shift registers
- Figure 42:** Block diagram of a 1<sup>st</sup> order digit-serial IIR filter (M=4)
- Figure 43:** Block diagram of CSAs array
- Figure 44:** Schematic of CSAs array
- Figure 45:** Symbol for CSAs array
- Figure 46:** Bit-level pipelined digit-serial adder
- Figure 47:** Bit-level pipelined digit-serial adder symbol
- Figure 48:** Simulation results bit-level pipelined digit-serial adder
- Figure 49:** Schematic of 1<sup>st</sup> order digit-serial IIR filter
- Figure 50:** 1<sup>st</sup> order digit-serial IIR filter simulation results
- Figure 51:** 16x16 bit digit-serial multiplier with parallel-in/serial out shift register (FPGA version)

## Introduction

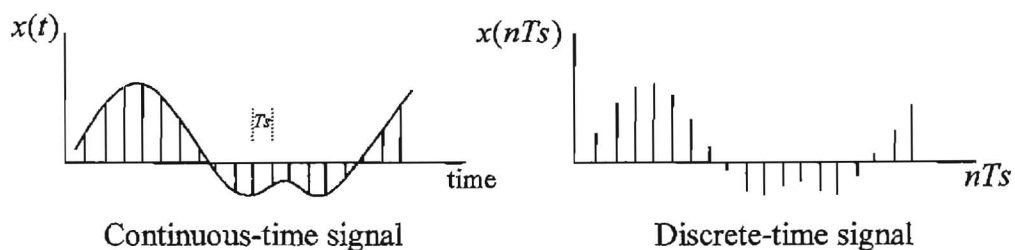
### ➤ Brief introduction to digital filters

Signals exist in almost every field of science such as acoustics, biomedical engineering, communications, control systems, radar, physics, seismology and telemetry.

There are two general types of signals, namely *continuous-time* signals and *discrete-time* signals. A continuous-time signal is one that defined at every instant of time. At the other hand discrete-time signal is one that that defined at discrete instants of time.

Fig. 1 shows the two types of signals. If the continuous-time signal is denoted by  $x(t)$ , then the discrete-time sequence is denoted as

$$x(nTs) = x(t) \quad \text{where, } t = nTs$$



**Figure 1:** Types of signals

A discrete-time signal can be represented by the *frequency spectrum* of the signal. The frequency spectrum describes the frequency contents of the signal.

The process, by which the frequency spectrum of a signal can be modified, reshaped or generally manipulated according to the required specifications, is called *filtering*. In the filtering process the frequency components of the signal can be attenuated or amplified or some specific ones of them rejected or isolated.

Filtering could be used in applications to eliminate a signal noise, remove signal distortion, and separate purposely-mixed signals or resolve signals into their frequency components. It could also be used in other tasks such as signal demodulation, convert discrete-time signals to band limit signals, data smoothing, spectrum analysis and electrocardiogram processing.

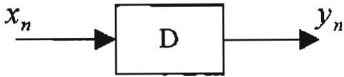
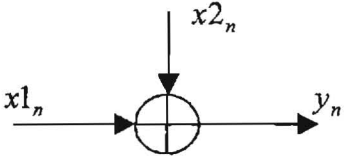
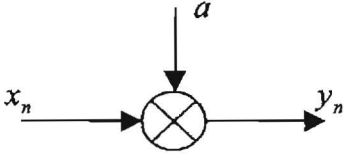
Digital filters are possible to be implemented by software or by dedicated hardware. In both cases, they can be used to filter real-time (frequency of samples must be very small) or non-real-time signals. Digital filters of the same characteristics can replace analogue filters used in real-time filtering purposes.

The advantages offered by using digital filters instead of analogue ones is the high accuracy, small physical size, high reliability, flexibility, and that component tolerance is non-critical on the system performance. Since a digital filter has been designed, the filter coefficients can be modified to change the filter characteristics. This has as an advantage the use of the same filter for different filtering tasks.

Digital filters can be modelled using three basic elements. These are the delay, the adder and the multiplier. Delay elements are implemented using registers. Adders and



multipliers could be implemented using networks of logic gates such as NAND or NOR gates. Table 1 shows the basic digital-filter elements.

	Element	Equation
Delay		$y_n = x_{n-1}$
Adder		$y_n = x1_n + x2_n$
Multiplier		$y_n = ax_n$

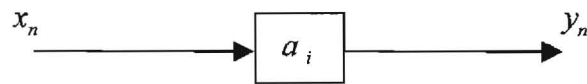
**Table 1:** Basic digital-filter elements

There are two types of digital filters, the *nonrecursive* filters and the *recursive* filters. Nonrecursive filters are also known as finite impulse response filters (FIR) and recursive filters as infinite impulse response filters (IIR).

Nonrecursive filters are the simplest ones and they are defined by Eq. (1)

$$y_n = \sum_{i=0}^M a_i x_{n-i} \quad (1)$$

Where  $a_i$  are the coefficients of the filter and determine its characteristics.  $x_{n-i}$  and  $y_n$  are the input and output data streams. A simple block diagram of a nonrecursive filter is shown in Fig. 2.

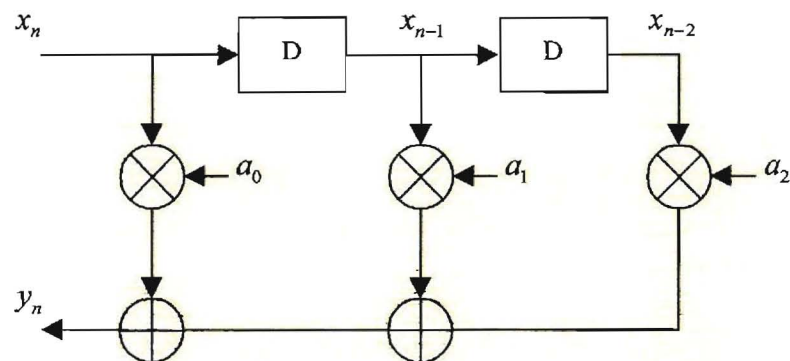


**Figure 2:** Nonrecursive filter block diagram

For a second order ( $M=2$ ) FIR filter, Eq. (1) becomes

$$y_n = a_0x_n + a_1x_{n-1} + a_2x_{n-2} \quad (2)$$

The data flow diagram constructed from Eq. (2) is shown in Fig. 3.



**Figure 3:** Second order ( $M=2$ ) FIR filter

IIR filters compute their output recursively, which means that they need the immediate past output for computing the current one. This feature makes IIR digital filters are more difficult to pipeline than FIR filters. Also IIR filters have the advantages of high selectivity and requiring less coefficients than the FIR with similar performance.

The block diagram of a recursive filter is shown in Fig. 4 on the page overleaf.

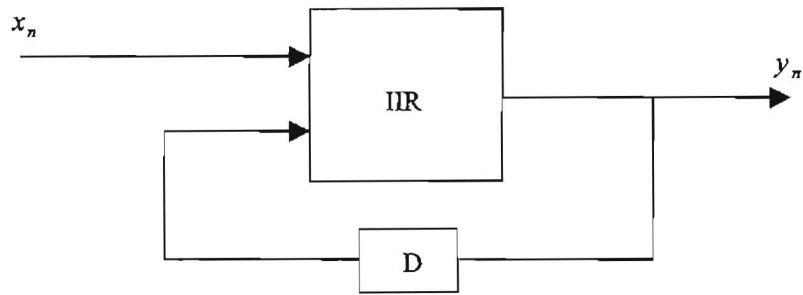


Figure 4: Recursive filter block diagram

Recursive filters are defined by

$$y_n = \sum_{i=0}^M a_i x_{n-i} + \sum_{i=1}^M b_i y_{n-i} \quad (3)$$

Where  $a_i$  and  $b_i$  are the coefficients of the filter.  $x_{n-i}$  and  $y_n$  are the input and output data streams. For a second order ( $M=2$ ) IIR filter, Eq. (3) becomes

$$y_n = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2} + b_1 y_{n-1} + b_2 y_{n-2} \quad (4)$$

The data flow diagram of the filter is shown in Fig. 5 below.

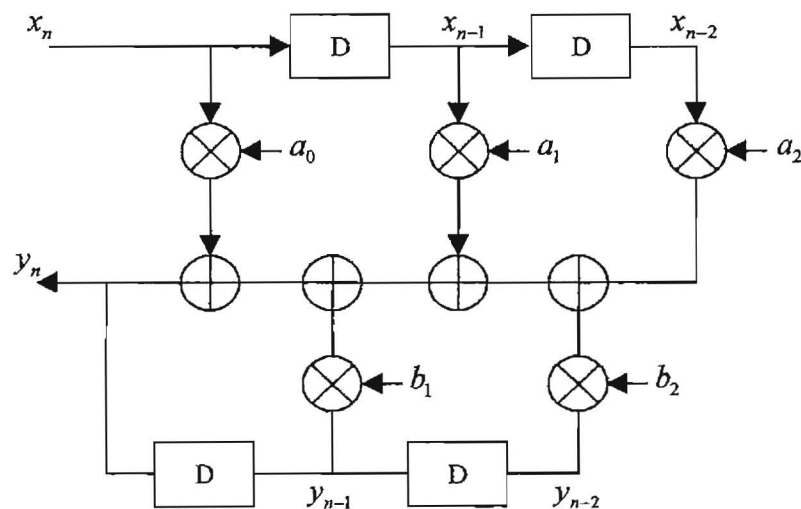


Figure 5: Second order ( $M=2$ ) FIR filter

## ➤ Scope of the project

The project is based on the implementation of a first order **digit-serial IIR filter**, on FPGA by either using VHDL or CAD software packages (*Viewlogic*).

The feed back loop in IIR filters makes them difficult to be pipelined. The application of the digit-serial structures to the design of IIR filters introduces delay elements in the feed back loop of the IIR filter. This offers the possibility of pipelining the feed back loop inherent in the IIR filters. The digit serial structure is based on the feed forward of the carry digit, which allows sub digit pipelining to increase the throughput rate of the IIR filters.

The implementation of the digital filter can be split in several tasks according to its block diagram. The first task is to implement a new cell architecture for digit-serial multiplication (digit serial multiplier). After that a digit-serial adder, should be implemented.

Furthermore the first order digit-serial IIR filter should be composed from the elements that already have been implemented, simulated and verified with already known results to make sure that is functioning properly and producing the correct results. Finally the first order digit-serial IIR filter must be downloaded to the FPGA chip and tested again to ensure that it works properly.

In look-ahead computation techniques, the algorithm is iterated as many times as required to create the necessary level of concurrency and the iterated version is implemented. Specifically, the first-order recursion is iterated to express the state  $x(n)$  as a function of  $x(n-M)$  to create  $M$  delay operators inside the loop so that the loop can be pipelined by  $M$  stages. This iteration process contributes to a non-recursive  $O(M)$  multiplication complexity.

In an  $N$ th-order recursive system, the state  $x(n)$  is expressed as a function of past  $N$  states  $x(n-1)$ ,  $x(n-2)$ , ..., and  $x(n-N+1)$ . In these higher order systems, look-ahead can be either clustered or scattered.

In scattered look-ahead approach,  $x(n)$  is expressed as a function of past  $N$  scattered states  $x(n-M)$ ,  $x(n-2M)$ , ..., and  $x(n-NM)$ , thus emulating the original  $N$ th-order filter by an  $NM$ -order filter. The scattered look-ahead process leads to an  $O(NM)$  complexity which guarantees stability.

However, the drawback of this technique is the overhead in hardware complexity, which is proportional to the number of pipelining levels. [12]

### **1.3 Signed-digit number representation**

Signed-Digit Number representations (SDNRs) were originally introduced by Avizienis [20] to eliminate carry propagation chains in operations such as add, subtract, multiply and divide.

Signed-digit numbers differ from conventional numbers in that the individual digits may assume negative as well as positive values and hence there is no need for an explicit mechanism (such as the 2's complement system in binary) to handle the overall sign of a number.

For example in radix-2 SDNR, the digits may assume the values 1,0 or  $-1$  (denoted by  $-1$ ). For higher radices, there is some choice in the digit set, which can be used; symmetric digit sets for radix-4 can be chosen as either  $\{2\dots 2\}$  or  $\{3\dots 3\}$ . The smallest set is termed the minimally redundant set and contains at least  $r+1$  values (where  $r$  is the radix). The largest set is termed the maximally redundant set and contains at most  $2r-1$  values.

Such numbers are termed redundant because there may be several possible representations for any given algebraic value. For example, the decimal value 3 may be represented in radix-2 SDNR as 011, 101, or 111 etc.

Redundancy in the number system used allows methods of addition and multiplication to be devised in which each digit of the result is (typically) a function only of the digits in two or three adjacent positions of the operands and does not depend on the other digits in any way.

This feature has a number of important consequences. (1) It allows arithmetic operations to be carried out completely in parallel with no carry propagation from the least significant digit (LSD) through to the most significant (MSD). (2) The time required for an operation such as parallel addition is constant and does not depend on

the word length. (3) The calculation of least significant digits may be avoided in situations where they are not required since the calculation may be performed with the most significant digits first.

The drawbacks of this method are the increased size of the computational elements, because of the use of signed digits rather than conventional binary digits and also the hardware overhead required for data conversion from signed digit number representation, to two's complement representation, and vice versa. [5]

## CHAPTER 2 Radix-2<sup>n</sup> arithmetic

### 2.1 Introduction

High degree of pipelining applied to digit-serial systems [17] has been proved to increase the throughput rate and as a result enhance the performance of the systems. However, since the initial delay is increasing by the number of pipelining levels, the chances of delivering an accurate and high-speed clock, are decreasing respectively. Obviously a solution to this problem would be to reduce the number of pipelining levels [2].

Recently, a new approach to the design of digit-serial structures has been proposed based on the radix-2<sup>n</sup> arithmetic [4,21]. The new radix approach has a number of advantages which are (1) it has enabled for the first time, the design of functionally correct digit-serial multipliers without the need of bit-serial or bit-parallel structures as an initial starting point, (2) it is more general than the bit-level cellular arrays approach due to the fact that more designs can be derived from the radix approach, (3) it allows, for the first time the direct application, of all the existing synthesis methods in designing digit-serial structures, (4) it only specifies the functionality of the basic cell, and hence any internal architecture, can be used as long as it satisfies the functionality specification of the cell [2].

The radix-2<sup>n</sup> algorithm can be used to find the best trade off between cost and time,



for the particular application being considered. Furthermore, the utilisation of a great number of different architectures becomes available, one for each radix.

## 2.2 The Radix-2<sup>n</sup> approach

Radix-2<sup>n</sup> arithmetic is based on Radix-2 (binary) arithmetic in general. On radix-2<sup>n</sup> the simplest building block is an AND gated Full Adder. The basic cell radix-2<sup>n</sup> arithmetic performs the multiplication of two n-bit digits,  $u_i$  and  $v_j$ , then the products of the multiplication would be added to two other n-bit digits,  $s_{in}$  and  $c_{in}$ . The performed computation is described by Eq. 5.

$$\{c_{out}, s_{out}\} = u_v + s_{in} + c_{in} \quad (5)$$

Equation (y) shows that the result will always be a two-digit number. According to the algorithm, the number will be split in the most significant digit (MSD) in the carry digit,  $c_{out}$  and the least significant digit (LSD) in the sum digit,  $s_{out}$ .

As a result of the above technique, any bit-level architecture can be used to perform a radix-2<sup>n</sup> algorithm by replacing the bit-level basic block with a radix-2<sup>n</sup> basic cell. [2]

### 2.3 New Radix-2<sup>n</sup> vector inner product algorithms

Let assume that the inner product of vector  $U = (U_0, U_1, \dots, U_{M-1})$  and  $V = (V_0, V_1, \dots, V_{M-1})$  can be obtained by multiplying one pair of numbers  $(U_M, V_M)$  and add their product to an accumulating result. This can be described at the word level by a simple recursion of the form  $W_M \leftarrow W_M + U_M V_M$ .

Assume that the elements  $U_M$  and  $V_M$  are unsigned numbers and can be divided into  $K$  digits of  $n$ -bit each. If  $u_{im}$  and  $v_{jm}$  represent the  $i$ th and  $j$ th digits of  $U_M$  and  $V_M$  respectively, then  $U_M$  and  $V_M$  will be equal to,

$$U_m = \sum_{i=0}^{K-1} u_{im} 2^{in} \quad \text{and} \quad V_m = \sum_{j=0}^{K-1} v_{jm} 2^{jn} \quad (6)$$

The multiplication  $W_m = U_m V_m$  can be computed, according to the following equation,

$$W_m = \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} 2^{(i+j)n} u_{im} v_{jm} \quad (7)$$

The Eq. (7) can be written in a recursive manner using Eq. (5). Because the partial product  $u_{im} v_{jm}$  is a  $2n$ -bit number,  $2n$ -bit adders are required for the accumulation in each radix- $2^n$  cell.

As a result, a new set of radix- $2^n$  can be derived if these partial products are split into the most significant digit (MSD),  $(u_{im} v_{jm})_{MSD}$ , and the least significant digit (LSD),  $(u_{im} v_{jm})_{LSD}$ .

According to that,  $W_M$  can be rewritten as,

$$W_m = \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} 2^{(i+j)n} ((u_{im} v_{jm})_{MSD} + (u_{im} v_{jm})_{LSD}) \quad (8)$$

The new radix- $2^n$  algorithms are resulting in a more efficient implementation of the radix- $2^n$  cell, by reducing the length of the adders required for the accumulation [2].

## 2.4 Radix- $2^n$ arithmetic cell

This section describes the architecture of the radix- $2^n$  arithmetic (basic) cell used to compute Eq. 5.

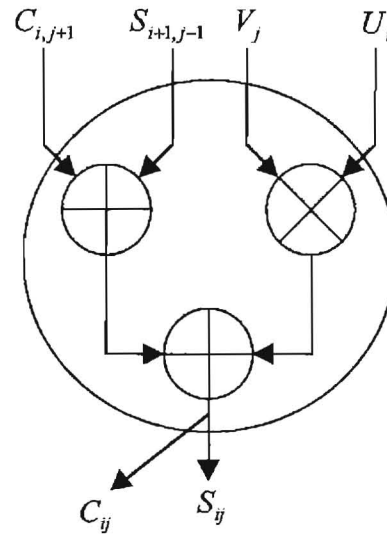
Eq. 5 can be also expressed as

$$\{c_{ij}, s_{ij}\} = u_i v_j + s_{i+1, j-1} + c_{i, j-1} \quad (9)$$

Multiplying two radix- $2^n$  digits and adding the product to two radix- $2^n$  digits, the result can be represented by a  $2n$ -bit number and hence a two radix- $2^n$  digits.

The radix- $2^n$  arithmetic cell performs the multiplication of two radix- $2^n$  digits and adds the product to two radix- $2^n$  digits. The output of the cell is two radix- $2^n$  digits of which the least-significant digit is the carry.

The architecture of the arithmetic cell used can be specified using any design criteria. One possible architecture is shown in Fig. 6.



**Figure 6:** Architecture of radix- $2^n$  arithmetic cell [4]

The basic cell consists of one  $n$ -bit multiplier, one  $n$ -bit adder and one  $2n$ -bit adder. Since that  $s$  and  $c$  are  $n$ -bits, only one  $n$ -bit adder is needed to add  $s$  and  $c$ . A  $2n$ -bit adder is needed for the second addition since the product  $uv$  is  $2n$ -bits.

From the above description it can be easily seen that, when  $n=1$ , the resulting multiplication algorithm is based on binary arithmetic [4].

## CHAPTER 3 BIT LEVEL PIPELINED IIR FILTER

### 3.1 Introduction

A new systolic architecture for high performance IIR digital filters based on two's complement number representation is presented for the first time. It is based on digit-serial computations [11-19], in which data words are decomposed into digits of some number of bits and the computations are carried out one digit at a time. It is shown that the application of the digit-serial structures to the design of IIR digital filters introduces extra delays in the recursive part of the IIR filter, which offers the possibility of pipelining of the feed back loop. The number of delay elements added in the recursive part is equal to the number of digits used to represent the partial results.

New cell architectures for digit-serial computation that offer a high degree of pipelining (bit-level pipelining) have been proposed in recent publications [7-19]. These architectures involve a feed forward of the carry digits which was proven functionally correct using radix- $2^n$  arithmetic. The use of carry feed forward has solved the major bottleneck of the carry feedback loops inherent in existing digit-serial designs. Also, the flexibility offered by the radix- $2^n$  approach in choosing the cell architecture has enabled the design of the basic cells using carry save arithmetic, which is faster and requires less area. The possibility of bit-level pipelining offers high-speed realisation of digit-serial systems. It was shown that bit-level pipelined digit-serial structures are much faster than the fully bit-parallel structures and use much less hardware [18].

The bit-level digit-serial IIR filter architecture is based on the radix-2<sup>n</sup> arithmetic approach to the realisation of bit-level pipelined digit-serial structures to take full advantage of the number of delay elements added in the recursive part. In this case, the number of pipelining levels can be made varied to obtain different trade-offs between area and speed and is only limited by the number of digits used to represent partial results. Digit-serial IIR filter architectures has several advantages over existing bit-parallel structures based on two's complement number representation. It can be pipelined to the sub-digit level to increase the throughput rate. The throughput rate is much higher than the fully bit parallel case. At the same time, the size of the hardware required and the number of I/O pins are reduced greatly. [1]

### 3.2 Design methodology of digit-serial IIR filter

In this section a systematic design methodology of digit-serial IIR filter is presented based on the radix-2<sup>n</sup> arithmetic and the classical theoretical framework of regular array architecture synthesis developed by Kung S Y [23]. To show how the digit-serial IIR filter architecture is derived, a first order filter is considered. The direct form of computation algorithm for a first order IIR digital filter at the word level is defined by

$$y_k = u_k = b_1 y_{k-1} \quad (10)$$

where  $u_k$  is the nonrecursive computation and is equal to  $a_0 x_k + a_1 x_{k-1}$ .  $x_k$  and  $y_k$  are the input and output data streams while  $a_0$ ,  $a_1$  and  $b_1$  are the filter coefficients.

In digit-serial computation the data and coefficient words are subdivided into M

digits. It is assumed that the output,  $y_k$ , is truncated to  $M$  digits before being fed back into the IIR filter.

The computation of the recursive component of the first order IIR filter is given by  $v_{k-1} = b_1 \hat{y}_{k-1}$ , where  $\hat{y}_{k-1}$  is the truncated output of  $y_{k-1}$ . All the products involved in the nonrecursive computation can be performed using the same procedure as for the computation of  $v_{k-1}$ . Using the radix- $2^n$  arithmetic, the truncated output  $\hat{y}_{k-1}$  and the coefficient  $b_1$  are represented using  $M$  digits and can be written as

$$\hat{y}_{k-1} = \sum_{i=0}^{M-1} \hat{y}_{k-1,i} 2^{in} \quad \text{and} \quad b_1 = \sum_{j=0}^{M-1} b_{1,j} 2^{jn} \quad (11)$$

where  $n$  is the digit size and  $\hat{y}_{k-1,i}$  and  $b_{1,j}$  represent the  $i$ th and  $j$ th digits of  $\hat{y}_{k-1}$  and  $b_1$  respectively. Hence,  $v_{k-1}$  is given by

$$v_{k-1} = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} b_{1,j} \hat{y}_{k-1,i} 2^{(i+j)n} \quad (12)$$

Several radix- $2^n$  multiplication algorithms have been proposed by the authors [11-12]

Which can be used to compute  $v_{k-1}$ . One of the radix- $2^n$  multiplication algorithms involves the partitioning of the partial products of the form  $b_{1,j} \hat{y}_{k-1,i}$  into the most significant digit  $(b_{1,j} \hat{y}_{k-1,i})_{\text{MSD}}$  and the least significant digit  $(b_{1,j} \hat{y}_{k-1,i})_{\text{LSD}}$  [18]. Using these algorithms, Eq. (12) can be rewritten as

$$v_{k-1} = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} [(b_{1,j} \hat{y}_{k-1,i})_{\text{MSD}} 2^{(i+j+1)n} + (b_{1,j} \hat{y}_{k-1,i})_{\text{LSD}} 2^{(i+j)n}] \quad (13)$$

In the tree dimensional space  $(k,i,j)$  shown in Fig. 7,  $i+j$  form a family of vertical planes and represent the significance.

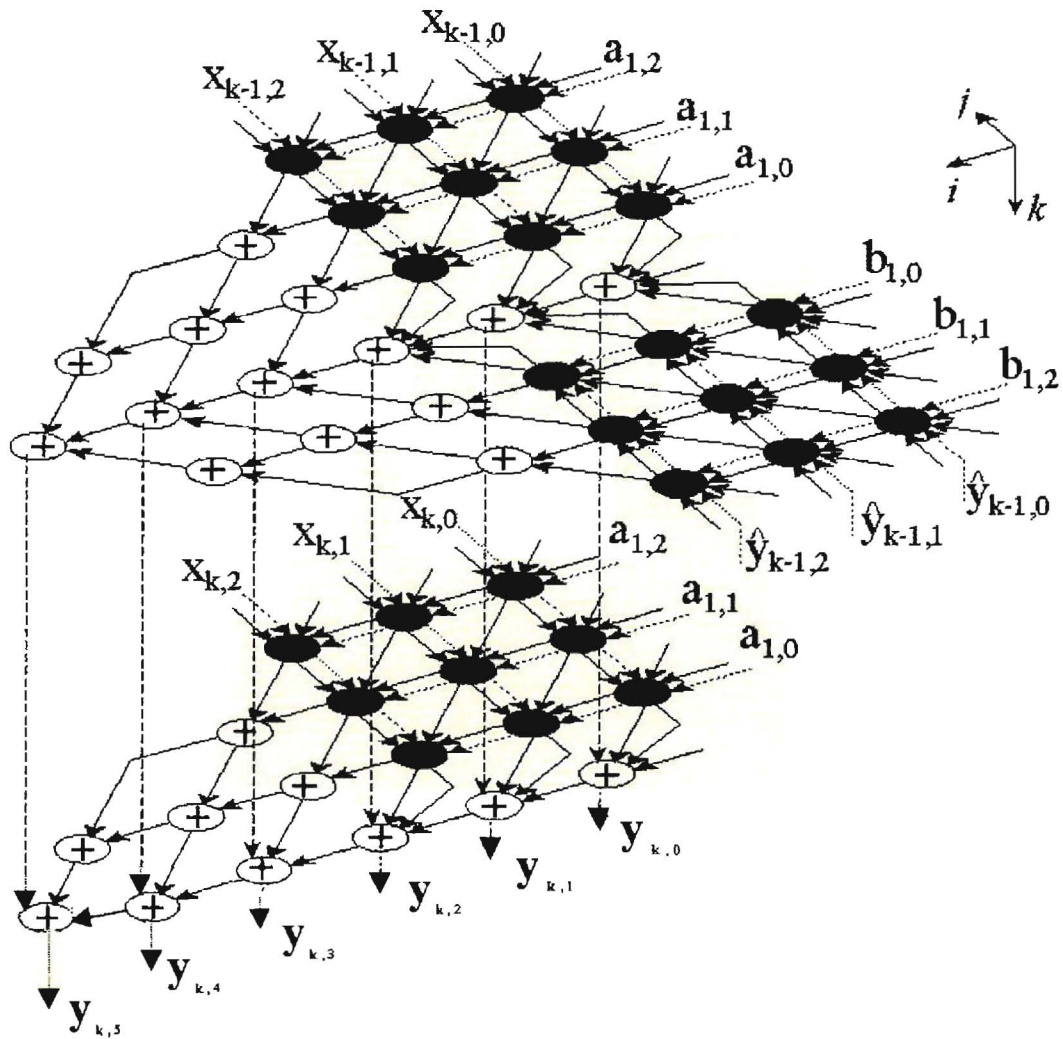


Figure 7: Dependency graph of the first order radix IIR filter

From Eq. (13) it can be seen that, the MSDs generated at the plane  $(i+j)$  must be added to any of the LSDs on the next highest significance at the  $(i+j+1)$  plane. This implies that either the LSDs are transferred to the next lowest significance or the MSDs are transferred to the next highest significance.

These transfers can be performed on the same horizontal plane  $k$  or from one horizontal plane to another. In this paper, the former is considered. One way to



compute Eq. (13) is to transfer the LSDs generated in the node  $(k, i, j+1)$  to be added to the MSDs generated in the node  $(k, i, j)$ . Using this approach, the digits of the recursive computation,  $v_{k-1}$ , can be formed in the following recurrence

$$\{c_{kij}, s_{kij}\} = (b_{1,j}\hat{y}_{k-1,i})_{MSD} + (b_{1,j+1}\hat{y}_{k-1,i})_{LSD} + s_{k,i-1,j+1} + c_{k,i-1,j} \quad (14)$$

where  $s_{kij}$  is an  $n$ -bit digit and is the sum of the partial product along the lines formed by the intersection of the vertical planes,  $i+j$  with the horizontal planes,  $k$ , and  $c_{kij}$  is the partial carry generated at the node  $(k, i, j)$ . It can be easily shown that  $c_{kij}$  is a 2-bit wide by stating the fact that the maximum value of the term on the right hand side of Eq. (14) is  $3(2^n - 1) + 3$  which is a  $(n+2)$ -bit number.

The nonrecursive computation of the first order IIR filter,  $u_k$ , can be formed by first using an equation similar to Eq. (14) to compute each product  $a_0x_k$  and  $a_1x_{k-1}$ , vis

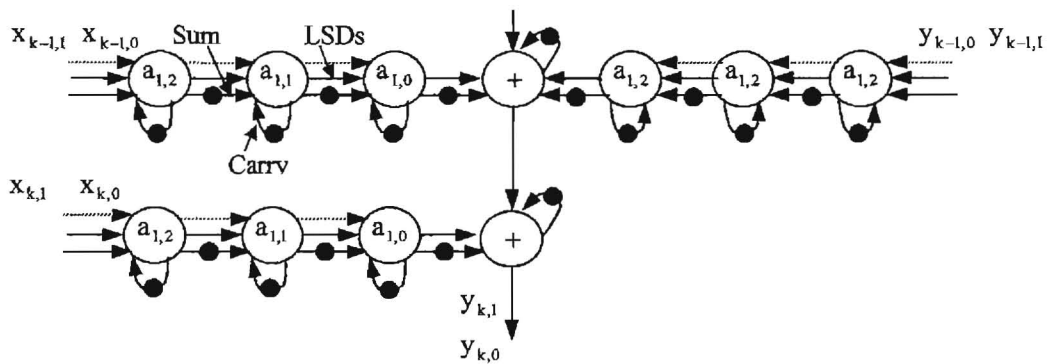
$$\begin{aligned} \{c_{kij}, s_{kij}\} &= (a_{0,j}x_{k,i})_{MSD} + (a_{0,j+1}x_{k,i})_{LSD} + s_{k,i-1,j+1} + c_{k,i-1,j} \\ \{c_{kij}, s_{kij}\} &= (a_{1,j}x_{k-1,i})_{MSD} + (a_{1,j+1}x_{k-1,i})_{LSD} + s_{k,i-1,j+1} + c_{k,i-1,j} \end{aligned} \quad (15)$$

Once the three products involved in Eq. 10 are computed, all their digits with the same significance are accumulated along the  $k$  axis to compute the digits of the output  $y_k$ .

A dependency graph, which represents the above computation, is shown in Fig. 7. It is embedded in a three dimensional index space represented by  $(k, i, j)$ . Each horizontal plane of this graph shows the interaction of the digits within each data and coefficient words and is such that each digit of one data word interacts with each digit of the corresponding coefficient word as required.

The horizontal planes in Fig. 7, perform the computation of the three products  $a_0x_k, a_1x_{k-1}$  and  $b_1\hat{y}_{k-1}$  using radix- $2^d$  arithmetic. The products,  $a_1x_{k-1}$  and  $b_1\hat{y}_{k-1}$ , are computed on the same horizontal plane,  $k-1$ . Their accumulation is performed on that same plane. The  $2M$  digits of the term  $(a_1x_{k-1} + b_1\hat{y}_{k-1})$  are transferred down the horizontal plane,  $k$ , to be added to the  $2M$  digits of the product  $a_0x_k$  with the same significance to form the digits of the output,  $y_k$ , as shown by the vertical plane in Fig. 7. It is worth mentioning that the  $M$  digits of the previous truncated output  $\hat{y}_{k-1}$ , (i.e.  $\hat{y}_{k-1,0}, \hat{y}_{k-1,1}$  and  $\hat{y}_{k-1,2}$  in Fig. 7) are equal to the  $M$  MSDs of  $y_{k-1}$ , (i.e.  $y_{k-1,3}, y_{k-1,4}$  and  $y_{k-1,5}$  in Fig. 7).

The dependency graph can be projected in several directions to obtain different digit-serial IIR filters. Considering the projection in direction  $[100]^T$ , the first order digit-serial IIR filter shown in Fig. 8 is obtained, where, each row on each side of the accumulation path represents a digit-serial multiplier. [1]



**Figure 8:** First order digit-serial IIR filter obtained by projecting the dependency graph in Fig. 8 in the direction  $[0 \ 1 \ 0]$

### 3.3 Pipelined digit-serial IIR filter:

Another major advantage of the radix methodology for the design of digit-serial structures is that only the functionality of the basic cell is specified and hence any internal architecture can be used so long as it satisfies the functionality specification architecture, has enabled the design of the basic cells using carry save arithmetic which is faster and requires less area and allows bit-level pipelining [1]

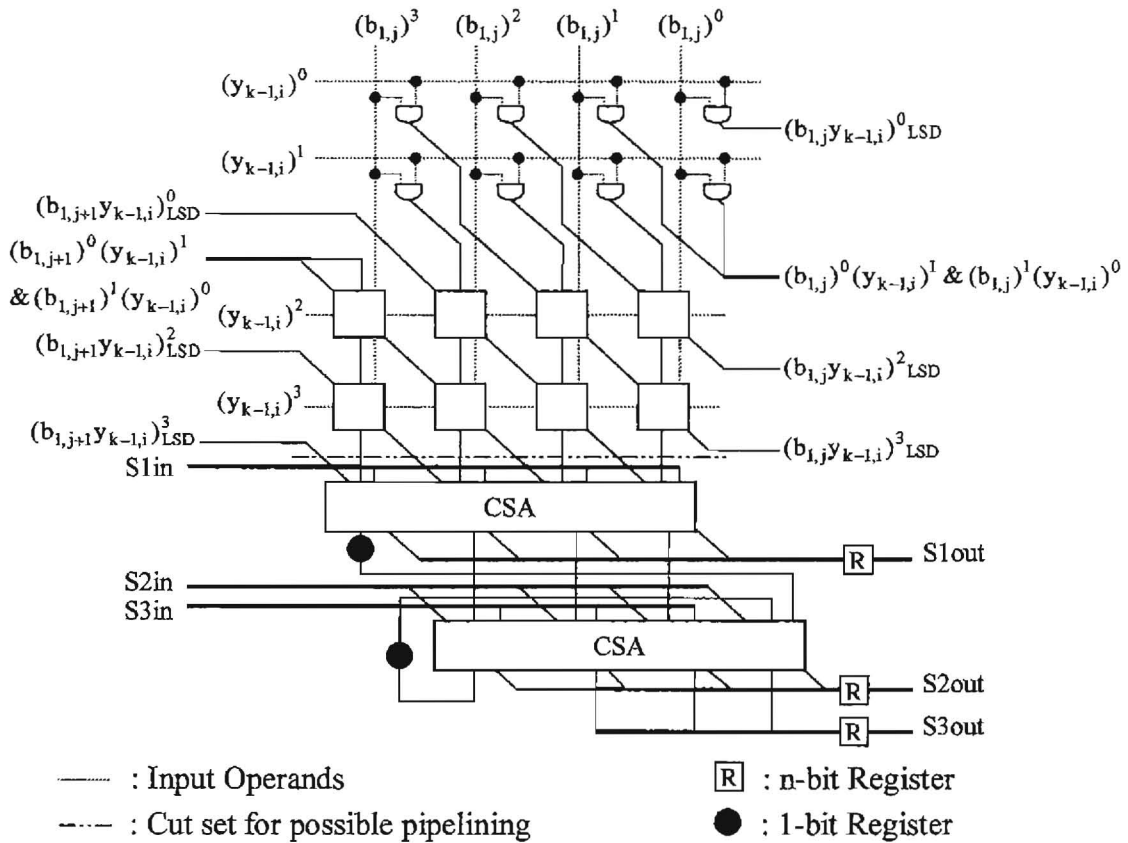
### 3.4 Bit-level pipelined digit-serial multiplier

The digit-serial multiplier used in the digit-serial IIR filter is similar to that proposed by the authors in [18]. The basic cell of the digit-serial multiplier can be implemented using a carry-save array multiplier. The basic cell which computes Eq. (14) is shown in Fig. 9 for  $n=4$  where the upper indices are used to represent the bit significance. It uses the fact that the full adders at the boundaries of the carry save array multiplier have free bit positions. The  $n$  sum bits produced on the right hand side of the array multiplier which form the LSD,  $(b_{l,j}\hat{y}_{k-1,i})_{\text{LSD}}$ , are transferred to the next cell on the right and added with the MSD,  $(b_{l,j-1}\hat{y}_{k-1,j})_{\text{MSD}}$ . This addition is performed using the empty bit positions on the left-hand side of the array multiplier.

In conventional carry save array multipliers, a carry propagate adder is required to add the carry and sum bits from the lower row of full adders to form the carry digit. To allow bit-level pipelining, these carry and sum bits from the lower boundary of the array multiplier are summed with the partial sums from the previous cell using two

carry save adders as shown in Fig. 9. The partial result from previous cell on the left consists of three digits,  $S1_{in}$ ,  $S2_{in}$ ,  $S3_{in}$ . The digit,  $S1_{in}$ , is summed to the two digits from the lower boundary of the array multiplier using the first CSA to produce two 1-digit outputs. One of the two digits,  $S1_{out}$ , obtained from the first CSA is fed to the next cell on the right. While the second digit is fed down into a second CSA to be summed with the two digits,  $S2_{in}$  and  $S3_{in}$ , coming from the cell on the left. The two carry bits produced by the two CSAs are summed to form the carry digit,  $c_{kij}$ . The carry digit,  $c_{kij}$ , if fed back into the same cell to be added as shown by Eq. (14). This is performed using the empty least significant bit position available in the second CSA as shown Fig. 9. As it can be seen Fig 9, the feedback of  $c_{kij}$  will not effect the pipelining, since the delay within the loop is one FA delay. It should also be noted that the dependency graph can be modified to allow  $c_{kij}$  to be fed forward to the next cell.

The addition of the two CSA adders can be compensated by the fact that only  $n-2$  CSAs are required to add the partial products of the multiplication of two  $n$ -bit digits. This is due to the fact that the top three  $n$ -bit partial products of an  $n \times n$  multiplication can be summed using only one  $n$ -bit CSA as shown in Fig. 9. The two least significant bits (LSBs) of the first  $n$ -bit partial product, and the LSB of the second  $n$ -bit partial product of an  $n \times n$  multiplication, are not computed within the present cell but fed to be summed in the next cell on the right using the two full-adders (FA), occupying the two most significant bit (MSB) positions as shown in Fig. 9. The digits obtained from the last cell on the right of each digit-serial multiplier are added together using two CSAs to produce two digits. [1]



**Figure 9:** Cell architecture of the digit-serial multiplier based on an array multiplier

### 3.5 Truncation of the output

Assuming that  $2M$  digits are used for the partial results, the application of digit-serial computation to design IIR filter introduces  $2M$  delay cycles in the feed back loop which can be used for sub-digit pipelining of the proposed architecture. However, if the whole output word is fed back, this will result in a word length growth due to the recursive nature of the IIR filter. It was reported in [24] that since the filter will have a unity gain, such a wordlength growth represents a growth in precision, which implies that the  $M$  most significant digits (MSDs) of the output have the same

significance as the  $M$  digits of the input data,  $x_k$ . Hence the output,  $y_k$ , can be truncated to its  $m$  MSDs before it is fed back into the recursive part to prevent any growth in the wordlength. Figure 10 shows the timing procedure of an IIR filter, where  $x_k$  and  $y_k$  are the input and output data streams, and  $\hat{y}_k$  is the truncated output which is fed back to the IIR filter. It is clear from the above and Fig. 10, that the truncated output,  $\hat{y}_k$ , has the same significance as the input sample,  $x_{k+1}$ , which implies that the  $M$  LSD of  $\hat{y}_k$  (which represents the  $M$  MSDs of the output,  $y_k$ ), must be feedback in a serial fashion at the same time as the digits of the input data  $x_{k+1}$ , as shown in Fig. 10. For the example, the first digit of  $\hat{y}_k$ , which is the  $(M+1)$ th digit of  $y_k$ , must be fed at the same time as the first digit of  $x_{k+1}$ . As a result, there are only  $M$  delays between the time in which the  $M$  MSDs of  $y_k$  are computed and the time they are feedback as the truncated value,  $\hat{y}_k$ , for the computation of the next output samples. This implies that only  $M$ , rather than  $2M$  delay cycles are available in the feedback loop, which can be used for sub-digit pipelining of the proposed architecture.

Finally, it should be note that since the computation of each output sample requires  $M$  cycles, the values of the  $M$  MSD digits of  $\hat{y}_k$  are set to zeros during the second set of  $M$  cycles in a similar fashion as the input samples  $x_k$ . This implies that in the second  $M$  cycles of each sample calculation, zeros are fed to both the input and the feedback path of the filter [1].

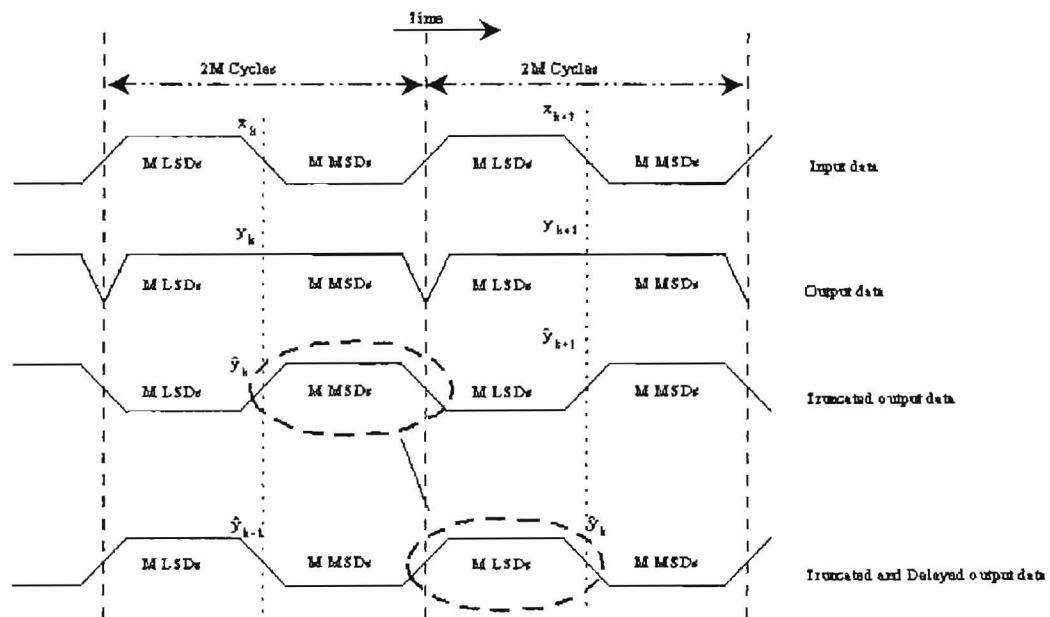


Figure 10: Timing procedure of a digit-serial IIR filter

### 3.6 Systolic digit-serial IIR filter structure:

The first order digit-serial IIR filter shown in figure 8, can be generalised to any order,  $K$ , by replicating the top two digit-serial multipliers  $K$  times. It should be pointed out that the computation of  $a_1 x_{k-1}$  takes place  $2M$  cycles after the start of the computation of  $a_0 x_k$ . This implies that  $2M$  delay elements are required in the recursive and nonrecursive data paths between the digit-serial multipliers. The accumulation of the three terms in the first order IIR filter can be rearranged so that the  $a_0 x_k$  is first added to  $b_1 \hat{y}_{k-1}$  then added to  $a_1 x_{k-1}$ . In general, the cell of the digit-serial IIR filter performs  $a_l x_{k-l} + b_{l+1} \hat{y}_{k-1+l}$ , where  $l = 0, \dots, K-1$ .

Since the data paths and the partial products path are in opposite directions,  $M$  of the  $2M$  delay elements in the data paths can be moved to the partial product path without





total of six digits are needed to be added using CSAs to produce two output digits. This is performed using an arrangement of CSAs as shown in figure 12. This arrangement is chosen to allow bit-level pipelining. The bit-level pipelining can be achieved by moving two of the M delay elements in the partial result path in between the two CSAs on the left as shown in figure 12. Since all the inputs to these two CSAs form the same path as the partial results from cell to cell, this rearrangement of the delay elements should not affect the functionality of the structure. Since the partial result path contain two digits, a final digit-serial adder is required in the feed back loop of the IIR filter which is shown in cell A in figure 11. To fully exploit the high degree of pipelining offered by the proposed architecture, a digit-serial bit level pipelined adder is required. The design of a bit-level pipelined digit-serial adder is reported in [17-19] and discussed in section 3.11 [1].

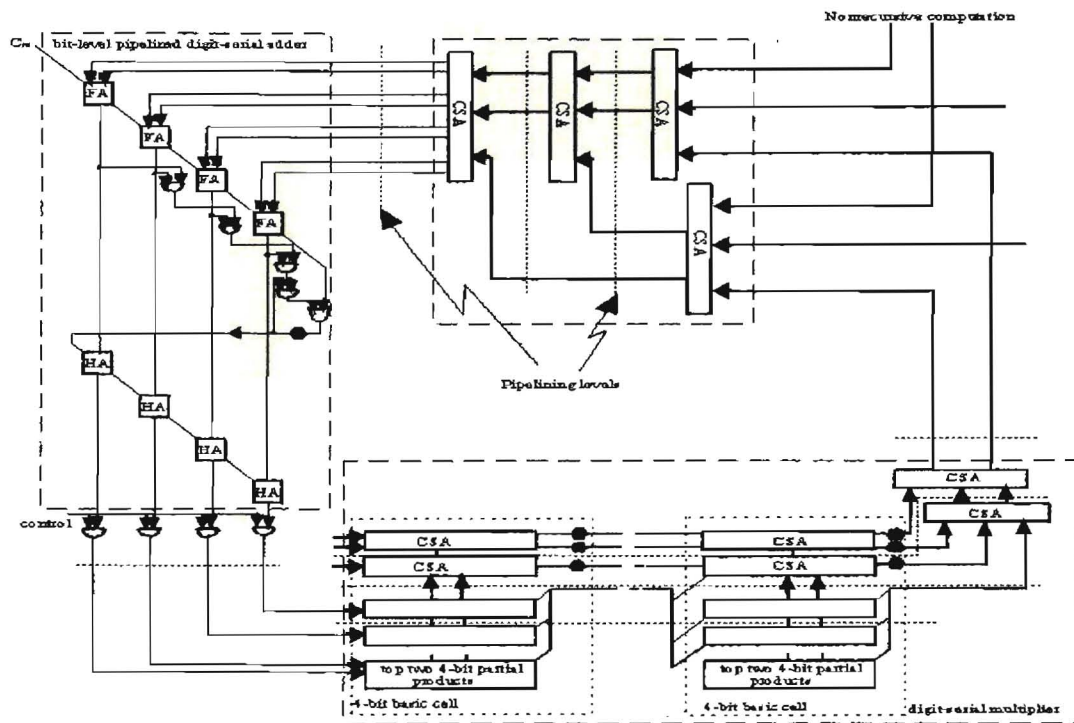
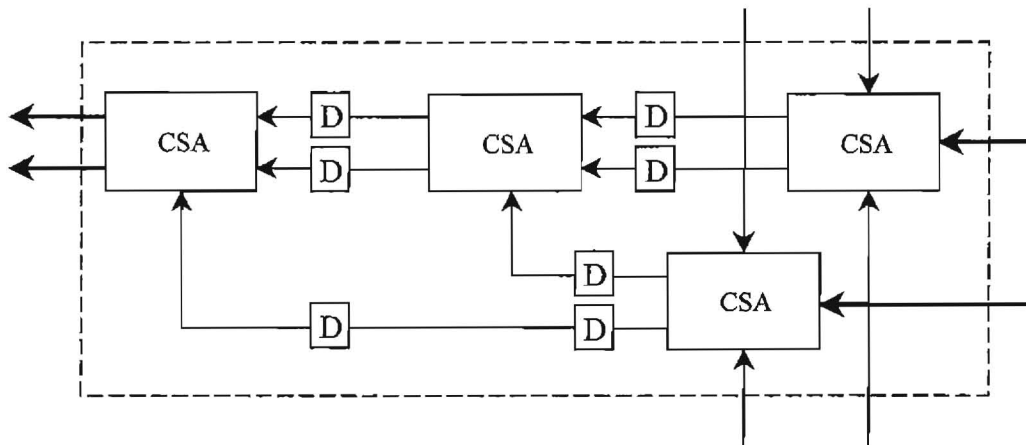


Figure 12: sub-digit pipelining of the feed back loop of the digit-serial IIR filter ( $n=4, M=32$ )

### 3.8 Pipelining of the digit-serial IIR filter:

As indicated in section 3.5, applying the digit-serial computation to the design of the IIR digital filters introduces  $M$  delay elements in the feed back loop, which allows  $M$  possible levels of pipelining. The delay elements are used to pipeline the digit-serial adder as well as the digit-serial multiplier in the recursive computation. To show how these delay elements are distributed over the digit-serial multiplier and adder, a first order digit-serial IIR filter is considered. Figure 13 shows details of the recursive computation part of the first order digit-serial IIR filter for  $n=4$  and  $M=8$ . Figure 13 also shows a detailed architecture of the bit-level pipelined digit-serial adder.



**Figure 13:** Accumulation of the recursive and nonrecursive partial products

### 3.9 Pipelining of the digit-serial adder:

The two CRA in the digit-serial pipelined adder as shown in figure 13, require  $(n + \frac{n}{2})$  full adder stages, where n is the digit size. However, a close examination of the internal structure of the full adder (FA) and the half adder (HA) shows that the propagation delay between the carry in and carry out is only that of two NAND gates. The time delay of the carry ripple digit-serial adder is given by that of the propagation of the carry bit through the CRA or the propagation through the tree of AND gates. The propagation delay through the AND gates or that of the carry bit within the CRA is given by the time delay of two EXORs plus that of  $2(n+1)$  NAND gate delays. It is assumed that the unit time (D) is that of one NAND gate and that one AND gate delay or one OR gate delay equals two NAND gate delays and that one EXOR gate delay equals three NAND gates. As a result the total time delay of the carry ripple digit-serial pipelined adder is  $(3n+5)D$ . Now, assuming that the time cycle of a bit-level pipelined structure is that of an AND gated FA and that a FA has a propagation delay of six NAND gate delays. Hence, the total number of equivalent AND gated FA stages within the carry ripple digit-serial pipelined adder is given by  $\frac{n+4}{4}$  [1].

### 3.10 Pipelining of the digit-serial IIR filter:

In order to achieve bit-level pipelining of the digit serial IIR filter, the number of delay elements in the feedback loop should be greater or equal to the total number of the equivalent AND gated FA stages in the digit serial adder plus those in the digit

serial/parallel multiplier. Bit-level pipelining of the digit-serial multiplier requires  $n$  pipelining levels, where  $n$  is the digit size. This implies that to attain bit-level pipelining of the digit-serial IIR filter, the digit size should satisfy the following condition,

$$n \leq \frac{4(M-4)}{5}$$

where  $N$  is the input data word length,  $N=Mn$  [1].

### 3.11 Design of bit-level pipelined digit-serial adder

The conventional digit-serial adder uses a carry propagate adder (CPA), where the last carry bit is fed back to the same adder which prevents bit-level pipelining. The first digit-serial bit-level pipelined adder was proposed in [17]. It uses two carry ripple adders (CRAs), where the carry bit produced by the first CRA is propagated forward and added to the next digit output using the second CRA as shown in figure 13. Note that the only case where carry bit obtained from the second CRA is equal to one is when the current output of the first CRA is  $2^n-1$ , where  $n$  is the digit size, and the carry from the previous digit is one. In this case the carry to the next significant digit should be set to one. A simple circuit that will propagate the correct carry to the second CRA is shown in figure 13. The digit from the first CRA is fed to a  $n$ -input AND gate to check whether it is equal to  $2^n-1$ . This is performed by using an array of 2-input AND gates arranged in a tree structure as shown in figure 13. The resulting bit and the carry bit obtained in the previous cycle are fed to 2-input AND gate to check whether the current output of the first CRA is  $2^n-1$  and the carry from the previous

digit is one simultaneously. The output bit and the carry of the current digit are fed to 2-input OR gate to calculate the correct bit to be fed to the second CRA.

As can be seen from figure 6, there is only one feedback loop within the digit-serial adder. However, the propagation delay within the feedback loop of the carry bit is equal to one AND gate plus one OR gate delay, and hence pipelining within the loop is not necessary. The remaining data paths in the digit-serial adder can all be pipelined to the bit-level since they are moving in the same direction.

The conventional digit-serial adder uses a carry propagate adder (CPA), where the last carry bit is fed back to the same adder which prevents bit-level pipelining. The first digit-serial bit-level pipelined adder was proposed in [17]. It uses two carry ripple adders (CRAs), where the carry bit produced by the first CRA is propagated forward and added to the next digit output using the second CRA as shown in figure 6. Note that the only case where carry bit obtained from the second CRA is equal to one is when the current output of the first CRA is  $2^n-1$ , where  $n$  is the digit size, and the carry from the previous digit is one. In this case the carry to the next significant digit should be set to one. A simple circuit that will propagate the correct carry to the second CRA is shown in figure 6. The digit from the first CRA is fed to a  $n$ -input AND gate to check whether it is equal to  $2^n-1$ . This is performed by using an array of 2-input AND gates arranged in a tree structure as shown in figure 6. The resulting bit and the carry bit obtained in the previous cycle are fed to 2-input AND gate to check whether the current output of the first CRA is  $2^n-1$  and the carry from the previous digit is one simultaneously. The output bit and the carry of the current digit are fed to 2-input OR gate to calculate the correct bit to be fed to the second CRA.

As can be seen from figure 6, there is only one feedback loop within the digit-serial adder. However, the propagation delay within the feedback loop of the carry bit is equal to one AND gate plus one OR gate delay, and hence pipelining within the loop is not necessary. The remaining data paths in the digit-serial adder can all be pipelined to the bit-level since they are moving in the same direction [1].

## **Chapter 4      IMLEMENTATION OF 1<sup>st</sup> ORDER DIGIT-SERIAL IIR FILTER**

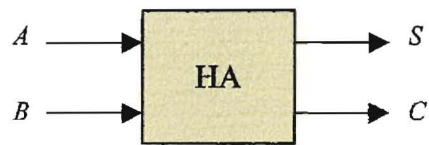
### **4.1 Introduction**

In this section the design work followed for the implementation of the digit-serial IIR filter (described in chapter 3) is presented. The digital design of each element of the digit-serial filter as well as the simulation and testing were performed using *Viewlogic* ECAD software. Great attention was paid to the design of the fundamental elements of the filter in order to achieve the minimum used number of gates possible. It should also mention that the number of bits processed in one clock cycle in the digit-serial systems is referred to as the digit-size.

### **4.2 Full Adder design and simulation**

One of the requirements set at the beginning of design of the full adder was to use as less number of gates as possible. Since the design of the digit-serial filter requires a big number of full adders, keeping the number of gates of each full adder low will result in a much less hardware complexity, improved speed performance, and reduced power consumption.

The design of the full adder can be subdivided in the design of a half adder first, since the first consists of two of them. A half adder is a circuit capable of adding two bits. It has two inputs and two outputs and its block diagram is shown in (Fig. 14).



**Figure 14:** Half adder block diagram

Where  $A$ ,  $B$  are two one-bit numbers and  $S$ ,  $C$  is the sum and carry inputs respectively.

The truth table for a half-adder is shown in table 2.

Input		Output	
$A$	$B$	$S$	$C$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

**Table 2:** Truth table of a binary half adder

From the truth table the following equations for the sum and carry were derived:

$$S = A\bar{B} + \bar{A}B = A \oplus B \quad (16)$$

$$C = AB \quad (17)$$

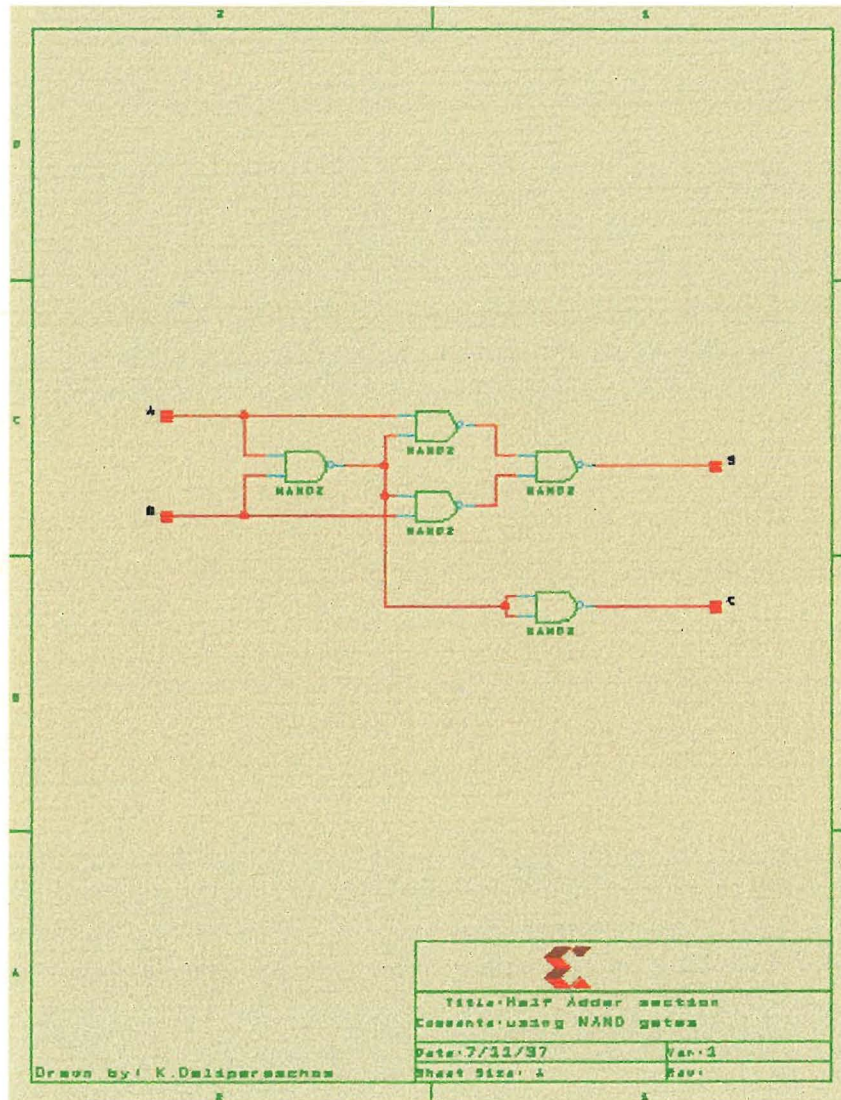
From equation 16 and 17, equation 18 and 19 were derived as follows:

$$S = \overline{\overline{A\bar{B} + \bar{A}B}} = \overline{\overline{A\bar{B}} \cdot \overline{\bar{A}B}} = A(\overline{\bar{A}B}) \cdot B(\overline{A\bar{B}}) \quad (18)$$

$$C = AB = \overline{\overline{AB}} \quad (19)$$



Using equation 18 and 19 the half adder can be implemented using NAND gates (Fig.15). The schematic capture program, *Viewdraw*, integrated in *Viewlogic* software, was used for the schematic capture.

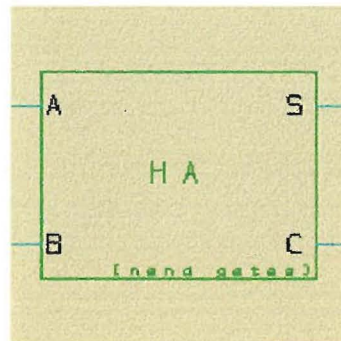


**Figure 15:** Half adder implementation using NAND gates

The inverter, NAND and NOR gates counts for one gate each, the AND and OR gates for two gates each, and the XOR and XNOR for three gates. According to the above

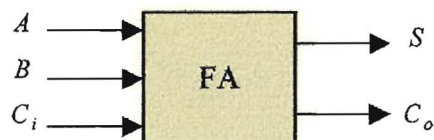
the design in Fig.15 only uses a total number of 5 gates ( $5 \times \text{NAND} = 5 \text{ gates}$ ), when compared to an AND/OR implementation which uses 10 gates ( $3 \times \text{AND} + 1 \times \text{OR} + 2 \times \text{INV} = 10 \text{ gates}$ ).

The symbol created out of the schematic in Fig. 15 is shown below (Fig. 16).



**Figure 16:** Half adder symbol

A full adder is a circuit capable of adding three bits. It has three inputs and two outputs. The block diagram of a full adder is shown in Fig. 17.



**Figure 17:** Full adder block diagram

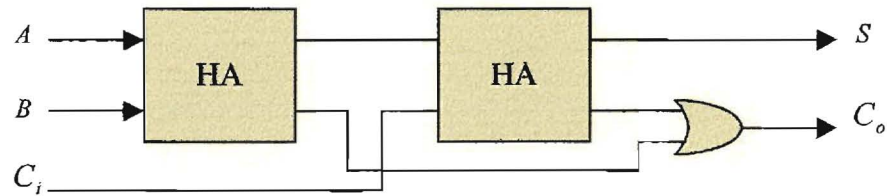
Where  $C_i$  is the carry-in from the previous addition and  $C_o$  is the carry-out to the next addition.

The truth table of a full adder circuit is shown in table 3.

Input			Output	
A	B	$C_i$	C	$C_o$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

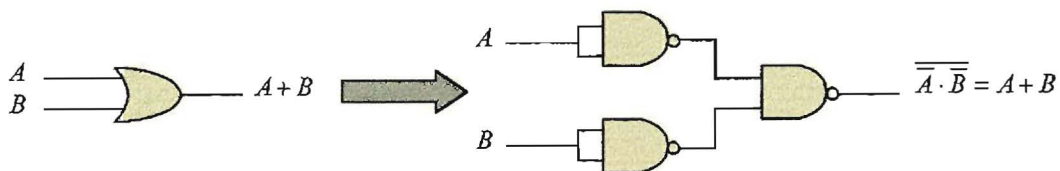
**Table 3:** Truth table of a binary full adder

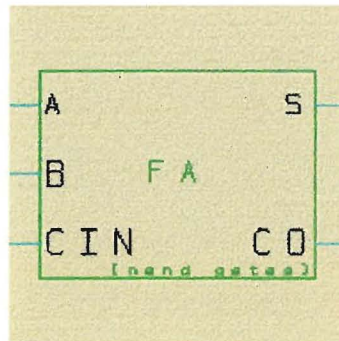
A full adder can be formed using two half adders (Fig. 18).



**Figure 18:** Block diagram of full adder comprising of two half adders

At this stage the full adder could be implemented, but this would result in a total of 12 gates. In order to reduce the total number of gates required further, the OR gate in Fig. 20 a can be modelled using NAND gates as follows





**Figure 20:** Full adder symbol

In order to simulate the full adder circuit, a simulation file (CMD file) was formed. This was based according to the full adder truth table (Table 3). The CMD file was executed through the *Viewsim* program provided by *Viewlogic* software. The CMD file for the full adder is shown in Table 4.

```

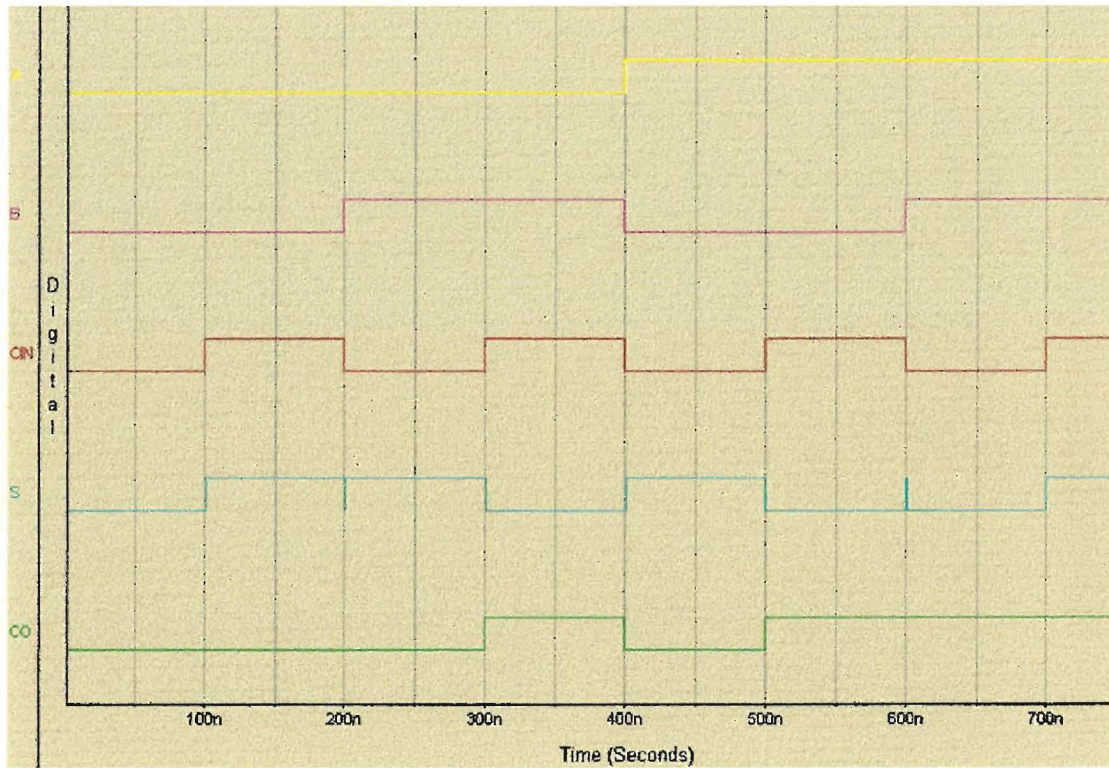
ECHO FINAL YEAR PROJECT
ECHO Digital Filter Implementation on FPGA
ECHO Full adder simulation file
ECHO 16/11/97
ECHO By K.Deliparaschos
|
RESTART
|
PAT CIN 0 1 0 1 0 1 0 1
PAT B 0 0 1 1 0 0 1 1
PAT A 0 0 0 0 1 1 1 1
|
WAVE FA_NAND.WFM A B CIN S CO
|
CYCLE 20

```

**Table 4:** Full adder CMD file

The simulation results were plotted using *Viewtrace* program part of *Viewlogic* software. The simulation results are shown in Fig. 21.

From the simulation results it is shown that the full adder functions properly. This can be also verified if the simulation results are compared with the truth table (Table c). The two glitches (200psec) appearing in the sum output are not considered to be critical.



**Figure 21:** Full adder simulation results

### 4.3 AND gated Full Adder design

The implementation of the radix- $2^n$  arithmetic cell (Fig. 6) could be performed by an AND gated full adder. The AND gated full adder consists of an AND gate connected to the most significant input of a full adder. In the present situation, the full adder designed in section 5.2 could be used.

Figure 22 shows the implementation of the AND gated full adder

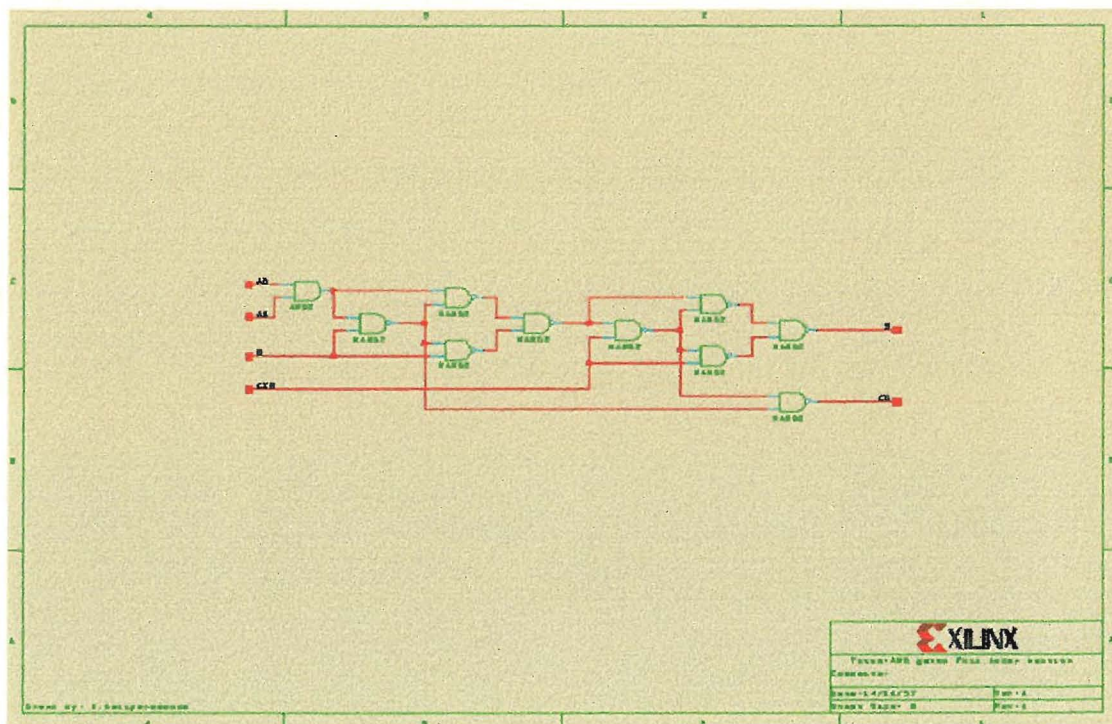
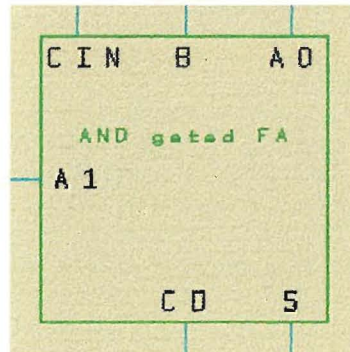


Figure 22: AND gated Full adder schematic

Where A0 and A1 represent  $V_j$  and  $U_i$  of the radix- $2^n$  arithmetic cell respectively in Fig. 6. The total number of gates required is 12.

The design of Fig. 22 was created as a symbol for later use in the implementation of the filter. Fig 23 shows the symbol for the AND gated full adder.



**Figure 23:** And gated full adder symbol

The AND gated full adder was not simulated individually for obvious reasons.

#### 4.4 Carry Save Adder design

To add a sequence of numbers, several full adders are connected. This can be achieved in more than one ways, remembering that a full adder receives three inputs (3-bit) of equal and produces two outputs.

- (a) A sum of the same significance as the inputs
- (b) A carry of double the significance of the inputs.

The carry save adder (CSA) shown on the next page consists of four full adders (Fig. 24.)

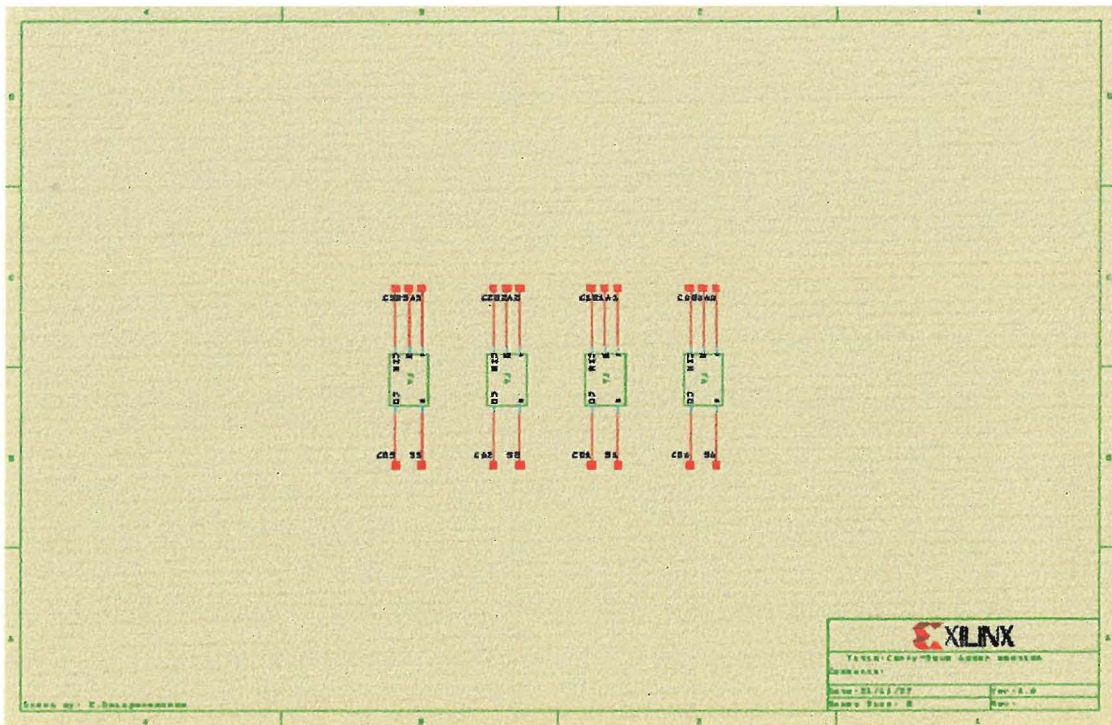


Figure 24: CSA schematic

Figure 25 illustrates the symbol created for the carry save adder.

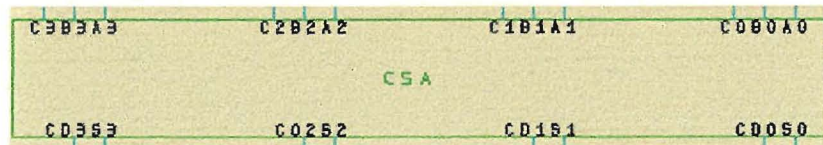


Figure 25: CSA symbol



## 4.5 4-bit Register design

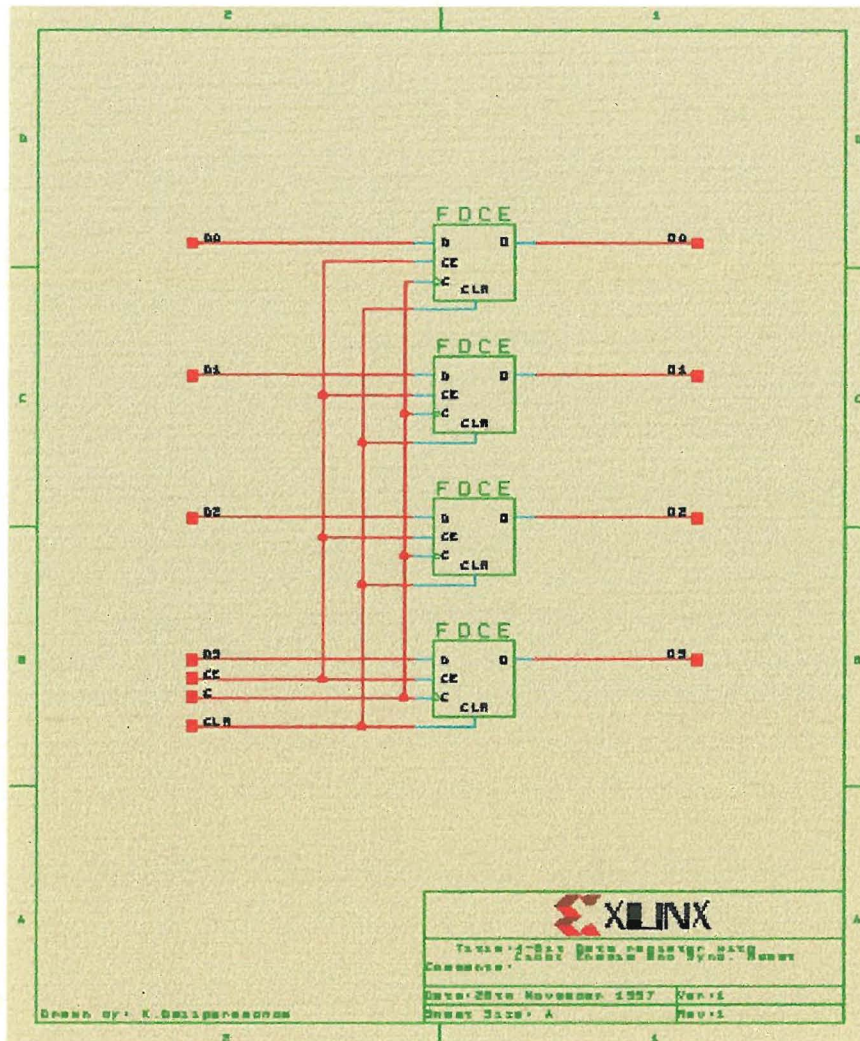
The most significant digits of the multiplication coming out from the CSAs in the digit serial multiplier cell need to be delayed by one cycle. Each digit consists of four bits, hence a 4-bit register with parallel input and parallel output is required to delay them by one cycle. As all of the four flip-flops need to be operating at the same time, their clock inputs need to be connected together as well as their clock enable and clear inputs.

In order to design the 4-bit register a D flip-flop with clock enable and asynchronous clear (FDCE) was chosen from the XC3000 library in *Viewlogic* software. Table 5 shows the transition table of FDCE.

Inputs				Outputs
CLR	CE	D	C	Q
1	X	X	X	0
0	0	X	X	No Change
0	1	1	↑	1
0	1	0	↑	0

**Table 5:** Transition table of FDCE

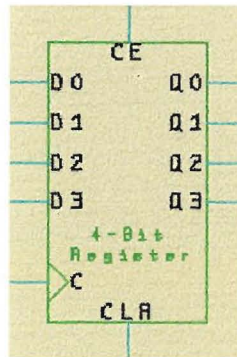
When clock enable (CE) is High, and asynchronous clear (CLR) is Low, the data on the data input (D) of FDCE is transferred to the corresponding data output (Q) during the Low-to-High clock transition. When CLR is High, it overrides all other inputs and resets the data output (Q) Low. When CE is Low, clock transitions are ignored. The schematic of the 4-Bit register is shown on the page overleaf (Fig. 26).



**Figure 26:** 4-Bit Register schematic

Where  $D_0, \dots, D_3$  and  $O_0, \dots, O_3$  are the inputs and outputs of the register respectively. CE and CLR are the clock enable and clear inputs and C is the clock input of the register.

The symbol created for the 4-bit register circuit is shown in Fig. 27 on next page.



**Figure 27:** 4-Bit Register symbol

## 4.6 Digit-Serial Multiplier Design and Simulation

The digit-serial multiplier was implemented according to figure 9 in section 3 using the symbols previously created in this chapter (Fig. 28).

The multiplicand and multiplier inputs were labelled  $B_3, \dots, B_0$  and  $Y_3, \dots, Y_0$  respectively (0 indicates the LSB). Outputs  $BYO_{00}$ ,  $BYO_{01}$ ,  $BYO_{10}, \dots, BYO_{03}$  represent the LSDs of the multiplication process to be fed in to  $BYIN_{00}$ ,  $BYIN_{01}$ ,  $BYIN_{10}, \dots, BYIN_{03}$  inputs in the following cell on the right. Outputs  $BYO_{10}$  and  $BYO_{01}$  are not computed in the present cell but sent to be computed to the empty bit position on the following cell on the right ( $BYIN_{01}$ ,  $BYIN_{10}$  inputs). The MSDs of the multiplication are presented in  $S1O[3:0], \dots, S3O[2:0]$  output buses delayed by one cycle to be fed in the following cell on the right ( $S1IN[3:0], \dots, S3IN[2:0]$  input buses). Output buses  $S1O[3:0], \dots, S3O[2:0]$  need to be added together to form the MSDs of the multiplication. Inputs  $CE$ ,  $CLR$ ,  $CLK$  represent the clock enable, clear input, and clock input of the registers respectively.

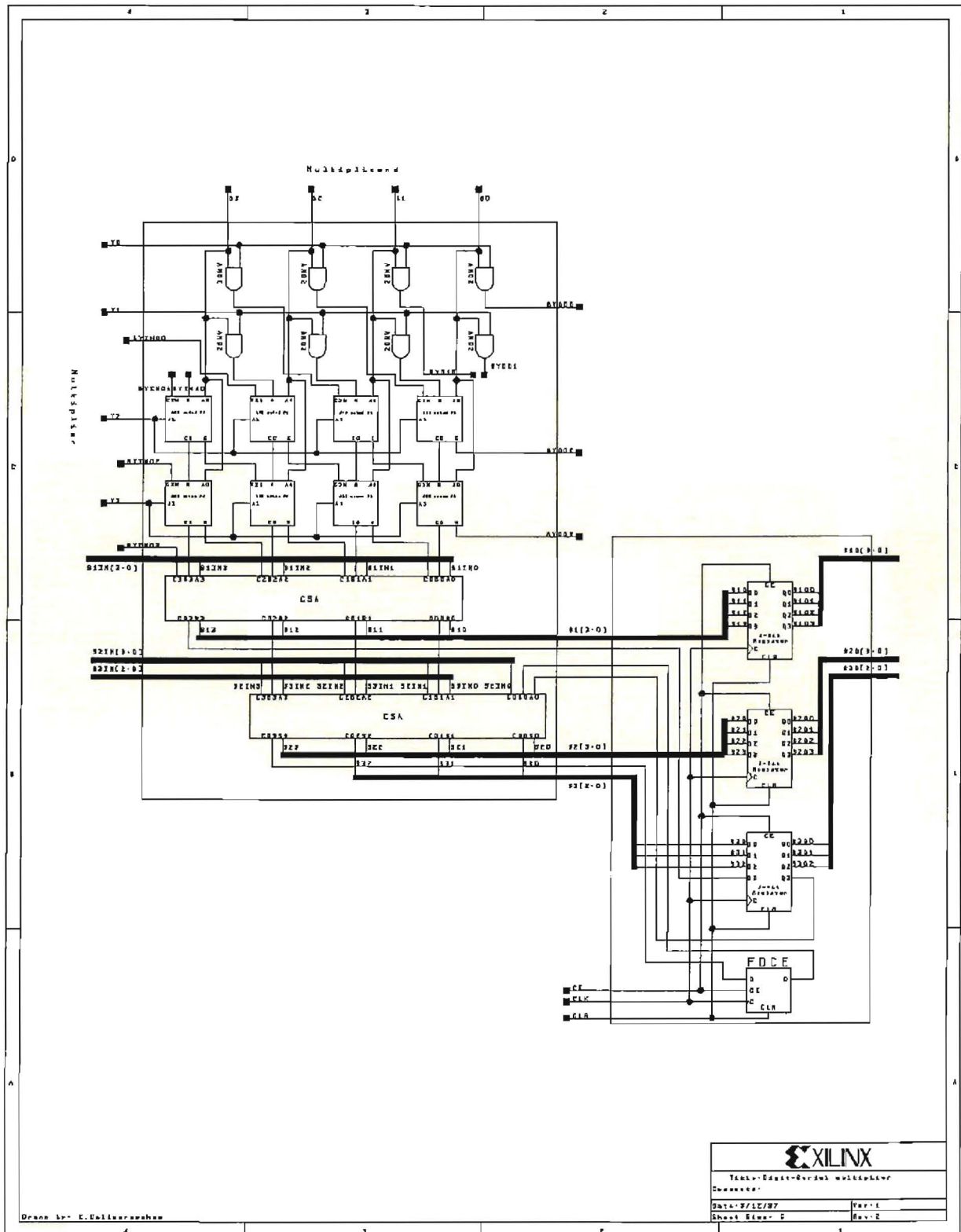
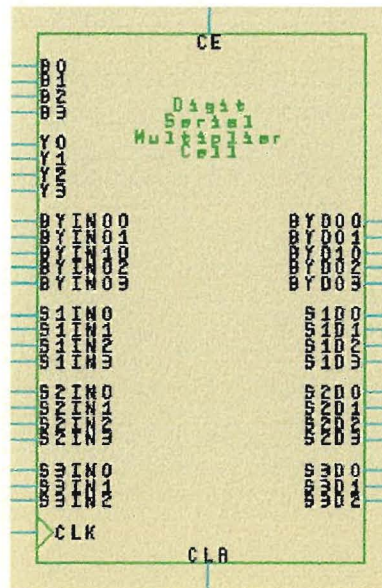


Figure 28: Digit-Serial multiplier

A symbol created for the schematic of the digit-serial multiplier cell to be used later in the implementation (Fig. 29).



**Figure 29:** Digit-Serial multiplier symbol

In order to simulate the digit-serial multiplier a CMD file was written and executed (Table 6). Two arbitrary numbers were chosen to be multiplied together in order to make sure that the digit-serial multiplier produces the correct result. The multiplicand was set equal to  $11_{10}$  or  $1011_2$  and the multiplier equal to  $13_{10}$  or  $1101_2$ . The product of those two numbers is  $143_{10}$  or  $\overbrace{10001111}^{MSD \quad LSD}_2$ . All the other inputs at the left hand side of the cell were set to zero since the multiplier was tested on its own without any data coming through from previous cells. The clear input set to 50ns in order to reset the registers before the multiplication process begins.

Below is the CMD file for the digit-serial multiplier

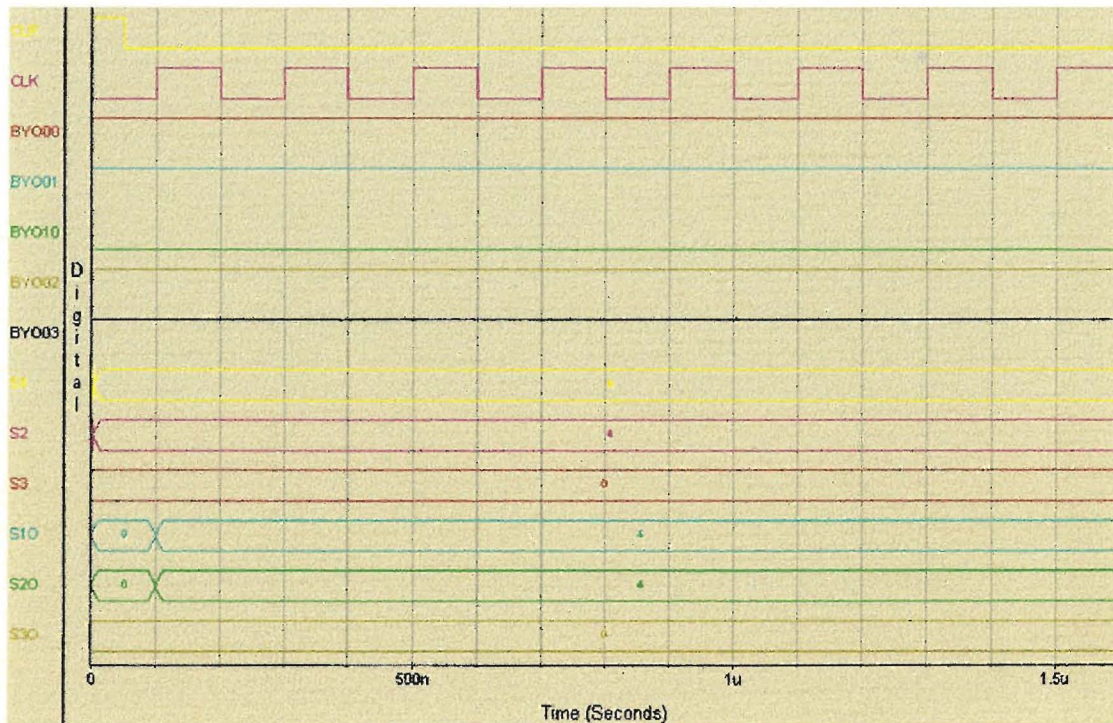
```

ECHO FINAL YEAR PROJECT
ECHO Digital Filter Implementation on FPGA
ECHO Digit-Serial Multiplier Simulation File
ECHO 12/12/97
ECHO BY K.Deliparaschos
|
RESTART
VECTOR S1IN S1IN[3:0]
VECTOR S2IN S2IN[3:0]
VECTOR S3IN S3IN[2:0]
|
VECTOR S1 S1[3:0]
VECTOR S2 S2[3:0]
VECTOR S3 S3[2:0]
|
VECTOR S10 S10[3:0]
VECTOR S20 S20[3:0]
VECTOR S30 S30[2:0]
|
WFM CE @0NS=1
WFM CLR @0NS=1 @50NS=0
|
WFM BYIN00 @0=0
WFM BYIN01 @0=0
WFM BYIN10 @0=0
WFM BYIN02 @0=0
WFM BYIN03 @0=0
|
|MULTIPLICAND
WFM B0 @0=1
WFM B1 @0=0
WFM B2 @0=1
WFM B3 @0=1
|
|MULTIPLIER
WFM Y0 @0=1
WFM Y1 @0=1
WFM Y2 @0=0
WFM Y3 @0=1
|
PATTERN S1IN 0\H 0\H 0\H 0\H 0\H 0\H 0\H 0\H
PATTERN S2IN 0\H 0\H 0\H 0\H 0\H 0\H 0\H 0\H
PATTERN S3IN 0\H 0\H 0\H 0\H 0\H 0\H 0\H 0\H
|
CLOCK CLK 0 1
WAVE MUL_REG.WFM CLR CLK BY00 BY01 BY10 BY02 BY03 S1 S2 S3 S10 S20 S30
|
CYCLE 8

```

**Table 6:** Digit-Serial multiplier CMD file

The simulation results for the digit-serial multiplier are shown underneath (Fig. 30). The present simulation could also prove the correct functionality of the 4-bit register designed in section 4.5.



**Figure 30:** Digit-Serial multiplier simulation results

By inspecting the simulation results, the LSD of the multiplication result was checked to be correct ( $1111_2$ ), after adding together BYO01 and BYO10 bits. In order to check the MSD as well, output busses S10, S20, S30 should be added together. Therefore  $MSD = S1 + S2 + S3$  or  $MSD = 4_{16} + 4_{16} + 0 = 8_{16} = 1000_2$ , which is the correct result. The same procedure was repeated for different couples of arbitrary chosen number and in all cases the digit-serial multiplier was found to function properly.

## 4.7 Carry Ripple Adder Design

The present carry ripple adder (CRA) consists of four full adders and is shown in Fig 31. The carry-out bit of the first full adder on the right is propagating to the next full adder, to be added with the other two bits and so on.

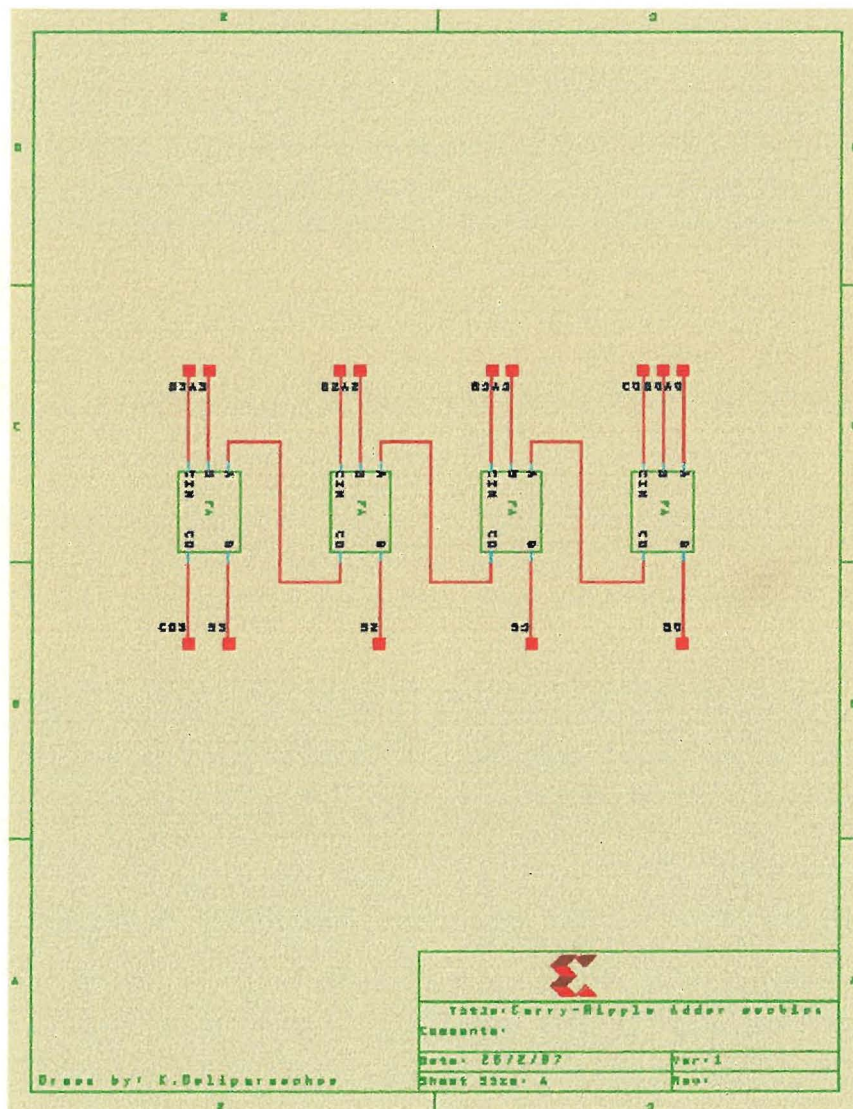
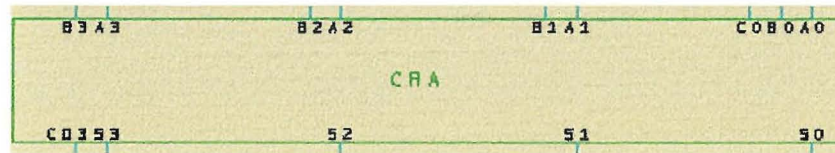


Figure 31: CRA schematic



Below is shown the symbol created for the CRA (Fig. 32)



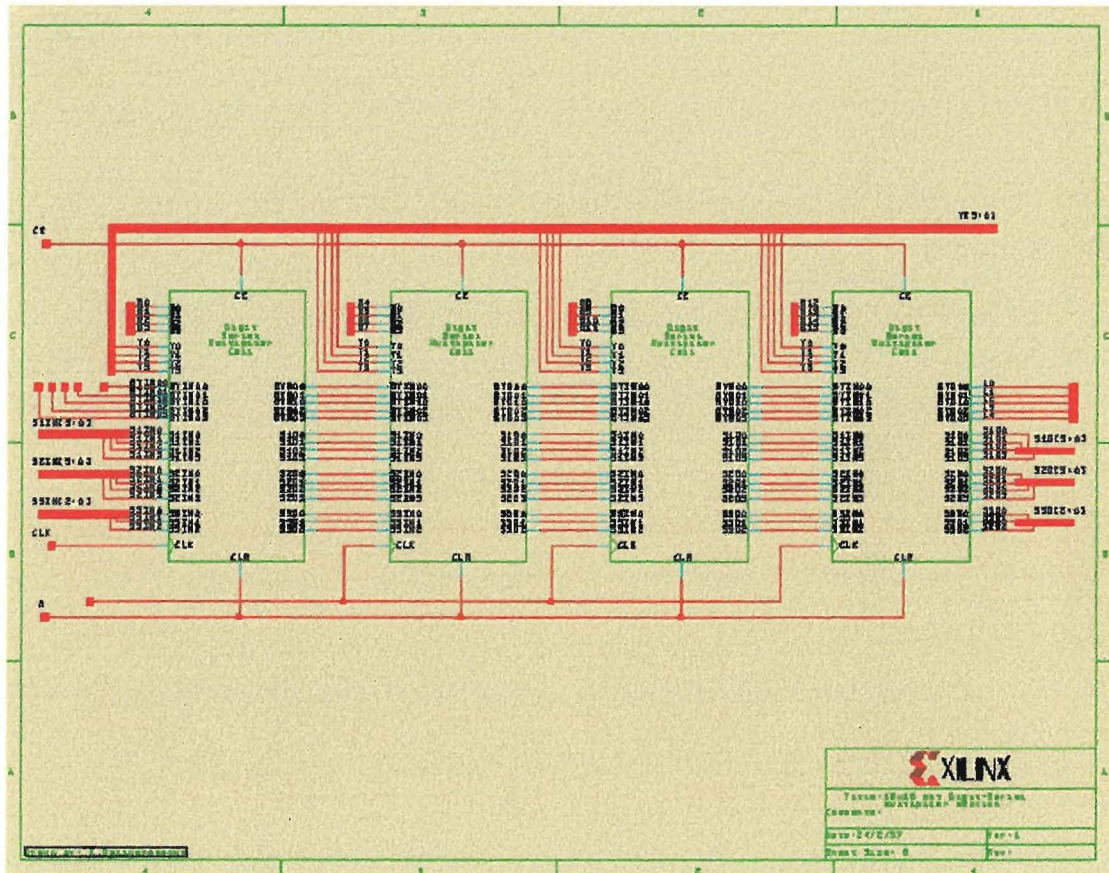
**Figure 32:** CRA symbol

## **4.8 16x16 bit Digit-Serial Multiplier with Digit-Serial Adder Design and Simulation**

Table 7 on the following page demonstrates the manual multiplication of two 16-bit numbers producing a 32-bit result.



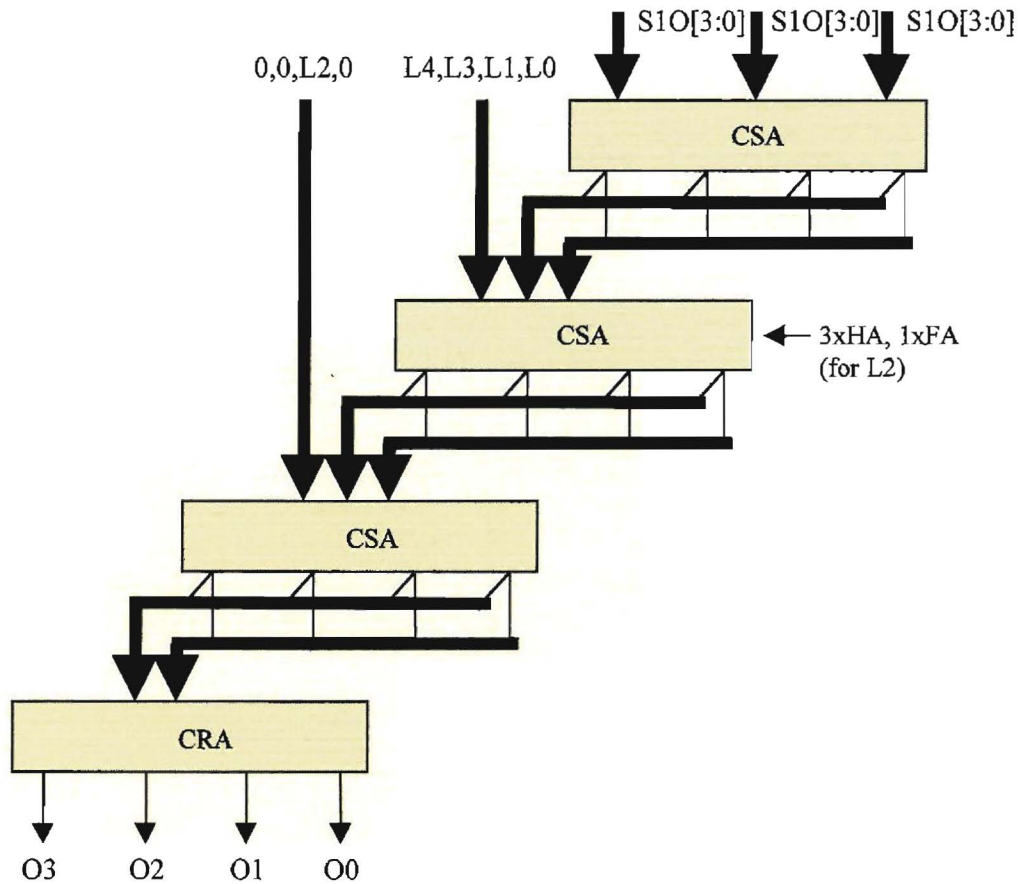
At this stage four digit-serial multiplier cells (discussed previously in section 4.6, and shown in Fig. 28) were cascaded together in order to form a 16x16 bit digit-serial multiplier (Fig. 33).



**Figure 33:** 16x16 bit Digit-Serial multiplier

The LSDs and MSDs produced from the two 16-bit multiplied numbers need to be added (every cycle) to give the product of the multiplication, which is a 32-bit result. The result of the addition must be 4-bit long at each cycle, for 8 cycles ( $8 \times 4\text{-bit} = 32\text{-bit}$ ). The addition discussed above could be obtained with a digit-serial adder. The last could be formed by a network of three CSAs and one CRA appropriately connected.

The block diagram underneath illustrates the suggested method to form the digit-serial adder (Fig. 34).



**Figure 34:** Digit-Serial Adder block diagram

The third CSA starting from the top of the diagram, only requires three HAs and one FA instead of four FAs. The reason that only one FA is required is because only one spare bit position is needed, for L2 and since the other three bits are always zero (not need to be added) HAs can be used instead. The carry output of the most significant FA in the CSA must be delayed by one cycle, before fed to least significant FA in the CSA underneath and so on. Finally the most significant carry output of the CRA

should be delayed by one cycle and fed back to the least significant FA. Since a number of four registers are required, a 4-bit register could be used.

A 4-bit register was connected at the outputs ( $S_0, \dots, S_3$ ) of the digit-serial adder, in order to allow the multiplier time to finish the computation at each cycle.

Figure 35 on page overleaf shows the 16x16 bit digit-serial multiplier together with the digit-serial adder, previously described.

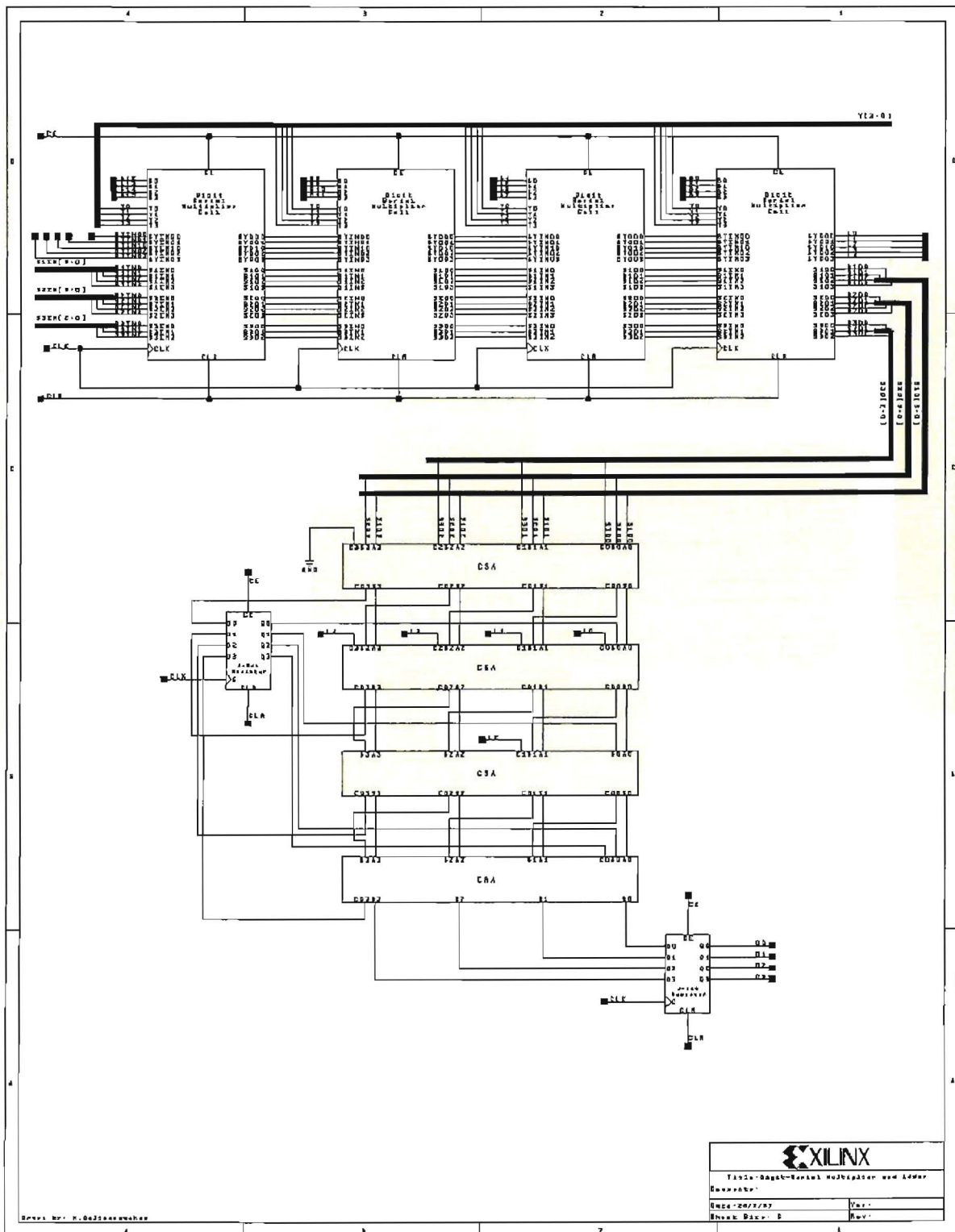
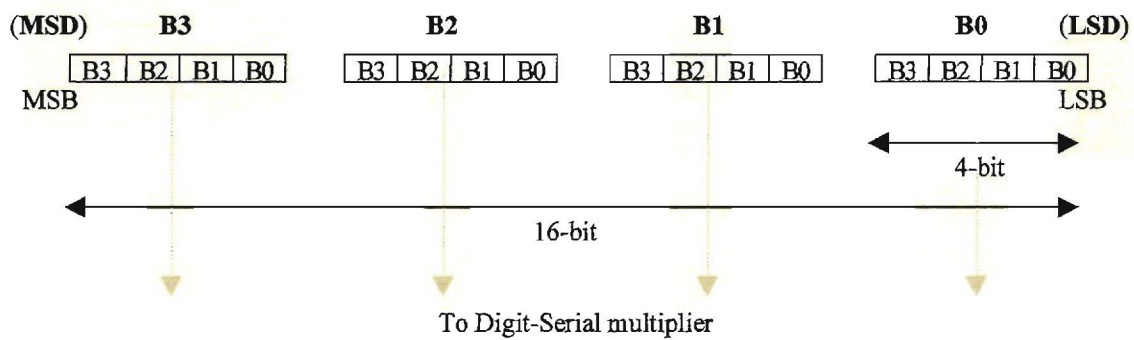


Figure 35: 16x16 bit Digit-Serial multiplier and Digit-Serial Adder

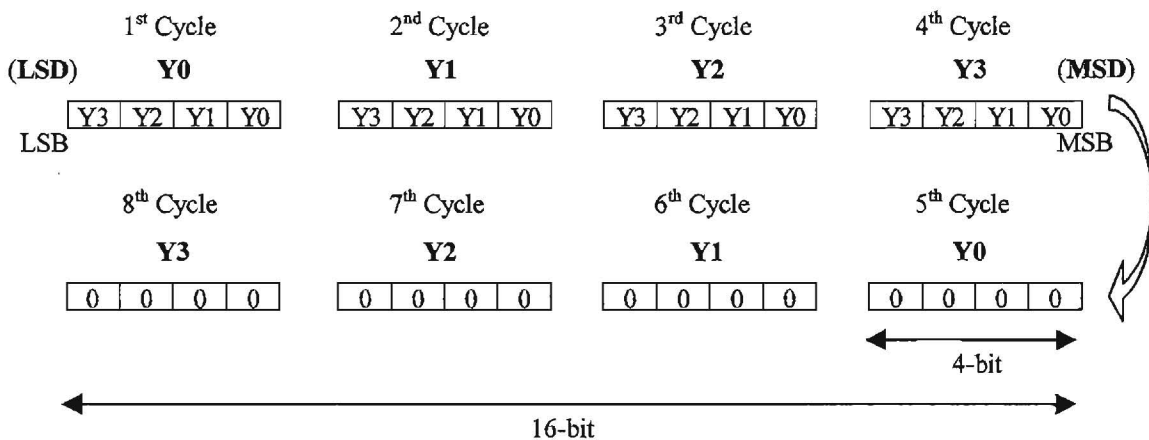
The digit-serial multiplier needs 8 cycles in order to compute a 16x16 bit = 32-bit result. The serial data (16-bit length) are divided to 4 digits of 4-bit each digit. After the first 4 cycles (16-bit result out of 32-bit), 0s must be fed to the serial input (16-bit length) for the next 4 cycles, to allow the multiplier to finish the computation.

The following diagram illustrates the structure of parallel data (multiplicand) and serial data (multiplier) fed in to the digit-serial multiplier (Table 8).

**Parallel Data  
(Multiplicand)**



**Serial Data  
(Multiplier)**



**Table 8:** Structure diagram of Parallel and Serial data

In order to be able to simulate the digit-serial multiplier (Fig. 35) a CMD file was written (Table 9). Again two arbitrary 16-bit numbers were chosen for the multiplicand and the multiplier. The multiplicand (B) was set to  $45123_{10}$  or  $1011000001000011_2$  and the (Y) multiplier was set to  $57000_{10}$  or  $\underbrace{1101111010101000}_{\substack{D \\ E \\ A \\ 8 \\ 2}}$ . The product of the current multiplication is  $2572011000_{10}$  or

$\underbrace{1001100101001101110001011111000}_{\substack{9 \\ 9 \\ 4 \\ D \\ C \\ 5 \\ F \\ 8 \\ 2}}$ .

```
ECHO FINAL YEAR PROJECT
ECHO Digital Filter Implementation on FPGA
ECHO 16x16 bit digit-serial multiplier+adder Simulation File
ECHO 26/2/97
ECHO BY K.Deliparaschos
|
RESTART
VECTOR S1IN S1IN[3:0]
VECTOR S2IN S2IN[3:0]
VECTOR S3IN S3IN[2:0]
|
VECTOR S1O S1O[3:0]
VECTOR S2O S2O[3:0]
VECTOR S3O S3O[2:0]
|
VECTOR Y Y[3:0]
|
CLOCK CLK 0 1
|
WFM CE @0NS=1
WFM CLR @0NS=1 @50NS=0
|
WFM BYIN00 @0=0
WFM BYIN01 @0=0
WFM BYIN10 @0=0
WFM BYIN02 @0=0
WFM BYIN03 @0=0
|
|MULTIPLICAND
WFM B0 @0=1
WFM B1 @0=1
WFM B2 @0=0
WFM B3 @0=0
WFM B4 @0=0
WFM B5 @0=0
WFM B6 @0=1
WFM B7 @0=0
WFM B8 @0=0
WFM B9 @0=0
```

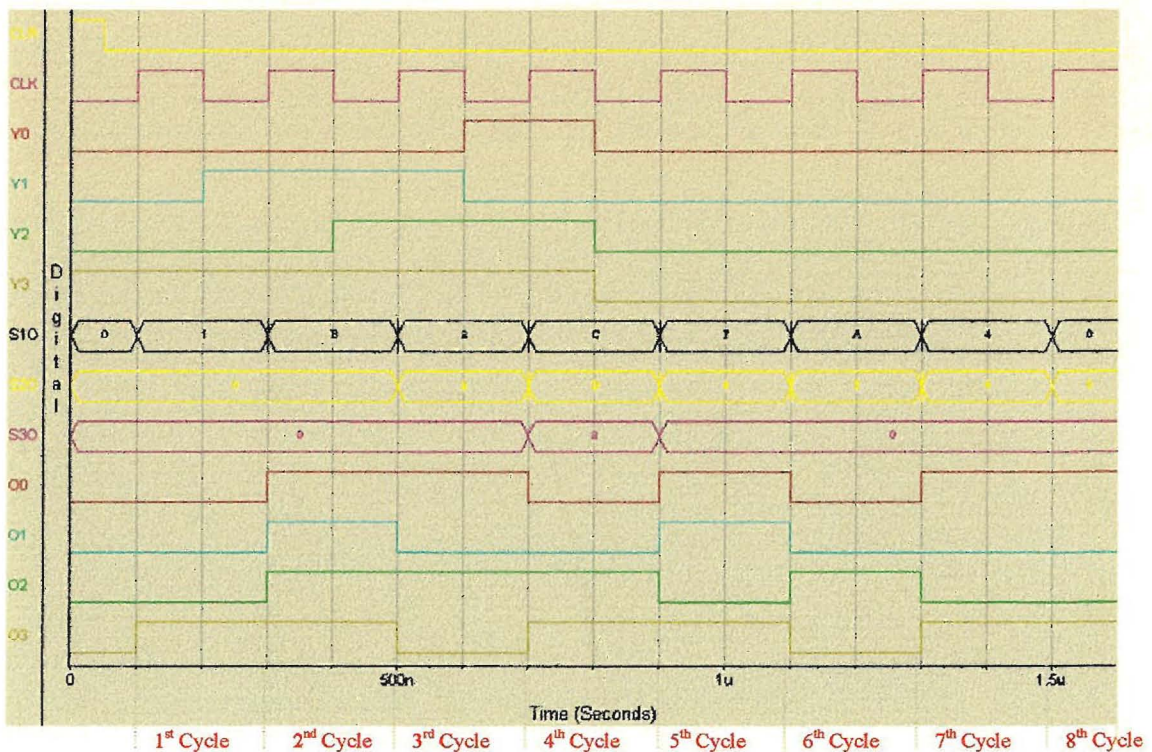


```

WFM B10 @0=0
WFM B11 @0=0
WFM B12 @0=1
WFM B13 @0=1
WFM B14 @0=0
WFM B15 @0=1
|
|MULTIPLIER
PATTERN Y 8\H A\H E\H D\H 0\H 0\H 0\H 0\H
|
PATTERN S1IN 0\H 0\H 0\H 0\H 0\H 0\H 0\H 0\H
PATTERN S2IN 0\H 0\H 0\H 0\H 0\H 0\H 0\H 0\H
PATTERN S3IN 0\H 0\H 0\H 0\H 0\H 0\H 0\H 0\H
|
WAVE 16bitaa.wfm CLR CLK Y0 Y1 Y2 Y3 S10 S20 S30 O0 O1 O2 O3
|
CYCLE 8
    
```

**Table 9:** CMD file for 16x16 bit Digit-Serial multiplier

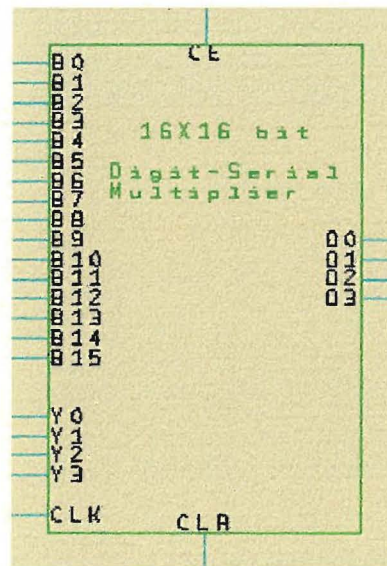
Below are the results occurred from the simulation of the 16x16 bit digit-serial multiplier (Fig. 36).



**Figure 36:** Simulation result for the 16x16 bit Digit-Serial multiplier

The 100 ns of delay at the outputs, O0,...,O4 of the digit-serial multiplier are caused due to the existence of the 4-bit register at the output. From the simulation results in Fig. 36, is shown that the correct result of the multiplication were obtained after 8 cycles. The same method of simulation was carried out a number of times using different pairs of numbers each time.

Finally the schematic of figure 35 was created as a symbol for later use (Fig. 37)



**Figure 37:** 16x16 bit digit-serial multiplier with digit-serial adder symbol

Where B0,...,B15 are the parallel data inputs (multiplicand), Y0,...,Y3 are the serial data inputs (multiplier) and O0,...,O3 are the outputs (in serial form).

## 4.9 16x16 bit Digit-Serial Multiplier and Digit-Serial Adder with Shift Registers Design and Simulation

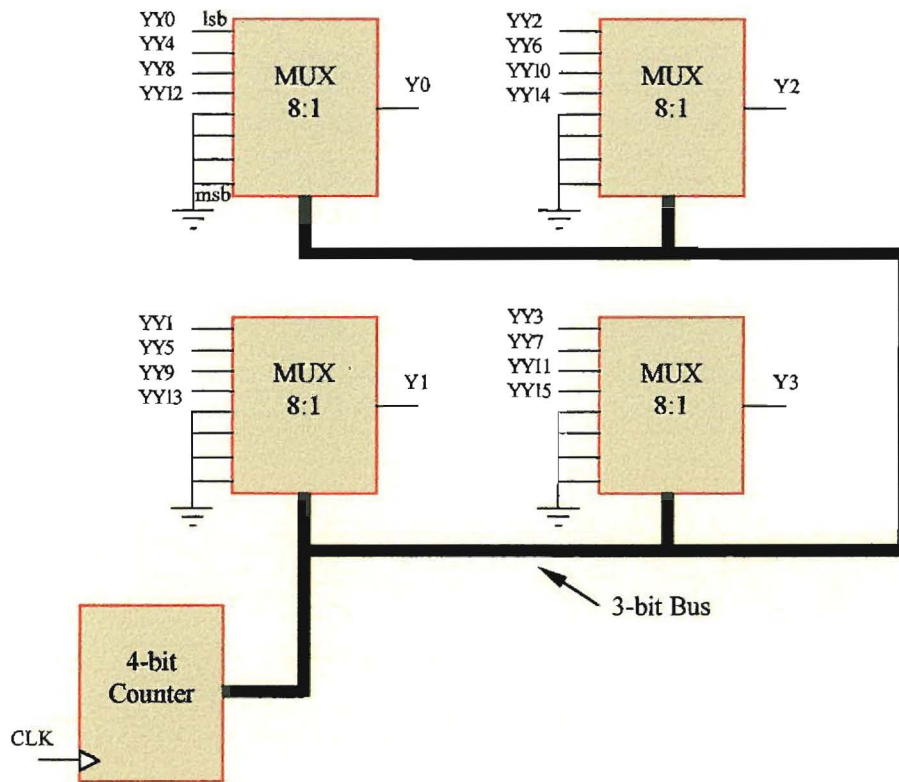
If the 16x16 bit digit-serial multiplier is to be downloaded on an FPGA and tested in real time, some sort of hardware is needed at the input to feed the serial data in, and at the output to store the result of the multiplication after this is available (8 cycles of computation time). Obviously this was not required till now, since a CMD file was used and the simulation was performed through *Viewlogic* software.

The serial data input can be created by using a 16-bit parallel-in to 4-bit serial-out shift register. The recommended method has the advantage of allowing to change the multiplier number (16-bit therefore 16 pins required) by just altering pins. At the output of the digit-serial multiplier a 4-bit serial-in to 16-bit parallel-out shift register could be used, to store the results of computation each cycle for 8 cycles. After the results have been stored, can be checked by simply observing the outputs of the register (using an oscilloscope).

### 4.9.1 Parallel-in to Serial-out Shift Register Design

The parallel-in to serial-out shift register is using four 8:1 multiplexers (ULM) having the four MSB front inputs connected to ground (part number *M8\_1E* available from *Xilinx XC3000* library). This arrangement enables 0s to be fed in to the serial input of the digit-serial multiplier after the first 4 cycles have finished. The control lines of the ULMs are connected to a 4-bit binary counter, which is utilised as a 3-bit counter

(part number *CB4CE* available from *Xilinx XC3000* library). The counter is clocked at each cycle to produce the serial data. The block diagram of the parallel-in to serial-out shift register described above is shown in figure 38 below.

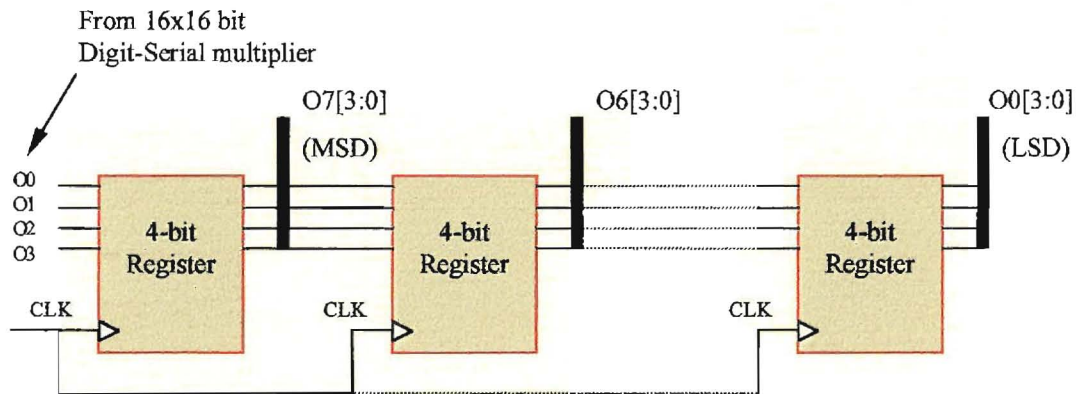


**Figure 38:** Parallel-in to Serial-out shift register block diagram

During the first cycle the inputs  $YY_0, \dots, YY_3$  are selected to form the LSD which appears at the outputs  $Y_0, \dots, Y_3$  of the ULMs. The same procedure is repeated for all 8 cycles.

## 4.9.2 Serial-in to Parallel-out Shift Register Design

The serial-in to parallel-out shift register must be able to store one digit (4-bit) every cycle for a total of 8 cycles (Fig. 39). This could be simply achieved by using a network of eight 4-bit registers (section 4.5) to store the data after each cycle. The LSD of the multiplication is first stored in the most significant 4-bit register. When the next digit arrives replaces the previous one, which is shifted from left to right to the next 4-bit register and so on.



**Figure 39:** Serial-in to Parallel-out shift register block diagram

Finally the 16x16 bit digit-serial multiplier together with the shift registers (described in sections 4.8.1 and 4.8.2) was captured using *Viewdraw* and is shown on next page (Fig. 40).

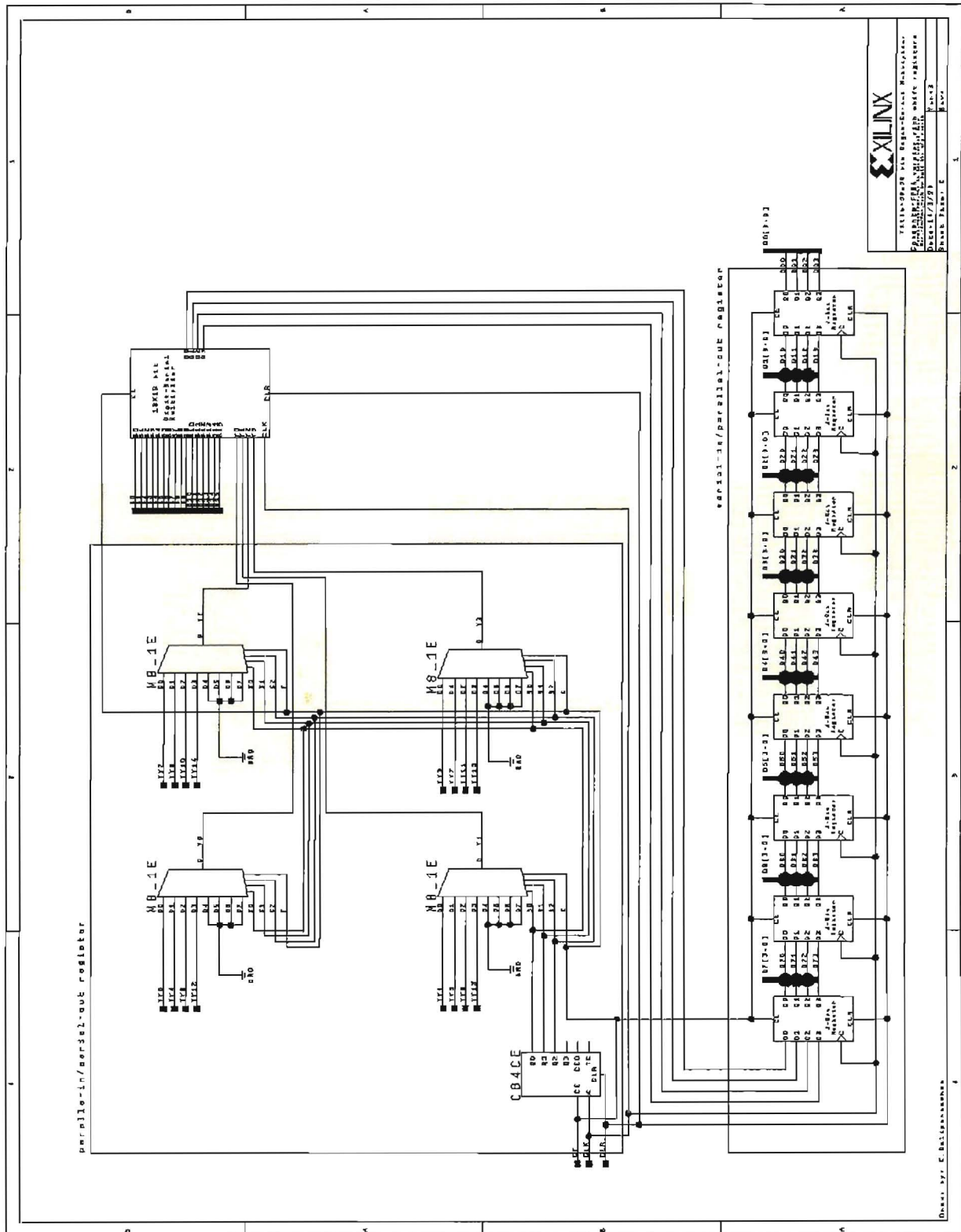


Figure 40: 16x16 bit Digit-Serial multiplier with shift registers

A CMD file (Table 10) was written to define the numbers under multiplication. The multiplicand (B) was set to  $45123_{10}$  or  $1011000001000011_2$  and the (Y) multiplier was set to  $57000_{10}$  or  $\underbrace{11011}_{D}\underbrace{111010101000}_{E}$ . The product of the current multiplication is  $2572011000_{10}$  or  $\underbrace{1001100101001101110001011111000}_{9\ 9\ 4\ D\ C\ 5\ F\ 8\ 2}$ .

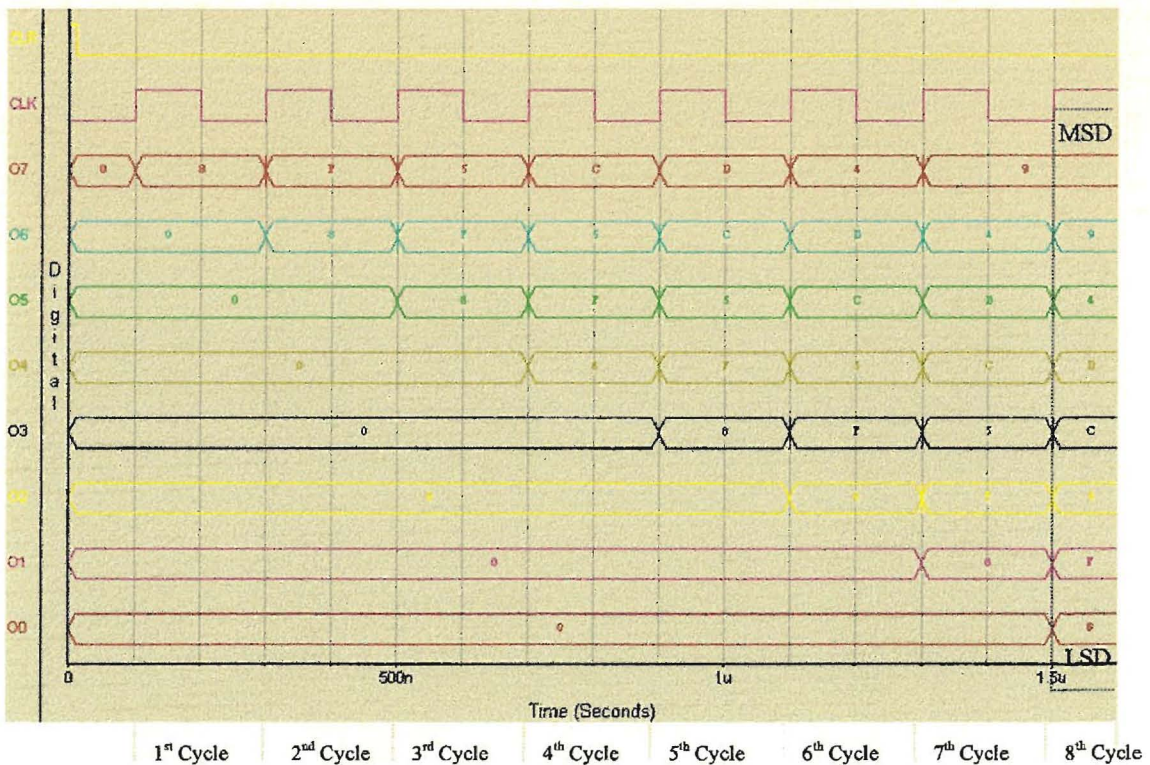
```
ECHO FINAL YEAR PROJECT
ECHO Digital Filter Implementation on FPGA
ECHO 16X16 bit digit-serial multiplier
ECHO FPGA version with shift registers
ECHO 13/3/98
ECHO by K.DELIPARASCHOS
|
RESTART
VECTOR O0 O0[3:0]
VECTOR O1 O1[3:0]
VECTOR O2 O2[3:0]
VECTOR O3 O3[3:0]
VECTOR O4 O4[3:0]
VECTOR O5 O5[3:0]
VECTOR O6 O6[3:0]
VECTOR O7 O7[3:0]
|
WFM CE @ONS=1
WFM CLR @ONS=1 1ONS=0
|
|MULTIPLIER
WFM YY0 @0=0
WFM YY1 @0=0
WFM YY2 @0=0
WFM YY3 @0=1
|
WFM YY4 @0=0
WFM YY5 @0=1
WFM YY6 @0=0
WFM YY7 @0=1
|
WFM YY8 @0=0
WFM YY9 @0=1
WFM YY10 @0=1
WFM YY11 @0=1
|
WFM YY12 @0=1
WFM YY13 @0=0
WFM YY14 @0=1
WFM YY15 @0=1
|
|MULTIPLICAND
WFM B0 @0=1
WFM B1 @0=1
WFM B2 @0=0
```

```

WFM B3 @0=0
WFM B4 @0=0
WFM B5 @0=0
WFM B6 @0=1
WFM B7 @0=0
WFM B8 @0=0
WFM B9 @0=0
WFM B10 @0=0
WFM B11 @0=0
WFM B12 @0=1
WFM B13 @0=1
WFM B14 @0=0
WFM B15 @0=1
|
CLOCK CLK 0 1
WAVE FPGAIMP2.WFM CLR CLK 07 06 05 04 03 02 01 00
CYCLE 8
    
```

**Table 10:** CMD file for 16x16 bit Digit-Serial multiplier with shift registers

The results of the simulation (plotted in *Viewtrace*) appear underneath in Fig. 41.



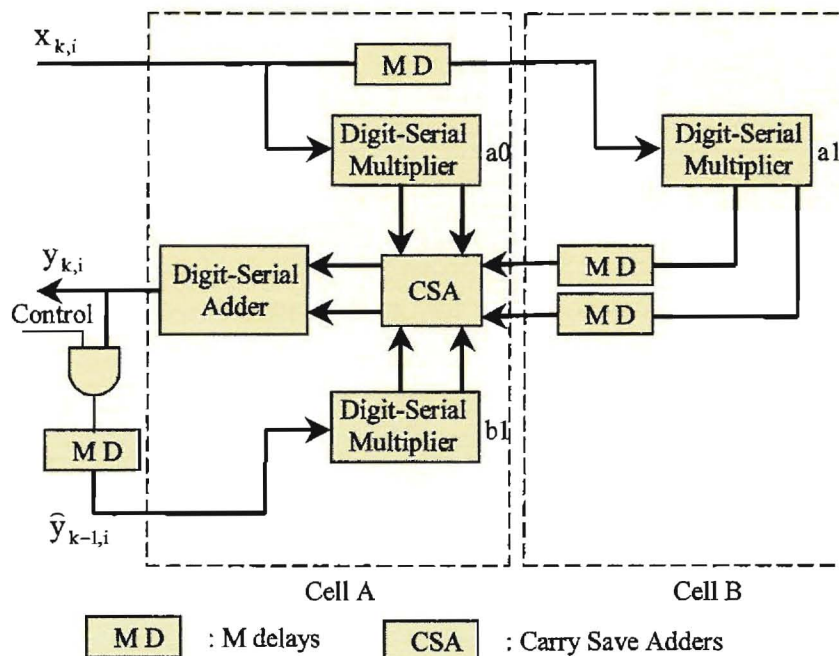
**Figure 41:** Simulation results for 16x16 bit digit-serial multiplier with shift registers



The simulation in Fig. 41 produced the correct results. The same procedure was carried out again with different pair of numbers, and produced satisfied results in all cases.

#### 4.10 1<sup>st</sup> order Digit-Serial IIR Filter Design and Simulation

The block diagram of figure 11 was reduced down from a 2<sup>nd</sup> order to a 1<sup>st</sup> order digit-serial IIR filter. The new block diagram of the 1<sup>st</sup> order digit-serial IIR filter is illustrated below (Fig. 42).

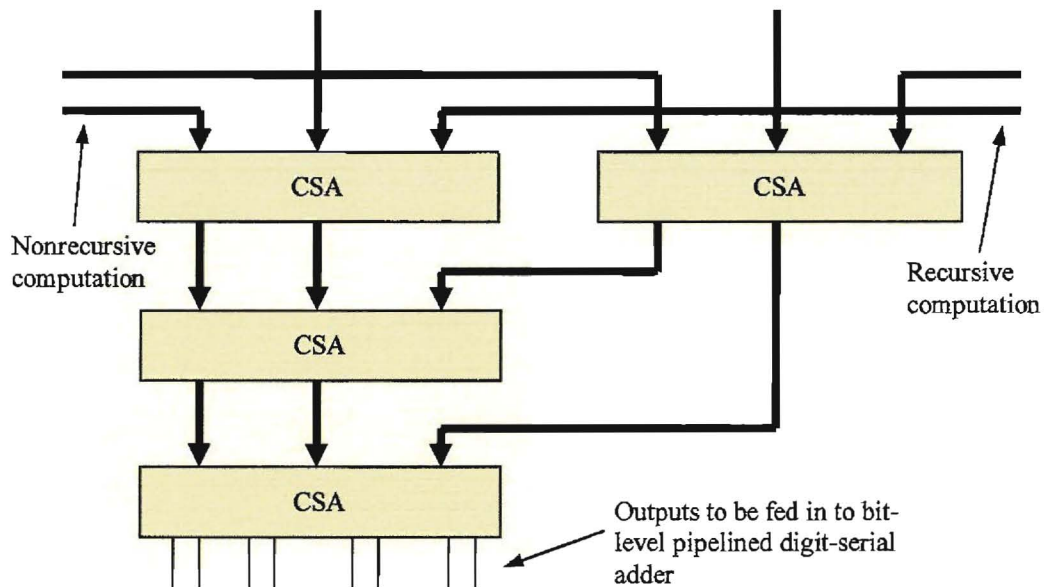


**Figure 42:** Block diagram of a 1<sup>st</sup> order digit-serial IIR filter (M=4)

Before the 1<sup>st</sup> order filter can be implemented and tested, the CSAs array and bit-level pipelined digit-serial adder should be implemented.

### 4.10.1 Carry Save Adders Array Design

The carry save adders (CSAs) array shown in the block diagram of Fig. 43 was implemented according to Fig.11, previously discussed in chapter 3.



**Figure 43:** Block diagram of CSAs array

The schematic of the CSAs array was captured using *Viewdraw* (Fig.42).

Where,

NS[3:0], NC[3:0]: Nonrecursive sum and carry 4-bit input bus

RS[3:0], RC[3:0]: Recursive sum and carry 4-bit input bus

FS[3:0], FC[3:0]: Sum and Carry 4-bit input bus from following cells

BS[3:0], BC[3:0]: Sum and Carry 4-bit bus

C0,...,C3 and S0,...,S3: Carry and sum output

A 4-bit register was used to delay the most significant carry bit, before fed to the next CSA.

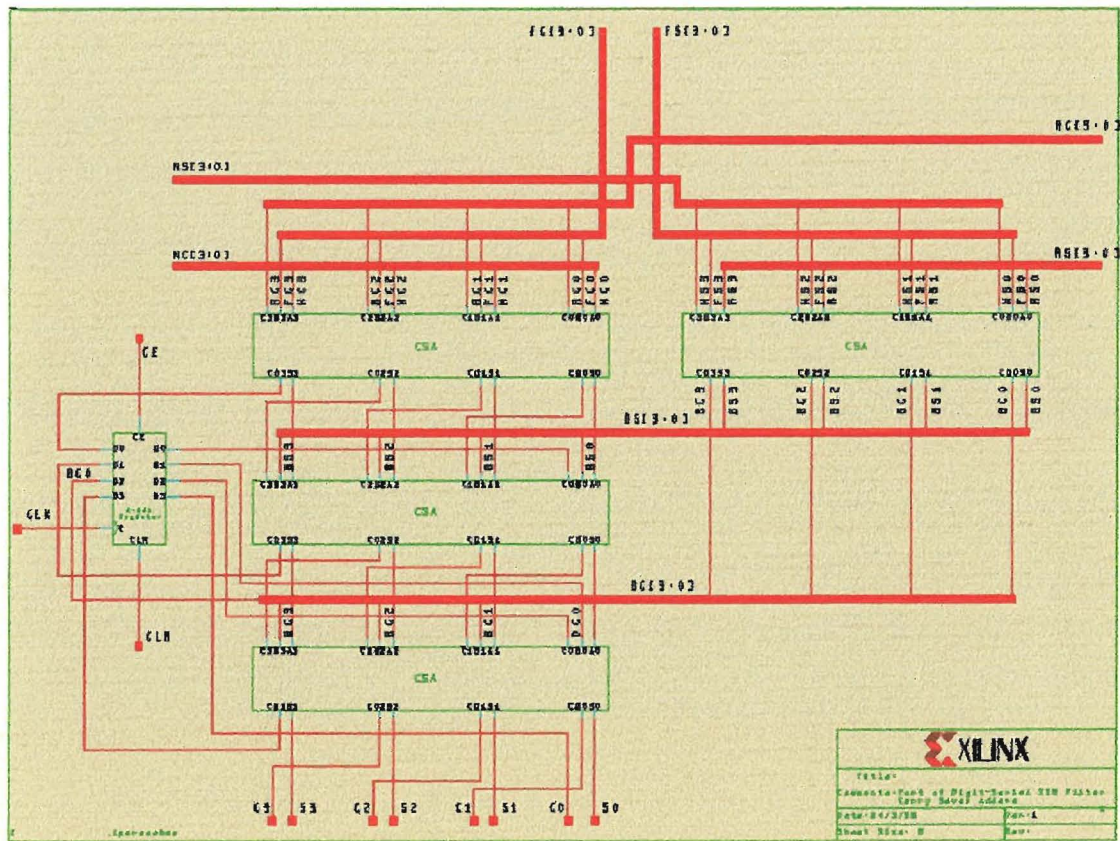


Figure 44: Schematic of CSAs array

The CSAs array was created as a symbol for later use (Fig. 45).

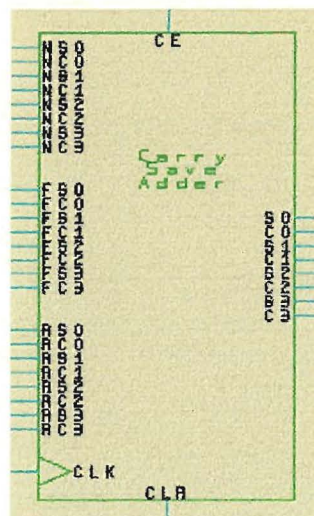


Figure 45: Symbol for CSAs array

### 4.10.2 Bit-level Pipelined Digit-Serial Adder Design and Simulation

The schematic of the bit-level pipelined digit serial adder was implemented according to Fig. 12 in chapter 3 and is shown underneath (Fig. 46).

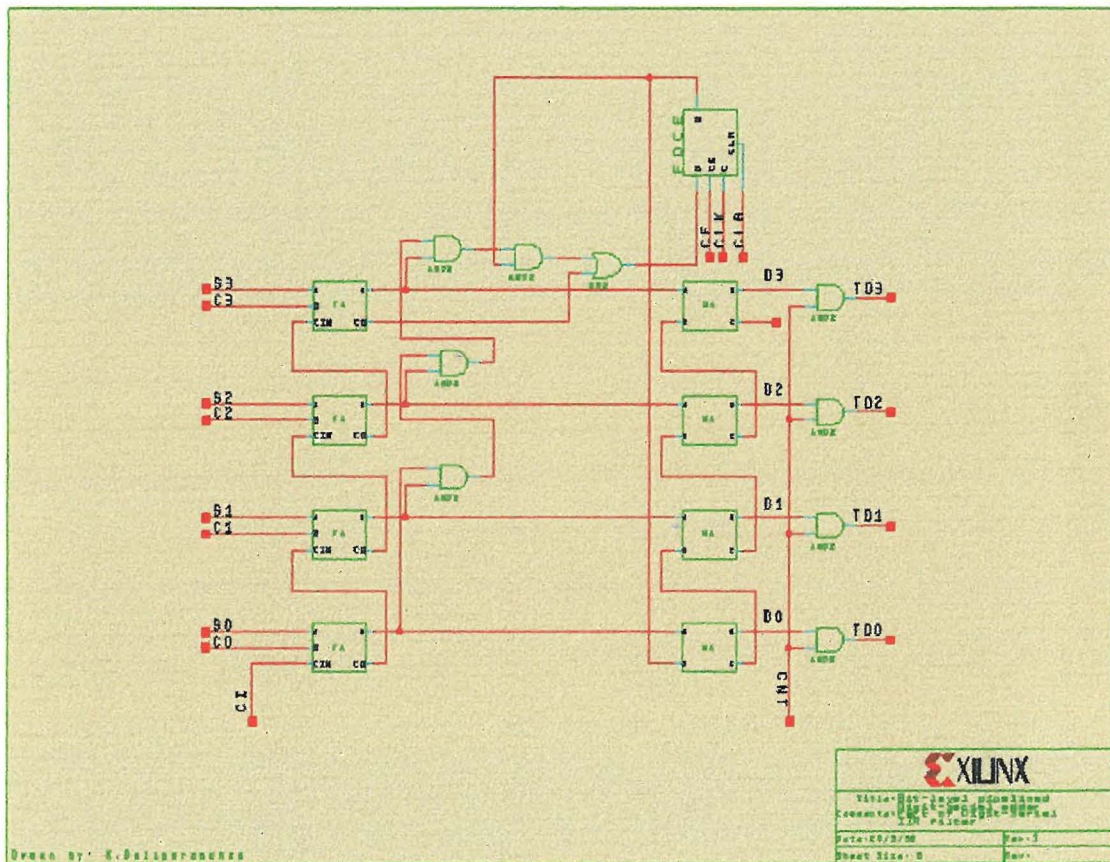


Figure 46: Bit-level pipelined digit-serial adder

Where,

$C_0, \dots, C_3$  and  $S_0, \dots, S_3$ : Carry and sum inputs (from CSAs array).

$O_0, \dots, O_3$ : Outputs (addition result)

$T_0, \dots, T_3$ : Truncated  $O_0, \dots, O_3$  outputs

CI: Carry input

CNT: Control input for truncation of the output word

The schematic of the bit-level pipelined digit-serial adder was created as symbol to be used later in the implementation (Fig. 47).

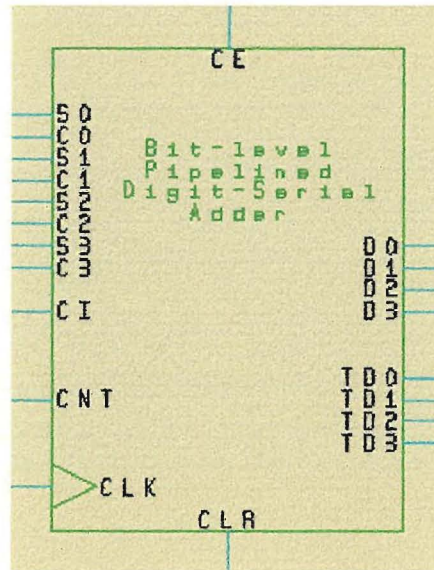


Figure 47: Bit-level pipelined digit-serial adder symbol

The bit-level pipelined digit-serial adder was simulated and tested according to the following way. It was connected to CSAs array part of the 16x16 bit digit-serial multiplier (Fig. 35) after the CRA had been removed. The multiplicand (B) was set to  $45123_{10}$  or  $1011000001000011_2$  and the (Y) multiplier was set to  $57000_{10}$  or  $\underbrace{1101111010101000}_D \underbrace{\phantom{1101111010101000}}_E \underbrace{\phantom{1101111010101000}}_A \underbrace{\phantom{1101111010101000}}_{8 \phantom{2}}$ . The product of the current multiplication is  $2572011000_{10}$  or  $\underbrace{1001100101001101110001011111000}_9 \underbrace{\phantom{1001100101001101110001011111000}}_9 \underbrace{\phantom{1001100101001101110001011111000}}_4 \underbrace{\phantom{1001100101001101110001011111000}}_D \underbrace{\phantom{1001100101001101110001011111000}}_C \underbrace{\phantom{1001100101001101110001011111000}}_5 \underbrace{\phantom{1001100101001101110001011111000}}_F \underbrace{\phantom{1001100101001101110001011111000}}_{8 \phantom{2}}$  and the truncated output is  $2571960320_{10}$  or  $\underbrace{10011001010011010000000000000000}_9 \underbrace{\phantom{10011001010011010000000000000000}}_9 \underbrace{\phantom{10011001010011010000000000000000}}_4 \underbrace{\phantom{10011001010011010000000000000000}}_D \underbrace{\phantom{10011001010011010000000000000000}}_0 \underbrace{\phantom{10011001010011010000000000000000}}_0 \underbrace{\phantom{10011001010011010000000000000000}}_0 \underbrace{\phantom{10011001010011010000000000000000}}_0 \underbrace{\phantom{10011001010011010000000000000000}}_2$ . The CMD file written for the simulation is presented in Table 11. The CNT input was set to 0 for the first 4 cycles in order to truncate the LSDs and to 1 for the rest. The CI was set to 0.

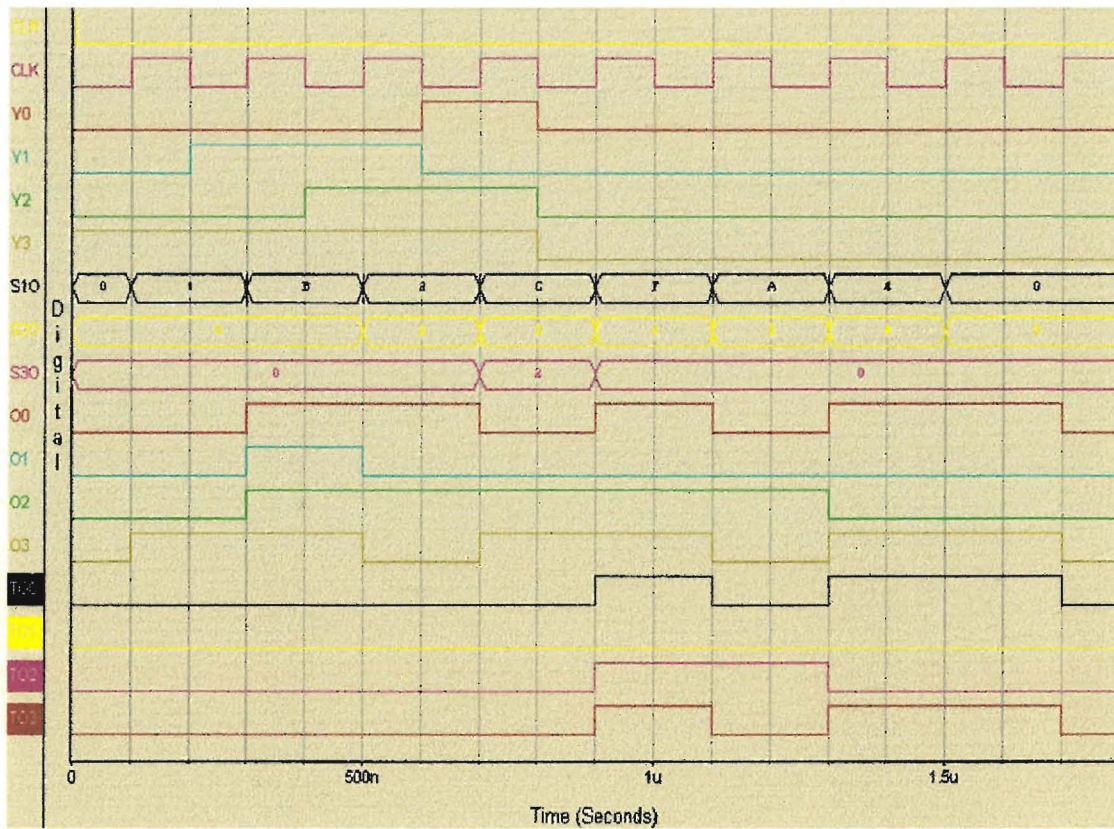
```

ECHO FINAL YEAR PROJECT
ECHO Digital Filter Implementation on FPGA
ECHO 16x16 bit Bit-level pipelined digit-serial adder Simulation File
ECHO 26/2/97
ECHO BY K.Deliparaschos
|
RESTART
VECTOR S1IN S1IN[3:0]
VECTOR S2IN S2IN[3:0]
VECTOR S3IN S3IN[2:0]
VECTOR S1O S1O[3:0]
VECTOR S2O S2O[3:0]
VECTOR S3O S3O[2:0]
VECTOR Y Y[3:0]
|
CLOCK CLK 0 1
|
WFM CE @0NS=1
WFM CLR @0NS=1 @5NS=0
|
WFM BYIN00 @0=0
WFM BYIN01 @0=0
WFM BYIN10 @0=0
WFM BYIN02 @0=0
WFM BYIN03 @0=0
|
|MULTIPLICAND
WFM B0 @0=1
WFM B1 @0=1
WFM B2 @0=0
WFM B3 @0=0
WFM B4 @0=0
WFM B5 @0=0
WFM B6 @0=1
WFM B7 @0=0
WFM B8 @0=0
WFM B9 @0=0
WFM B10 @0=0
WFM B11 @0=0
WFM B12 @0=1
WFM B13 @0=1
WFM B14 @0=0
WFM B15 @0=1
|
WFM CI @0=0
|
|MULTIPLIER
|
PATTERN CNT 0 0 0 0 1 1 1 1
PATTERN Y 8\H A\H E\H D\H 0\H 0\H 0\H 0\H
PATTERN S1IN 0\H 0\H 0\H 0\H 0\H 0\H 0\H 0\H
PATTERN S2IN 0\H 0\H 0\H 0\H 0\H 0\H 0\H 0\H
PATTERN S3IN 0\H 0\H 0\H 0\H 0\H 0\H 0\H 0\H
WAVE DSATEST.wfm CLR CLK Y0 Y1 Y2 Y3 S1O S2O S3O O0 O1 O2 O3 T00 T01
TO2 TO3
CYCLE 9

```

**Table 11:** Bit-level pipelined digit-serial adder CMD file

The results obtained from the simulation are presented below (Fig. 48).



**Figure 48:** Simulation results of the bit-level pipelined digit-serial adder

The correct results were obtained and hence the functionality of the bit-level digit serial adder was proven to be correct.

Finally the 1<sup>st</sup> order digit-serial filter was implemented according to the block diagram in Fig. 42. In order to reduce the hardware complexity and since only the functionality of the filter needs to be proved, the  $a_1$  coefficients were set to 0 so that the 16x16 bit digit-serial multiplier could be omitted. The schematic of the filter is illustrated on next page (Fig. 49).

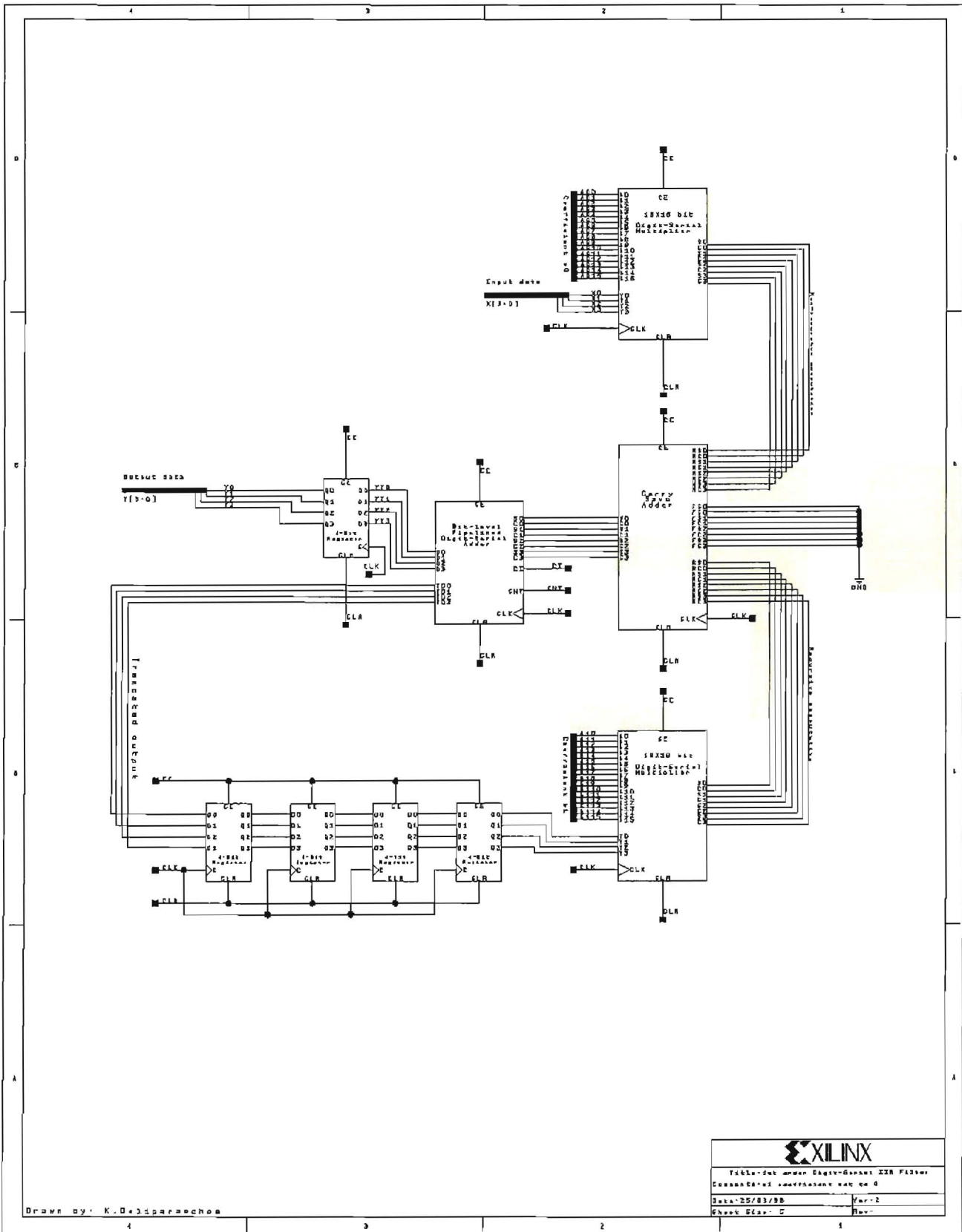


Figure 49: Schematic of 1<sup>st</sup> order digit-serial IIR filter



The filter was simulated for 24 cycles. Both  $a_0$  and  $b_1$  coefficients of the filter were assigned with the same number,  $45123_{10}$  or  $1011000001000011_2$ . The input,  $x_k$  was set to  $57000_{10}$  or  $\underbrace{1101111010101000}_{\substack{D \\ E \\ A \\ 8 \quad 2}}$  for the first 8 cycles and for the remaining cycles to 0. The results for the 24 cycles were calculated before the simulation in order to be compared with the simulation results afterwards.

For the **nonrecursive** computation (0-8 cycles):

Coefficients  $a_0, b_1$ : 1011 0000 0100 0011  
 Input  $x_k$ : 1101 1110 1010 1000  
 Output  $y_k$ : 1001 1001 0100 1101 1100 0101 1111 1000  
 Truncated o/p of  $y_k$ : 1001 1001 0100 1101 0000 0000 0000 0000

For the **recursive** computation (8-16 cycles):

Coefficients  $a_0, b_1$ : 1011 0000 0100 0011  
 Input (truncated o/p carried forward): 1001 1001 0100 1101  
 Output  $y_k$ : 0110 1001 1000 1101 0000 1111 0010 0111  
 Truncated o/p of  $y_k$ : 0110 1001 1000 1101 0000 0000 0000 0000

For the **recursive** computation (16-24 cycles):

Coefficients  $a_0, b_1$ : 1011 0000 0100 0011  
 Input (truncated o/p carried forward): 0110 1001 1000 1101  
 Output  $y_k$ : 0100 1000 1010 1100 1000 1111 1110 0111  
 Truncated o/p of  $y_k$ : 0100 1000 1010 1100 0000 0000 0000 0000

The CMD file listing is shown below (Table 12).

```
ECHO FINAL YEAR PROJECT
ECHO Digital Filter Implementation on FPGA
ECHO 1st order digit-serial IIR (a1 coefficient set to 0)
ECHO 26/3/98
ECHO BY K.Deliparaschos
|
RESTART
STEP SIZE 100NS
|
VECTOR X X[3:0]
VECTOR Y Y[3:0]
|
WFM CE @0NS=1
WFM CLR @0NS=1 @20NS=0
|DIGIT SERIAL ADDER CARRY INPUT
WFM CI @0=0
|TRANCACTION CONTROL
PATTERN CNT 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
|
|a0 coefficient
WFM A00 @0=1
WFM A01 @0=1
WFM A02 @0=0
WFM A03 @0=0
WFM A04 @0=0
WFM A05 @0=0
WFM A06 @0=1
WFM A07 @0=0
WFM A08 @0=0
WFM A09 @0=0
WFM A010 @0=0
WFM A011 @0=0
WFM A012 @0=1
WFM A013 @0=1
WFM A014 @0=0
WFM A015 @0=1
|
|b1 coefficient
WFM B10 @0=1
WFM B11 @0=1
WFM B12 @0=0
WFM B13 @0=0
WFM B14 @0=0
WFM B15 @0=0
WFM B16 @0=1
WFM B17 @0=0
WFM B18 @0=0
WFM B19 @0=0
WFM B110 @0=0
WFM B111 @0=0
WFM B112 @0=1
WFM B113 @0=1
WFM B114 @0=0
WFM B115 @0=1
```



## Chapter 5      **IMPLEMENTATION AND TESTING OF 16X16 BIT DIGIT-SERIAL MULTIPLIER ON FPGA**

### **5.1 Introduction**

Even though that the original title of the project was ‘implementation of digit-serial IIR filter on FPGA’, due to limitations on the hardware available at the present time, the current implementation on FPGA was reduced to that of the 16x16 bit digit-serial multiplier. The major reason for this alteration was, that the available *Xilinx* FPGA chip (XC3042PC84) at the time was contained on a general use experimental board with many of the user defined I/O pins already engaged (Appendix A). Also the size of the present FPGA chip (2,000 to 3,000 gates) did not allow the size required (>7,000 gates) by the 1<sup>st</sup> order digit-serial IIR filter [33].

### **5.2 Overview for *Xilinx* FPGAs**

Every *Xilinx* FPGA performs the function of a custom LSI circuit, like a gate array, but the *Xilinx* device is user programmable and even reprogrammable in the system. *Xilinx* sells standard off-the-shelf devices in three families, and many different sizes, speeds, operating temperature ranges, and packages (Appendix B). The user selects the appropriate *Xilinx* device, and then converts the design idea or schematic into a configuration data file, using the *Xilinx* development software (*XACT*) running on a PC or workstation, and loads this file into the *Xilinx* FPGA.

The *XACT* development system generates the configuration program bitstream used to configure the LCA device [33].

### 5.3 Programming or Configuring the Device

A design usually starts as block diagram or schematic, drawn with one of the popular CAE tools, e.g. *Viewdraw* (part of *Viewlogic* software). Many of these tools have an interface to *XACT*, the *Xilinx* development system.

After schematic- or equation-based entry, the design is automatically converted to a *Xilinx* Netlist Format (XNF). The *XACT* software first partitions the design into logic blocks, then finds a near-optimal placement for each block, and finally selects the interconnect routing. This process of Partitioning, Placement, and routing (PPR) runs automatically, but the user may also affect the outcome by imposing specific constraints, or selectively editing critical portions of the design, using the graphic Design Editor (XDE).

Once the design is complete, it is documented in an LCA file, from which a serial bitstream file can be generated. The user then exercises one of several options to load this file into the *Xilinx* FPGA device, where it is stored in latches, arranged to reassemble one long shift register. The data content of these latches customises the FPGA to perform the intended digital function [33].

## 5.4 Downloading of 16x16 bit Digit-Serial Multiplier on FPGA and Testing

The 16x16 bit digit-serial multiplier with shift registers described in section 4.9 of chapter 4, was prepared to be downloaded into the FPGA chip. In order to reduce the hardware size and need for I/O pins further, for the reason described in section 5.1, the following changes were made.

The serial-in to parallel-out register was omitted from the design and only one 4 bit register was left at the output to allow time for the computation to finish. Furthermore, the parallel input data (multiplicand) of the multiplier was predefined to a randomly selected number ( $45123_{10}$  or  $1011000001000011_2$ ), leaving only the serial input data (multiplier) user defined. These changes resulted to a significant reduction of the I/O pins to 22.

The debounced switch S12 on the FPGA board was used to clock the multiplier at each cycle. Since the output of the switch was inverted (see Appendix A) an extra inverter was used at the clock input of the multiplier to cancel the inversion. Switch S15 on the FPGA board was used to reset the circuit. After a close examination at the FPGA board schematic (Appendix A), the following pins shown on next page were assigned to the inputs/outputs of the multiplier.

<b>YY0 to pin 75</b>	<b>YY4 to pin 82</b>	<b>YY8 to pin 9</b>	<b>Y12 to pin 4</b>
<b>YY1 to pin 76</b>	<b>YY5 to pin 84</b>	<b>YY9 to pin 11</b>	<b>YY13 to pin 2</b>
<b>YY2 to pin 77</b>	<b>YY6 to pin 3</b>	<b>Y10 to pin 10</b>	<b>YY14 to pin 83</b>
<b>YY3 to pin 78</b>	<b>YY7 to pin 5</b>	<b>YY11 to pin 8</b>	<b>YY15 to pin 81</b>

<b>O0 to pin 37</b>	<b>O1 to pin 38</b>	<b>O2 to pin 39</b>	<b>O3 to pin 40</b>
---------------------	---------------------	---------------------	---------------------

**CLK to pin 73**

**CLR to pin 6**

Input and output buffers (IBUF, OBUF) followed by input and output pads (IPAD, OPAD) were used before the design could be downloaded on the FPGA.

The new schematic of the 16x16 bit digit-serial multiplier with parallel-in to serial-out shift register is shown in Fig. 51 on the page overleaf.

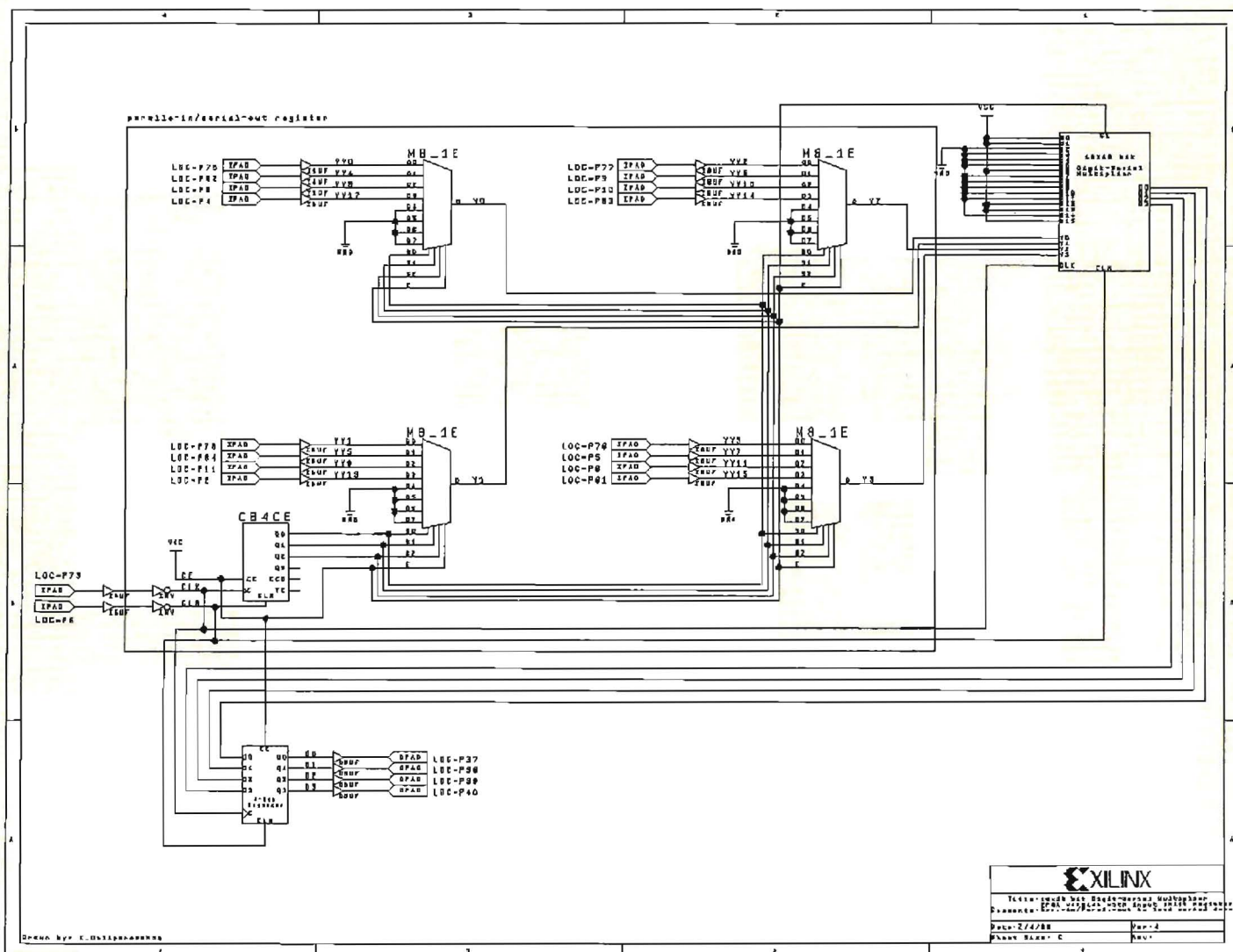


Figure 51: 16x16 bit digit-serial multiplier with parallel-in/serial out shift register (FPGA version)



With the use of XACT software and by carefully following the instructions described in section 6.3, the schematic of Fig. 53 was successfully downloaded on the FPGA board. The parallel input  $YY_0, \dots, YY_{15}$  was set to  $57000_{10}$  or  $\underbrace{11011}_{D} \underbrace{11101}_{E} \underbrace{01010}_{A} \underbrace{000}_{8} \underbrace{\phantom{000}}_2$ . A power supply was connected to the board to provide the 5v required and a 4 channel digital oscilloscope was used in order to observe the outputs  $O_0, \dots, O_3$ .

Finally the 16x16 bit digit serial multiplier was tested and produced the correct product of the multiplication, after 8 cycles ( $2572011000_{10}$  or  $\underbrace{1001}_{9} \underbrace{1001}_{9} \underbrace{1001}_{4} \underbrace{1101}_{D} \underbrace{110001}_{C} \underbrace{1011}_{5} \underbrace{1111}_{F} \underbrace{1000}_{8} \underbrace{\phantom{000}}_2$ ).

Appendix C demonstrates two photographs taken in the lab, showing the actual FPGA board (bottom left corner), set up with the computer (middle), oscilloscope (right) and power supply (top left corner). At the time the photograph was taken the circuit described before in this chapter was already downloaded onto the FPGA board and was functioning correctly.

## Chapter 6 CONCLUSION

The application of the digit-serial structures to the design of IIR filters introduces delay elements in the feed back loop of the IIR filter. This enables the pipelining of the feed back inherent in the IIR filters. The digit-serial structure is based on the feed forward of the carry digit, which allows sub digit pipelining to increase the throughput rate of the IIR filters.

Chapter 3 presented a systematic methodology to derive a wide range of digit-serial IIR filter architectures, which can be pipelined to the sub digit level. This will give the designers greater flexibility in finding the best trade-off between hardware cost and throughput rate.

Chapter 4 presents in detail the implementation of a 1<sup>st</sup> order digit-serial filter, which was previously described in depth in chapter 3. This can be achieved by designing, simulating and testing each element of the digit-serial filter. ECAD *Viewlogic* software was used for the design, simulation and testing of the digital elements of the filter. Great attention was paid to the design of the fundamental elements of the filter, in order to achieve the minimum number of gates used as possible, since this would reduce in a high order the hardware size and complexity. The aim of chapter 4 was to prove the functionality of the digit-serial IIR filter and all its sub-elements. Also to create a library of fundamental building blocks, to ease the design of future digit-serial IIR filter. Both of the aims were completed successfully.

Chapter 5 illustrated the methods carried out in order to download the design of a 16x16 bit digit-serial multiplier on FPGA board. Also the functionality of 16x16 bit digit-serial multiplier was proven and tested in practice. Due to the limitations on the variety of FPGA chips available at the present time and after a common agreement with the supervisor Dr A. Aggoun, it was decided to reduce the implementation on FPGA of the 1<sup>st</sup> order digit-serial filter to that of 16x16 bit multiplier. The aims set for chapter 5 were successfully completed, since the 16x16 bit digit-serial multiplier was proven to function satisfactorily in a real time situation. Also by achieving that a good step forward set in proving the functionality of digit-serial filter on FPGA, in future work. A few problems were encountered when the download of the 16x16 digit-serial multiplier on FPGA took place. These were due to loose connections between the FPGA chip and the I/O connectors of the board where the chip located. Fortunately the problem was overcome and the rest of the process was continued normally.

Obviously, in order to understand the concepts of digit-serial IIR structures took a great deal of time. Without any prior knowledge in this field, background reading was also essential. The correct results were not gained immediately and in many cases several attempts at understanding concepts required as necessary to achieve that. Regular meetings with the project supervisor were proven to be invaluable. Knowledge of the subject area was gained mostly by past papers, books and Internet resources. Throughout the period of this project many software packages such as, *Viewlogic, XACT, Office 97, Paint Shop Pro, and Microsoft Photo Editor, were used.*

As time is important in industry and deadlines need to be met, following the Gantt Chart (Appendix D) helped in achieving the completion of the required task within the available time.

Having the opportunity to undertake a project of this nature has proven to be an invaluable source of knowledge. It has allowed discovering previously unfamiliar areas of expertise, which will be beneficial when considering future career opportunities. Also by allowing the student to use appropriate tools such as, *Viewlogic*, *XACT* from previous areas of study enabled him to plan and undertake investigations both theoretical and practical.

## Chapter 7      **RECOMMENDATIONS FOR FURTHER WORK**

In future work the 1<sup>st</sup> order digit-serial IIR filter could be implemented on a bigger size FPGA chip. Since the 1<sup>st</sup> order filter requires more than 7,000 gates the XC3195 or XC3195A *Xilinx* FPGA chip could be used to download the design. For a 2<sup>nd</sup> order digit-serial filter (more than 13,000 gates) the XC4000/A/H *Xilinx* family, which has a capacity greater than 25,000 gates could be used.

Moreover future work could involve the design of an individual interface board to support the chip. This would make available the entire user programmable I/O pins of the chip, so that using all of I/O pins of the current design required would not be a problem.

Furthermore an investigation based on the propagation delays, after the circuit has be downloaded on the FPGA would be essential. The propagation delays can not be measured until a complete design is ready to be downloaded onto the chip. This is due the fact that the length of wires between the interconnections effects the delays.

## References

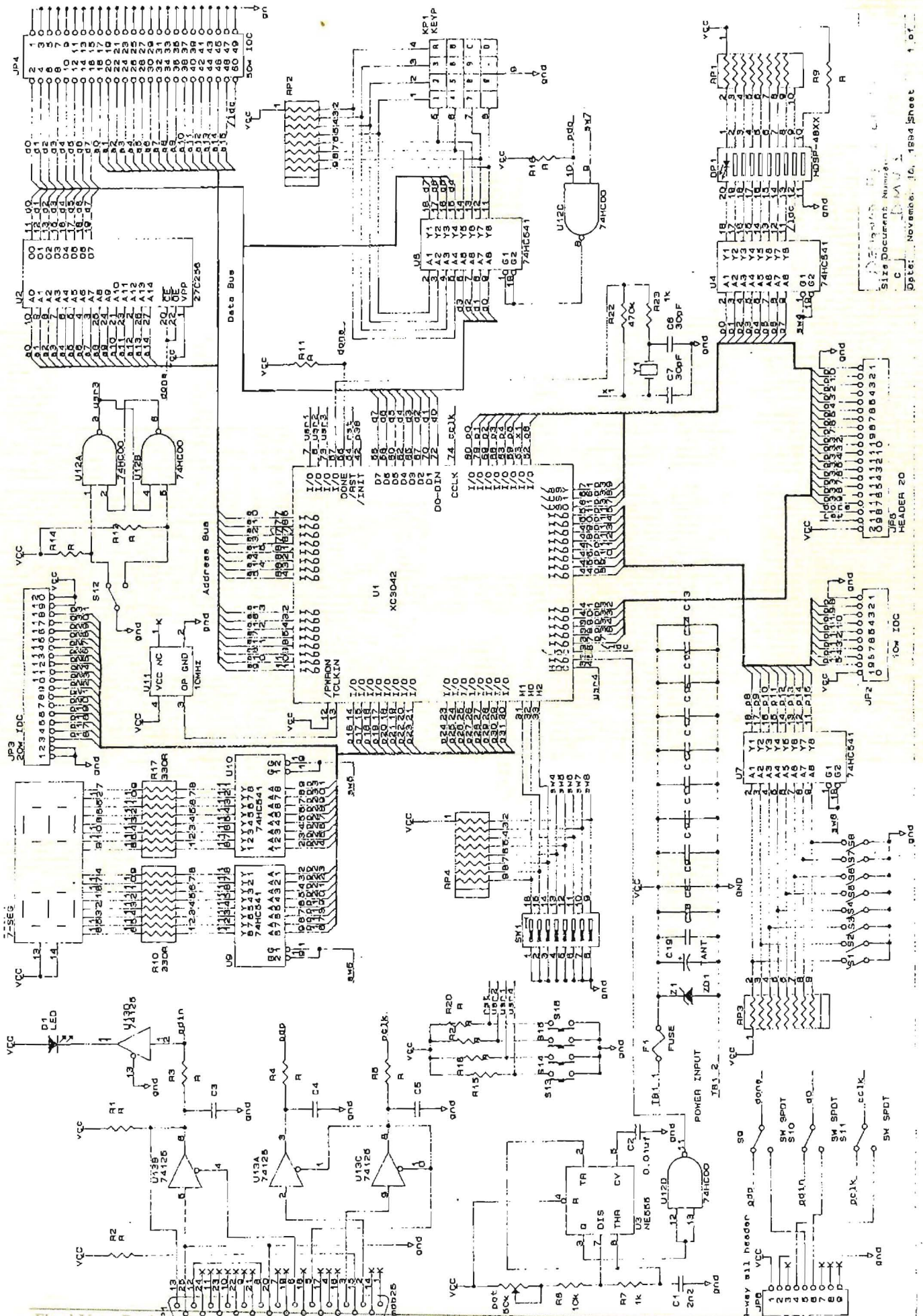
- [1] A. Aggoun, M.K. Ibrahim and A. Ashur, "Radix IIR filter structures: A framework for design trade-off analysis", Accepted for publication in *IEEE Transactions on Circuits and Systems*.
- [2] A. Aggoun, M.K. Ibrahim and A. Ashur, "Bit Level Pipelined Digit-Serial Array Processors", Accepted for publication in *IEEE Transactions on Circuits and Systems*.
- [3] A. Aggoun, A. Ashur and M.K. Ibrahim, "Bit Level Pipelined Digit-Serial Multiplier" *Int. J. Electronics*, 1993, Vol. 75, No. 6, pp. 1209-1219.
- [4] M. K. Ibrahim, "Radix-2<sup>n</sup> Multiplier Structures: A Structured Design Methodology", *IEE Proceedings-E*, Vol. 140, No. 4, July 1993, pp. 185-190.
- [5] S. C. Knowles, R. F. Woods, J. G. McWhirter and J. V. McCanny, "Bit-level Systolic Arrays for IIR Filtering", *Proceedings IEEE International Conference on Systolic Arrays*, San Diego, May 1998, pp. 653-663.
- [6] S. C. McQuillan, J. V. McCanny, "A Systematic Methodology for the Design of High Performance recursive Digital Filters", *IEEE Transactions on Computers*, Vol. 44, No. 8, August 1995, pp. 971-982.
- [7] K. K. Parhi, "A Systematic Approach for Design of Digit-Serial Signal Processing Architectures", *IEEE Transactions on Circuits and Systems*, Vol. 38, No. 4, April 1991, pp. 358-375.
- [8] M. A. Sid-Ahmed, "A Systolic realization for 2-D Digital Filters", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 37, No. 4, April 1989, pp. 560-565.
- [9] M. Hatamian, "Parallel Bit-level Pipelined VLSI Designs for High-Speed Signal processing", *Proceedings of the IEEE*, Vol. 75, No. 9, September 1987, pp. 1192-1202.
- [10] M. Hatamian, "An 85-MHz Fourth-Order Programmable IIR Digital Filter Chip", *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 2, February 1992, pp. 175-183.
- [11] R. Hartley, P. Corbett, "Digit-Serial Processing Techniques", *IEEE Transactions on Circuits and Systems*, Vol. 37, No.6, June 1990, pp. 707-718.
- [12] K. K. Parhi, "Pipeline Interleaving and Parallelism in Recursive Digital Filters-Part I: Pipelining Using Scattered Look-Ahead and Decomposition", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol.37, No. 7, July 1989, pp. 1099-1117.

- [13] Z. Jiang, A. N. Willson, "Design and Implementation of Efficient Pipelined IIR Digital Filters", *IEEE Transactions on Signal Processing*, Vol. 43, No. 3, March 1995, pp. 579-590.
- [14] W. Luk, G. Jones, "Systolic Recursive Filters", *IEEE Transaction on Circuit and Systems (CAS)*, Vol. 35, No. 8, August 1998.
- [15] P. E. Danielsson, "Serial/Parallel Convolver", *IEEE Transaction on Computers*, Vol. 33, No. 7, July 1984, pp. 652-667.
- [16] R. F Woods, J. V. McCanny, "Design of a High-Performance IIR Digital Filter Chip", *IEE Proceedings-E*, Vol. 139, No. 3, May 1992.
- [17] A. Aggoun, A. Ashur and M.K. Ibrahim, "Novel Cell Architecture for High Performance Digit-serial Computation", *Electronic Letters*, 27<sup>th</sup> May 1993, Vol. 29, pp. 938-940.
- [18] A. Aggoun, A. Ashur and M.K. Ibrahim, "Bit-level Pipelined Digit-serial Array processors", Accepted for publication in *IEEE Trans. on Circuits and Systems*.
- [19] M.K. Ibrahim, A. Aggoun, A. Ashur, "Radix Multiplication Algorithms", *Int. J. Electronics*, 1995, Vol. 79, No. 3, pp. 329-345.
- [20] A. Avizienis, "Signed-Digit Number Representations for fast Parallel Arithmetic", *IRE Trans. Computers*, Vol. EC-10, pp. 389-400, Sept 1961.
- [21] A. E. Bashagha, M. K. Ibrahim, "Radix Digit-serial Pipelined Divider/square-Root Architecture", *IEE Proc.-Comput. Digit. Tech*, Vol. 141, No6, November. 1994, pp. 375-380.
- [22] P. B. Denyer, D. Renshaw, "VLSI Signal Processor - a Bit-Serial Approach", (Addison-Wesley, 1985).
- [23] K. Hwang, "Computer Arithmetic Principle, Architecture and Design", (John Wiley & Sons, New York, 1979).
- [24] R. B. Urquhart, D. Wood, "Systolic Matrix and Vector Multiplication Methods for Signal Processing", *IEE Proceedings*, Vol. 131, Pt. F, No.6, October 1984, pp. 623-631.
- [25] A. Antoniou, "Digital Filters Analysis and design", 6<sup>th</sup> Edition, (McGraw-Hill, Inc., New York, 1979).
- [26] R. W. Hamming, (Digital Filters), 2<sup>nd</sup> Edition, (Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983).

- [27] M. Davio, J.-P. Deschamps, A. Thayse, "Digital Systems with Algorithm Implementation", (M. Davio, J.-P. Deschamps, A. Thayse, 1983).
- [28] A. E. A. Almaini, "Electronic Logic Systems", 3<sup>rd</sup> Edition, (Prentice Hall Int. (UK) Ltd., 1994).
- [29] V. Cappellini, A.G. Constantinides, P. Emiliani, "Digital Filters and their Applications", (Academic Press Inc. (London) Ltd, 1978).
- [30] E. L. Johnson, M. A. Karim, "Digital Filters a Pragmatic Approach", (PWS publishers, U.S.A, 1987).
- [31] J. McCanny, J. McWhirter, E. Swartzlander Jr, "Systolic Array Processors", (Prentice Hall Int. (UK) Ltd, 1989).
- [32] J. D. Broesch, "Practical Programmable Circuits a Guide to PLDs, State Machines and Microcontrollers", (Academic Press, Inc., (London), 1991).
- [33] D. Dawson, "An Introduction to Xilinx FPGAs", (Reproduced with the kind permission of Xilinx Inc., Feb. 1995).
- [34] "Xilinx Selection Guide and Specific Data Sheets From the XACT Libraries", (Reproduced with the kind permission of Xilinx Inc.).



## **Appendix A    FPGA Board Schematic**



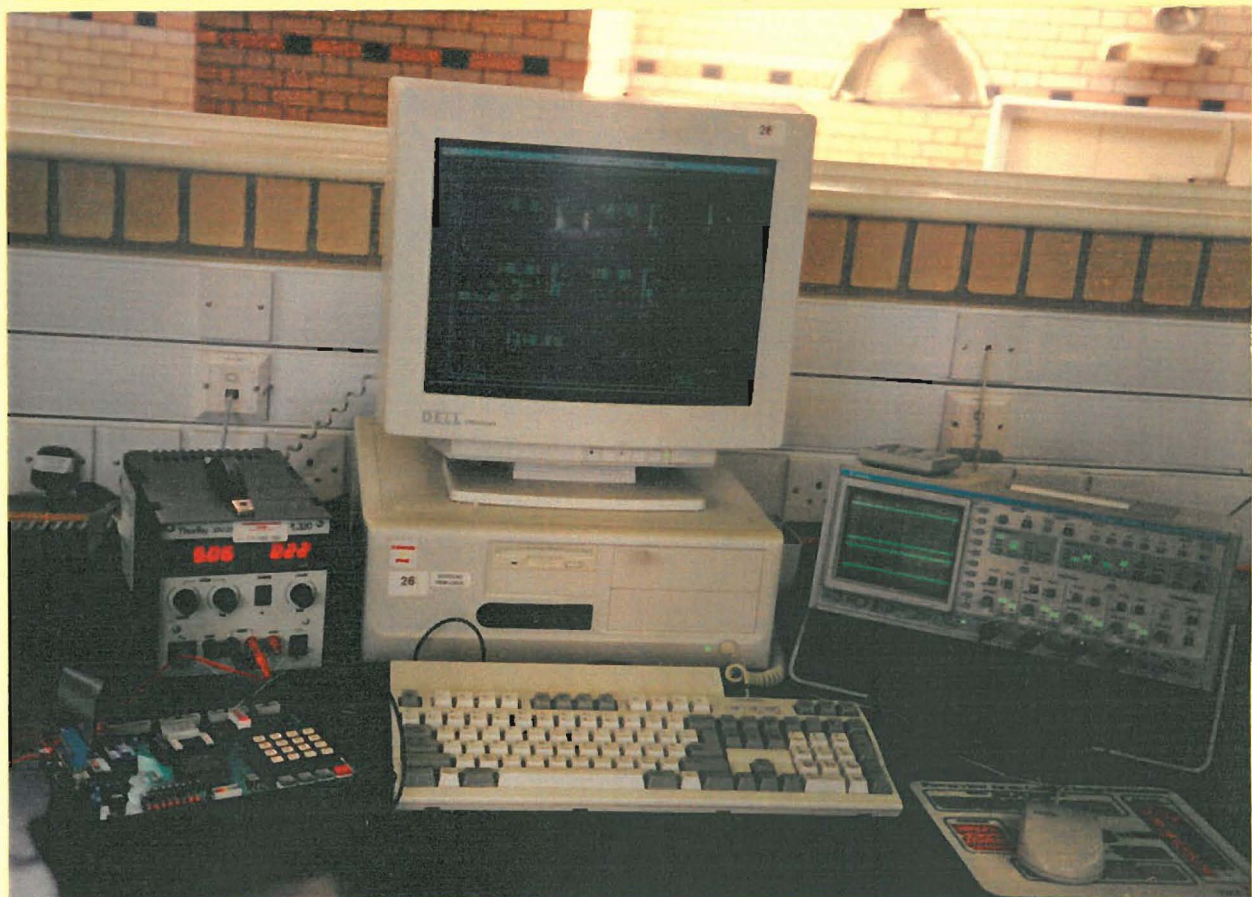
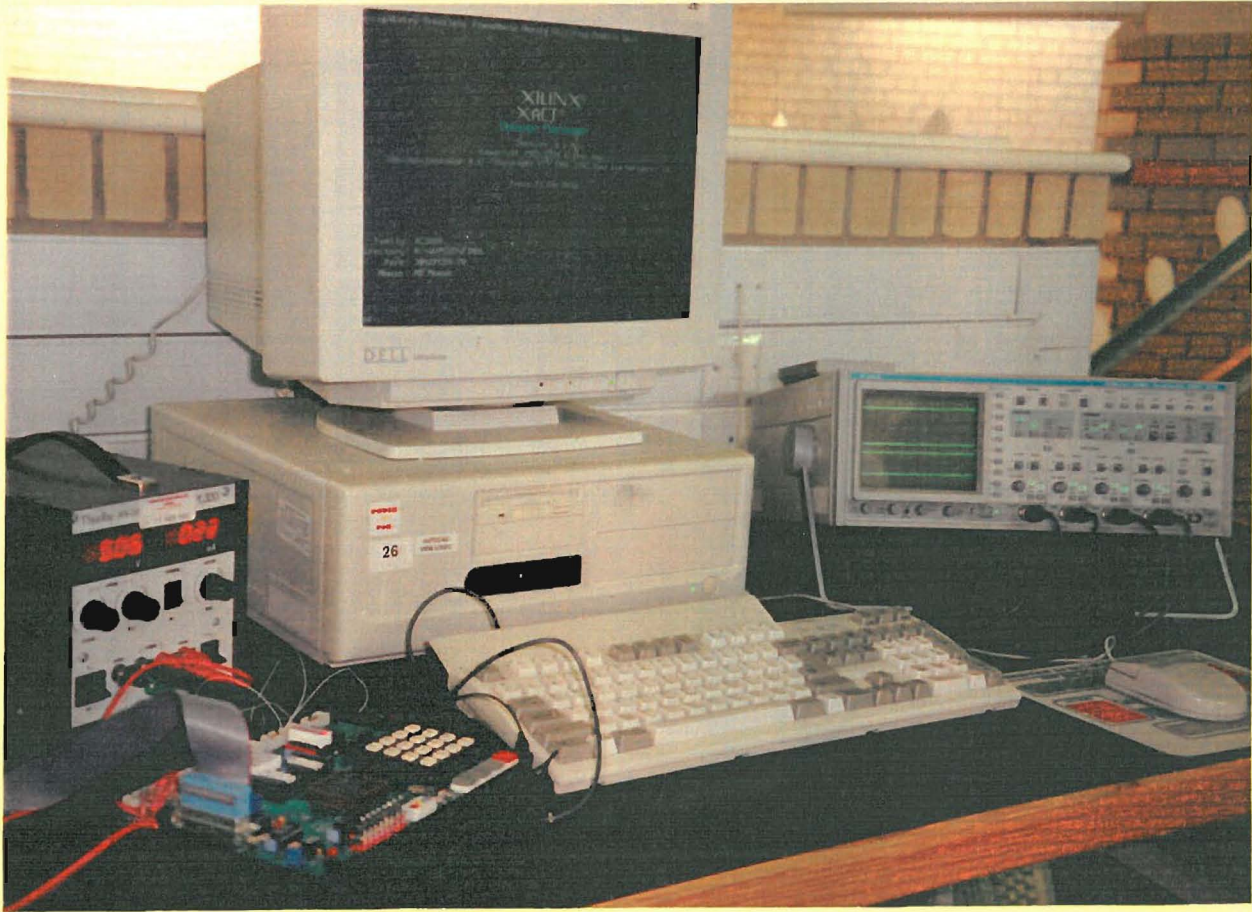
## **Appendix B    Xilinx Family Architecture Comparison**

Below is shown the *Xilinx* Family Architecture Comparison [33].

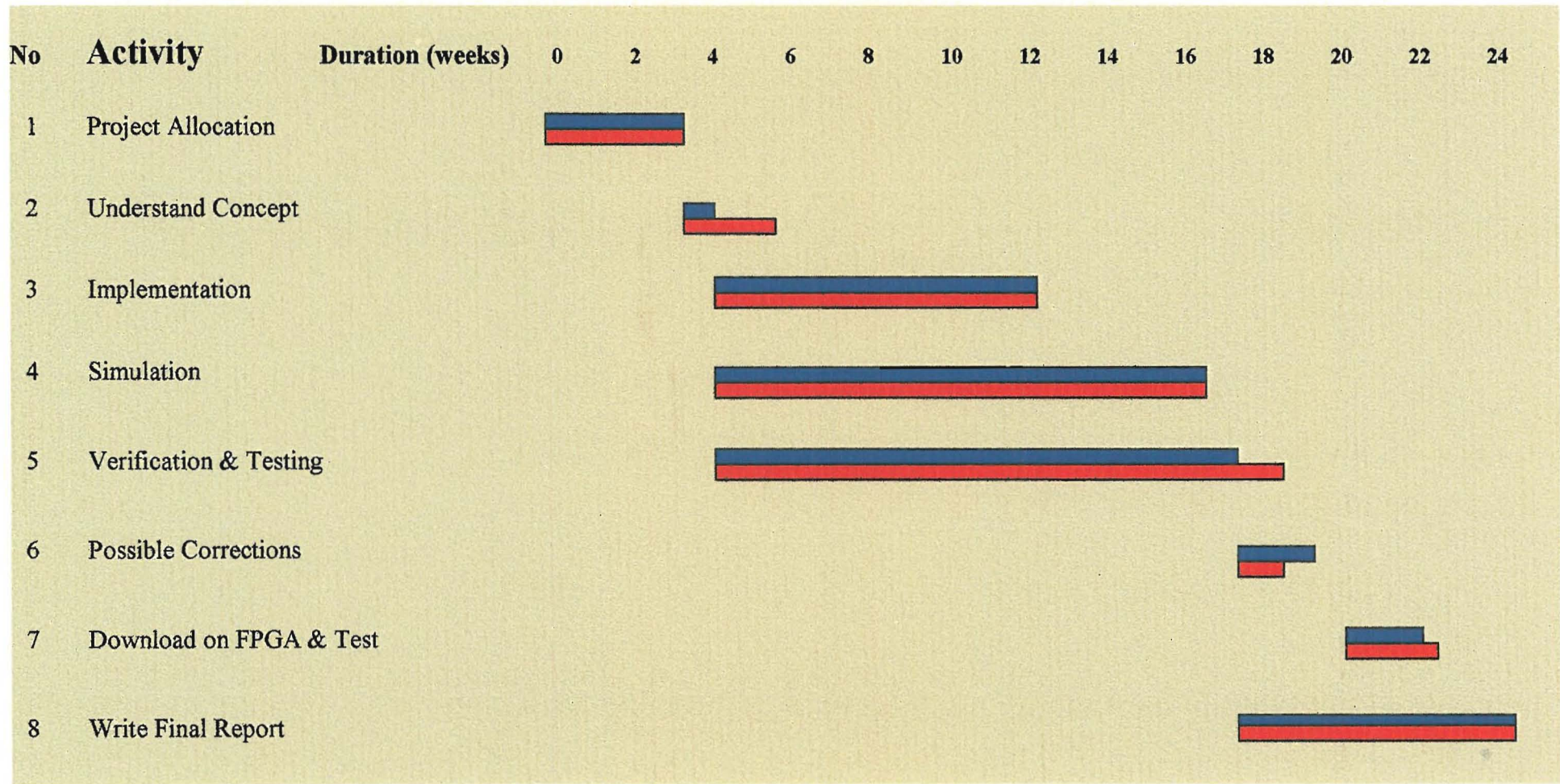
	EPLDs		FPGAs	
	XCF200 Family	XCF300 Family	XCF200Q, XCF300Q/A/L XCF3100 Family	XC4000/4000 Family
Architecture	PAL-like AND-OR plane Macrocells and product terms	Advanced PLD - high speed, high density function blocks (FB) in the same device	Gate array - like Many small blocks	Gate array - like Many small blocks
Logic Capacity	36 - 72 macrocells Integrate 4 - 8 PAL/ 22V10s	96 - 144 macrocells Integrate 4 - 16 PAL/ 22V10s	800 - 7,500 gates Integrate TTL, MSI, PLDs	1,600 - 25,000 + gates Integrate TTL, MSI, PLDs, RAM
Device Timing	Fixed PAL like 60 MHz - predictable	Fixed PAL-like 100 MHz - predictable	Gate array-like - depends on application Can be >100 MHz, typically 25 - 40 MHz (XCF300Q) or 50 - 80 MHz (XCF3100)	Gate array-like - depends on application Can be >100 MHz, typically 30 - 50 MHz
Number of VDDs	36 - 72	38 - 166	Many - like gate array 58 - 178	Many - like gate array 64 - 192
Number of FF	72 - 144	36 - 234	Many - like gate array 122 - 1,320	Very large number - 256 - 1,536 plus on-chip RAM - up to 18.4 K bits
Power Consumption	0.5 - 1.25 W static 0.75 - 1.8 W typical	0.4 - 2.0 W static 0.5 - 2.25 W typical Programmable power management	Very low, mW static Dynamic - depends on application 0.12 - 1.0 W typical	Very low, mW static Dynamic - depends on application 0.25 - 2.0 W typical
System Features	100% interconnect guaranteed Arithmetic carry logic ALU per macrocell 3.3 V/5 V VDD capability for XCF200/XCF300	100% interconnect guaranteed Arithmetic carry logic ALU per macrocell 3.3 V/5 V VDD capability for XCF200/XCF300 3 global clocks 12 mA/24 mA output drive Carry look ahead High output drive	Two global clock buffers Programmable output slew rate Internal 3-state busses Power-down mode 8 mA output drive for XCF3100	Eight global clock buffers Programmable output slew rate Internal 3-state busses RAM for FIFOs and registers JTAG for board test Fast carry logic for arithmetic Wide decoders 12 mA output drive, 24 mA per pair (24 mA/48 mA for AVN families)
Process	CMOS EPROM	CMOS EPROM	CMOS static RAM	CMOS static RAM
Programming Method	PRDM programmer OTP or UV erasable Configuration on chip	PRDM programmer OTP or UV erasable Configuration on chip	Programmed in circuit Four modes Configuration stored externally	Programmed in circuit Six modes Configuration stored externally
Re-programmable	Yes - after UV erasure	Yes - after UV erasure	Yes - in milliseconds Reprogrammable in circuit	Yes - in milliseconds Reprogrammable in circuit
Factory Tested	Yes	Yes	Yes	Yes
Key Applications	Complex state machines Complex counters Bus & peripheral interface Memory control PAL-brancher Accumulators/ integrators Magnitude/window comparators	High speed graphics multiplex memory controllers High speed bus interface 31 MHz, 18 bit accumulators DMA controllers Wide decoders High speed state machines Complex controllers	Simple state machines General logic replacement Reprogrammable logic devices Battery-powered logic 3-V operation Very fast counters	Simple state machines Complex logic replacement Board integration Accelerometers Reprogrammable applications RAM applications: FIFOs, buffers Fast/compact counters JTAG boards Bus interfacing

## **Appendix C    Photographs of the Project**

Below are photographs of the project under operation taken from the lab.



## **Appendix D    Gantt Chart**



Predicted Time: [Blue bar] Actual Time: [Red bar]



