The Development of an Application Specific Processor for the Transmission Line Matrix Method

by

David Stothard

A Doctoral Thesis

Submitted in partial fulfilment of the requirements for the award of

Doctor of Philosophy of Loughborough University

28 April, 2000

Abstract

This thesis details the development of an application specific processor for the transmission line matrix (TLM) method. The application of TLM to the modelling of wave propagation in two and three dimensions is introduced with the discussion focusing on the concept of computational efficiency. Methods for improving computational efficiency are reviewed, in particular the implementation of TLM on large scale parallel computers. It is shown that these methods, while increasing throughput, make inefficient use of available resources. The review of existing methods is used to define a set of goals for a new class of application specific TLM processor.

The development of an application specific processor based upon the two dimensional shunt node is presented. This gives rise to an efficient, bit serial scatter processor. The implementation of this processor within a complete, application specific TLM system is discussed. The system is based around a unique mapping of the TLM connect routine to hardware.

The bit serial scatter processor is modified to allow the modelling of inhomogenous and three dimensional media using the stub loaded shunt node, the symmetrical condensed node and the symmetrical super condensed node TLM schemes. It is shown that all four TLM schemes may be implemented within a single architecture without the introduction of redundant elements through the use of reconfigurable logic. The implications of interfacing this system to a host PC using the PCI bus are discussed.

The processor designs are reviewed within the context of the goals set for the work. It is shown that all of the goals were successfully met. The implications and limitations of the processor are discussed.

The thesis concludes with recommendations for areas worthy of further study.

Glossary

ρ	Boundary reflection coefficient
$\oplus$	Logical exclusive OR operation
ε	Permittivity
μ	Permeability
T	Transmission coefficient
~	Logical inversion
Architecture	The basic plan along which a computer has been developed
ARMA	Auto regressive moving averages
Bernstein's Condition	A condition for the independence of two operations
CAE	Computer aided engineering
CLB	Configurable logic block
CSAS	Carry save add shift
CUT	Circuit under test
Dispersion	In TLM - frequency dispersion. The spreading of frequency components of a wave front as the wave front propagates through the mesh.
DSP	Digital signal processing
EM	Electromagnetic
EMC	Electromagnetic compatibility
FD-TD	Finite difference - time domain
FE	Finite element
Fourier Transform	Method of obtaining frequency domain information from a time domain signal
FPGA	Field programmable gate array
FSP	Fast serial parallel multiplier
HDL	Hardware description language
IF	Inversion flag
LUT	Look up table
Mapping	The relation between a physical system and a computer model
M^W	Mesh width (nodes)
N^M	Number of nodes in a mesh
N^P	Number of scatter processors
PCI	Peripheral component interconnect bus
PE	Processing element
PPM	Prony-Pisarenko Method

Processing Rate	*See Throughput*
SCN	Symmetrical condensed node
Speed Up	The ratio of throughput on one processor to throughput on N processors
SSCN	Symmetrical super condensed node
Throughput	Number of results produced within a given time period
TLM	Transmission line matrix/method
t_p	Propagation delay
VHDL	VHSIC HDL (*see VHSIC, HDL)*
VHSIC	Very high speed integrated circuit
VLSI	Very large scale integration
W	Data word length (bits)
W^B	Data length including boundary data (bits)
W^S	Stub parameter word length
ZF	Zero flag

Contents

PUBLICATIONS

APPENDIX

Introduction

This thesis details the development of a new class of application specific processors for the transmission line matrix (TLM) method. The initial goal of the work was to develop a hardware based accelerator for TLM computations. A more structured set of goals were defined through a review of the literature surrounding this field. Potential applications for the processor include the motor industry and the electronics industry. In both of these areas TLM is being applied to increasingly complex models. A reduction in processing time for large models would provide a significant commercial advantage in these areas.

The application of TLM to the modelling of wave propagation in two and three dimensions is introduced in ***chapter 1***. The focus is on applications of TLM in electromagnetics as this represents the main field of application for TLM, however reference is made to other fields as appropriate. The discussion is based around the four main classes of node in electromagnetics. These are the two dimensional shunt node and stub-loaded shunt node, and the three dimensional symmetrical condensed node (SCN) and symmetrical super condensed node (SSCN). It is shown that the computational efficiency of TLM implementations on serial computers is low. This gives rise to long run times for large simulations. Methods for reducing run times are discussed, in particular the implementation of TLM on large scale parallel computers. These methods, while increasing throughput, make inefficient use of available resources. The two existing classes of application specific processor for TLM are introduced. A study of their operation reveals that neither class of processor successfully overcomes the limitations and inefficiency of other implementation methods. The key limiting factors of the parallel and application specific processing implementations of TLM are identified. From these a set of goals are derived for a new class of TLM processor.

- The granularity of the TLM algorithm must be successfully mapped to hardware. This means both removing redundant elements from the computational hardware and providing sufficient bandwidth for the connect process.
- The chosen architecture should not limit the processor to a single form of the TLM algorithm or a single mesh configuration.
- The chosen architecture should be scalable to allow any mesh size to be implemented.

- The processor must be accessible. That is its use should not be prohibited through
  - Portability/size
  - Cost
  - Programming requirements

Of these, possibly the most often overlooked criterion is that of accessibility. The new processor is of little practical importance if restrictions of access prevent its use by the TLM community as a whole.

The aim of this work is to develop an application specific TLM processor that successfully achieves the goals defined in ***chapter 1***, thus providing a significant step forward from existing implementations. A series of milestones are identified which together form an implementation strategy for achieving the specified aims. This implementation strategy is shown below.

OBJECTIVE	MILESTONE
Feasibility study	Demonstrate that an application specific processor can achieve a performance increase over existing computers
Definition of suitable target technology and implementation strategy	Evaluate technologies for implementing the processor. Define the design flow for the chosen technology
Development of a shunt node processor	Demonstration of a working shunt node scatter processor
Implementation of connect function	Demonstration of a complete system based on an array of shunt node processors
Extension of scatter processor to other TLM schemes (stub loading, SCN, SSCN)	Demonstration of a working processor incorporating several TLM schemes

The implementation strategy is designed to provide a structured development path. The initial design phases focus on the two dimensional shunt node. The shunt node is chosen as it is the simplest form of TLM, yet it encompasses many of the principles of the more complex TLM schemes. The shunt node processor is used as the basis for the development of scatter processors for the stub loaded shunt node, the SCN and the

SSCN. Once an understanding is gained as to the requirements of each form of TLM, an efficient method of combining the processors to form a general purpose processor can be found.

Chapter 2 introduces the basic concepts of digital arithmetic. Various architectures are presented for the four main arithmetic operations, addition, subtraction, multiplication and division. The compromises between logic complexity and throughput for these architectures are discussed. The concept of errors in digital arithmetic is introduced. The discussion focuses on quantisation error. It is argued that all digital arithmetic is inherently inaccurate and that the definition of an error is therefore application dependent.

With the clock rates of modern desktop computers exceeding 400 MHz the ability of an application specific processor to produce an increase in throughput must be validated. It is shown in ***chapter 3*** that the performance of the processor is closely linked to the implementation strategy. A review of implementation methods suggests the use of a field programmable gate array (FPGA) as a development platform. The Xilinx XC4000 family of FPGAs is shown to have an internal structure well suited to forming the logic required by the TLM processor. It is argued that the lack of defined structure for the TLM processor makes it a candidate for implementation using a behavioural hardware description language. The VHDL language is chosen as it is a widely recognised standard. The two dimensional (2D) shunt node algorithm is formulated in VHDL so as to optimise the circuit produced. The trade off between word length and accuracy is discussed. It is shown that given a sufficiently large word length, integer arithmetic provides an adequate level of accuracy. The use of a block floating point scheme is introduced. Logic synthesis is used to create a circuit from the HDL description. A review of the performance of the circuit, implemented on a Xilinx FPGA, reveals a potential throughput increase of an order of magnitude against software implementations at the time of writing. However the circuit does little to address the primary aims of the research . An array of scatter nodes is formed. This exhibits poor scalability and low efficiency. This is shown to arise from the mismatch between the needs of the TLM connect process and the interconnect capability of the processor.

In ***chapter 4*** it is shown that the main components of the shunt node processor, i.e. the ripple carry adders, may be replaced by full adders and delay elements. This effectively transforms the design into a bit serial shunt node processor. The bit serial design allows the word length to be varied so that throughput and accuracy can be

balanced to achieve the desired performance. It is shown that simple boundaries may be added to the shunt node processor by encoding boundary data passed to the processor. This allows an homogeneous array of scatter processors to operate on a TLM mesh of arbitrary geometry. The implementation of the scatter processor on a Xilinx XC4010 FPGA and the testing of its functionality are described .

The bit serial processor overcomes many of the bandwidth problems encountered when trying to form large arrays of scatter processors. There is potential for a low bandwidth SIMD array to be developed around the processor. However such arrays are shown to limit the size of mesh which may be efficiently implemented. ***Chapter 5*** investigates a more efficient mapping of the TLM mesh on to an array of scatter processors. This gives rise to the concept of a flexible connect logic array which allows a small number of scatter processors to operate on a mesh of arbitrary size. It is shown that unlike previous mappings, the new connect logic introduces no overhead to the calculations.

In order to eliminate any redundant elements from the scatter processor the TLM scatter equation is mapped directly into hardware. This limits the processor to a single form of TLM, the 2D shunt node representation. In order to expand the system to implement multiple TLM schemes an understanding must be gained of the requirements of each scheme. ***Chapters 6, 7*** and ***8*** discuss how the TLM system developed in ***chapter 5*** may be adapted to implement systems for the stub-loaded shunt node, the SCN and the SSCN. The issues involved in the design of a scatter processor for each scheme are presented and discussed. ***Chapter 7*** also shows how the connect logic of ***chapter 5*** may be extended to work in three dimensions.

Chapter 9 demonstrates how the processors for each of the four main node configurations may be combined in a single architecture. It is shown that by using reconfigurable processing elements as opposed to reprogrammable ones all the processors may be combined without introducing any redundant elements or instruction streams. The resulting architecture of the TLM general purpose processor (TLM-GPP) is shown to have achieved many of the aims laid out in ***chapter 1***. The implications of building a physical processor around the TLM-GPP concept are discussed.

Chapter 10 examines the importance of the interface between the TLM-GPP and the user. It is argued that the interface is important not only in terms of performance but also in terms of ease of use and accessibility. Given the popularity and wide

availability of personal computers, an implementation of the TLM-GPP on a PCI card is suggested. The modifications required to meet the PCI standard and their effect on the processor are discussed. It is concluded that while the PCI bus places limitations on the processor, it offers a development platform for a very powerful TLM processor within a PC.

The ***conclusions*** return to the aims specified in ***chapter 1*** and define how well each aim was achieved. It is shown that the TLM-GPP successfully addresses all the initial aims. The potential applications of the processor are discussed and are compared to those areas considered potential applications at the start of the work. It is concluded that the TLM-GPP represents a powerful and highly flexible tool for TLM, providing a significant step forward from existing implementations.

The adaptable nature of the TLM-GPP has been exploited to produce a system with genuine wide ranging potential. There is still much to discover before the full potential of this new class of TLM processor is realised. Some key points are discussed in the ***recommendations for further study*** which conclude the thesis.

1. Transmission Line Matrix Modelling

1.1 Introduction

Wave propagation is a major factor in all areas of everyday life, although the study of wave propagation through the use of analytical techniques is practical only in certain restricted cases. The idea of solving propagation problems via equivalent electrical networks has existed for many years[1]. However such solutions are impractical due to the difficulty of physically realising large, ideal networks. The advent of the digital computer has led to the development of various numerical propagation modelling techniques, particularly in the field of electromagnetics[2], such as the finite difference time domain (FD-TD), finite element (FE) and transmission line matrix (TLM) methods. Unlike FD-TD and FE, which are respectively differential and integral techniques, the TLM method, first introduced by Johns and Beurle[3] in 1971, is built around an array of scattering points each of which may act as a secondary radiator in a manner similar to that described in Huygens' principle. Thus the TLM method bears a close resemblance to the physical process of propagation, and is equally applicable to the study of both electromagnetic and acoustic waves. This chapter offers an introduction to TLM before discussing the specific issue of computational efficiency. Conclusions are drawn as to the failings of current TLM implementations and recommendations are developed for an improved TLM accelerator.

1.2 The Two Dimensional Transmission Line Matrix Method

Maxwell's equations[4] for wave propagation in two dimensions may be written as

$$
\begin{aligned}
-\frac{\delta}{\delta x}H_x-\frac{\delta}{\delta z}H_z &= \varepsilon\frac{\delta}{\delta t}E_y \\
-\frac{\delta}{\delta x}E_y &= -\mu\frac{\delta}{\delta t}H_x \\
-\frac{\delta}{\delta z}E_y &= -\mu\frac{\delta}{\delta t}H_z
\end{aligned}
\qquad (1.1)
$$

which combine, as demonstrated in the ***appendix***, to give the wave equation,

$$\frac{\delta^2}{\delta x^2}E_y + \frac{\delta^2}{\delta z^2}E_y = \varepsilon\mu\frac{\delta^2}{\delta t^2}E_y$$

Analytical solutions to these equations are possible only in restricted cases or through lengthy mathematical processes. The idea of solving complex numerical problems through the use of electrical network analogues has been around for many years[1]. Consider an LCR lumped element network, ***figure 1.1***, analogous to a length, Δl, of transmission line. Through consideration of the rates of change of the voltage and current in the line we arrive at the telegrapher's equation. A full derivation is given in the ***appendix***.

$$\frac{\delta^2 V}{\delta x^2} = L_d C_d \frac{\delta^2 V}{\delta t^2} + R_d C_d \frac{\delta V}{\delta t} \qquad (1.2)$$

Where L_d, C_d and R_d are the distributed inductance, capacitance and resistance of the transmission line. If we assume that R_d is negligible, i.e. we have an ideal transmission line, then the resistive term can be neglected and we are left with a one dimensional wave equation for lossless transmission.

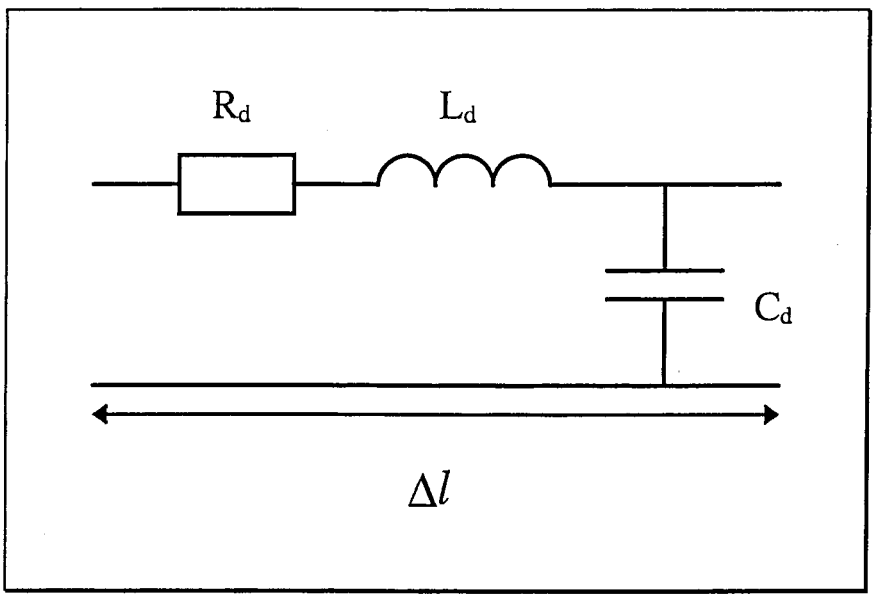


Figure 1.1 - Equivalent Circuit of a Length of Transmission Line

The basic building block of TLM is the shunt node[3], the two dimensional (2D) formulation of which is shown in ***figure 1.2***. It is formed by the orthogonal junction

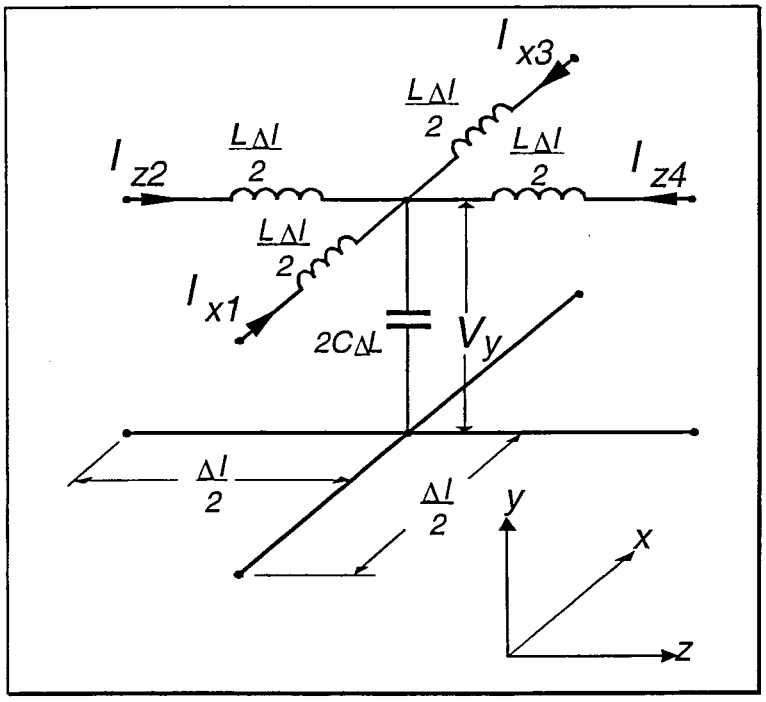


Figure 1.2 - Equivalent Circuit of a 2D Shunt Node

of two lossless transmission lines. This unit cell is repeated to fill the region under consideration with a Cartesian mesh of transmission lines. If Δl is small compared to the shortest wavelength under consideration then the node may be assumed infinitesimally small and the voltage and current changes at the node, according to Kirchoff's laws, are given by

$$-\frac{\delta}{\delta x}(I_{x1}-I_{x3})-\frac{\delta}{\delta z}(I_{z2}-I_{z4})=2C\frac{\delta}{\delta t}V$$

$$-\frac{\delta}{\delta x}V_y=L\frac{\delta}{\delta t}(I_{x1}-I_{x3}) \qquad (1.3)$$

$$-\frac{\delta}{\delta z}V_y=L\frac{\delta}{\delta t}(I_{z2}-I_{z4})$$

which may be combined as for Maxwell's equations, yielding the wave equation,

$$\frac{\delta^2}{\delta x^2}V_y+\frac{\delta^2}{\delta z^2}V_y=2LC\frac{\delta^2}{\delta t^2}V_y$$

Comparison of ***equation 1.3*** with Maxwell's equations, ***equation 1.1***, reveals the equivalences

$$E_y = V_y, \qquad -H_z = I_z$$
$$-H_x = I_x, \qquad \mu = L, \ \varepsilon = 2C$$

Although this discussion is centered on modelling electromagnetic phenomena it is noted that, from the basis of a suitable model or set of equations in the form of ***equation 1.3***, the TLM method may be used to study any form of transverse wave propagation. Of particular note is the wide application of TLM to acoustics[5] where equivalences are formed between the shunt node capacitance and inductance and the material properties of compressibility and density respectively.

A voltage impulse approaching a node 'sees' an impedance one third that of the line segment due to the three branches leaving the junction. This gives rise to a reflection coefficient, ρ, of

$$\rho = \frac{\frac{1}{3} - 1}{\frac{1}{3} + 1} = -\frac{1}{2}$$

and therefore a transmission coefficient in the other three branches of $(1+\rho) = +1/2$. Superposition of reflected and transmitted impulses for all four branches gives rise to a scattering matrix at the node which is commonly stated as[6]

$$\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix}^r = \frac{1}{2} \begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix}^i \tag{1.4}$$

and which may be expressed as

$$_{k+1}V_n^r = \frac{1}{2}\left[\sum_{m=1}^{4} {}_kV_m^i\right] - {}_k\ V_n^i \tag{1.5}$$

where the suffixes i and r denote incident and reflected impulses respectively and k is the iteration counter.

The velocity of impulses along the transmission lines is constant, therefore the discretisation of the spatial domain leads to a corresponding discretisation in the times at which scattering events occur. Impulses scattered from a node traverse a distance Δl in a time

$$\Delta t = \frac{\Delta l}{u}$$

where Δl is the node separation and u is the wave velocity. Δt is the iteration time, impulses scattered from a node at a time t form the incident impulses upon the neighbouring nodes at a time $t + \Delta t$. This process may be summarised by the TLM connect equations[3]

$$\begin{aligned}
{}_{k+1}V_1^i(z, x) &= {}_k\ V_3^r(z, x-1) \\
{}_{k+1}V_2^i(z, x) &= {}_k\ V_4^r(z-1, x) \\
{}_{k+1}V_3^i(z, x) &= {}_k\ V_1^r(z, x+1) \\
{}_{k+1}V_4^i(z, x) &= {}_k\ V_2^r(z+1, x)
\end{aligned} \qquad (1.6)$$

The modelling of propagation within the array follows an iterative sequence.

1. Excitation - The field components modelled at each node may be excited by applying the appropriate voltages or currents to the node.
2. Scatter - Radiated impulses are calculated at each node according to ***equation 1.4.***
3. Connect - New incident values at each node are calculated from ***equation 1.6.***

Data can be output from the array for visualisation purposes either as individual impulses or as the total energy incident upon each node in the array. Stages 1-3 are repeated until sufficient data is recovered to formulate a solution in the time domain or, via a Fourier transform, in the frequency domain. Sample outputs from TLM code written in C++ are presented in ***figure 1.3***. The ability to view the wave intensity at all points in the mesh simultaneously with little computational overhead is one of the major advantages of TLM.

Many practical problems require the modelling of a closed system, therefore TLM requires some form of boundary representation. To preserve synchronisation of the impulses within the mesh, boundaries are traditionally placed at a distance $\Delta l/2$ from a node. An impulse launched from a branch of a node upon which a boundary is present will be returned to the same node in the next iteration. Simple electric or magnetic walls can be formed by respectively short circuiting ($\rho = -1$) or open

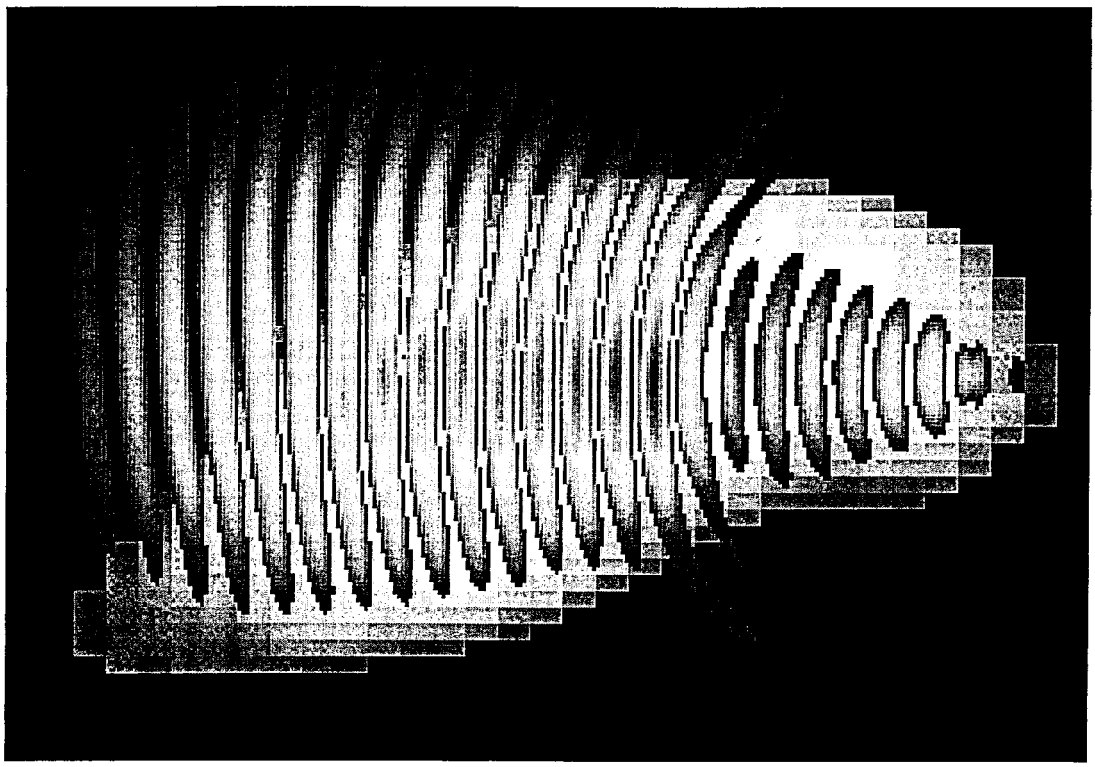


Figure1.3(a) - TLM Model of Dolphin Echolocation Beam Forming[1]

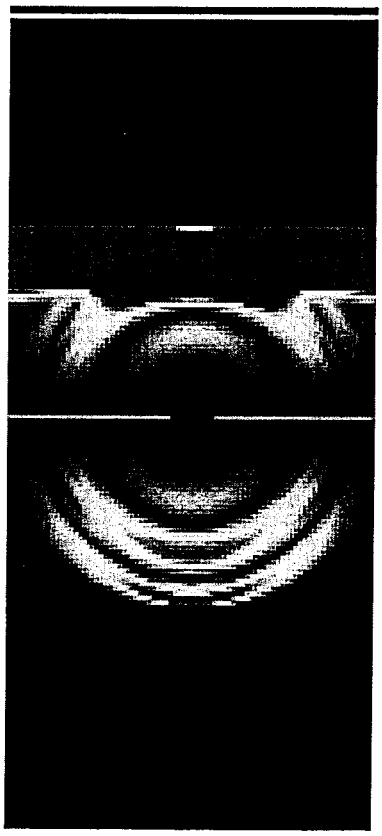


Figure 1.3(b) - Diffraction of a Wavefront Through a Narrow Slot

[1] Flint, J.A *et al* 'Visualising Wave Propagation in Bio-Acoustic Lens Structures using the Transmission Line Matrix Method', Proc. of the Institute of Acoustics, Vol. 19(9), pp.29-37, 1997

circuiting ($\rho = 1$) the relevant branch. More general boundaries can be represented by modifying the connect equation for that branch thus

$$_{k+1}V_4^i(p,q) =_k V_2^r(p+1,q) = \rho_k V_4^r(p,q) \tag{1.7}$$

where ρ is the reflection coefficient of the boundary and $-1 \leq \rho \leq +1$.

It has been shown[3] that the propagation velocity of a wave travelling upon the mesh is dependent upon the direction of travel and the mesh discretisation. The constituent axial components of a wave travelling at 45• to the axes have identical phase and amplitude, therefore when they converge at a node both will 'see' an impedance match. Propagation at 45• to the axes is hence unperturbed with a velocity of $\frac{1}{\sqrt{2}}u$.

However axial propagation is frequency dependent, giving rise to the dispersion characteristic of ***figure 1.4***[3]. If $\Delta l/\lambda \leq 0.1$ then dispersion is low and may be ignored. It is therefore accepted practice to allow at least 10 nodes per wavelength at the highest frequency encountered in a given problem.

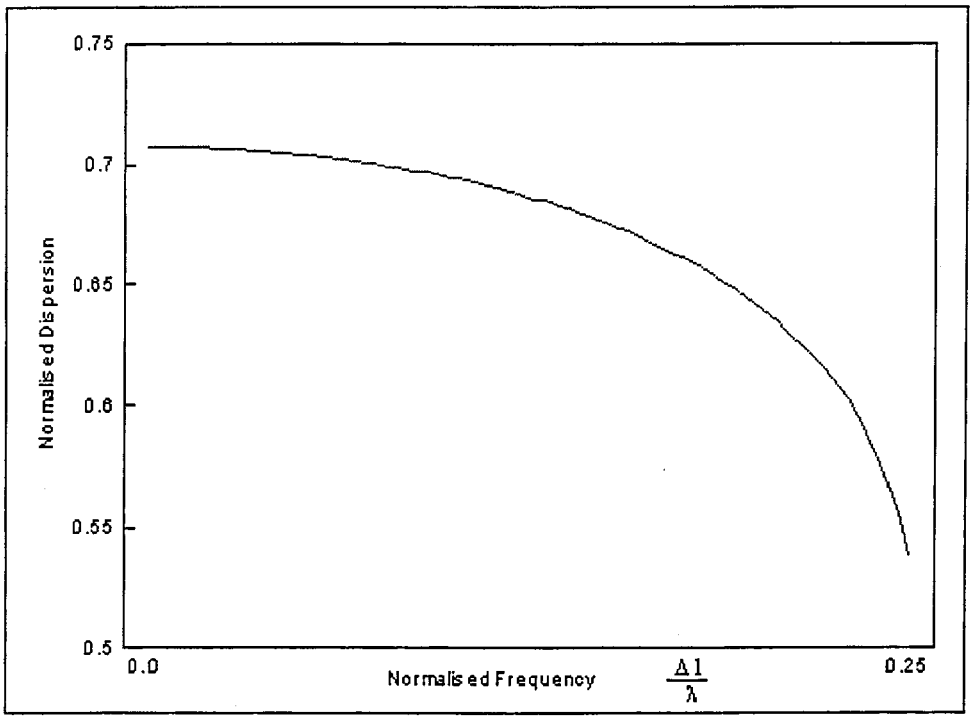


Figure 1.4 - Shunt Network Dispersion Characteristic

A second two dimensional node exists. This is created from the connection in series of four lossless transmission line segments. The series node is shown in ***figure 1.5***.

The series node scattering matrix has a similar form to the shunt node scattering matrix. Solutions to the series node are of the form:

$$ {}_{k+1}V_n^r = \frac{1}{2}\left[\sum_{m=1}^{4} {}_kV_m^i\right] - {}_kV_{s-n}^r $$

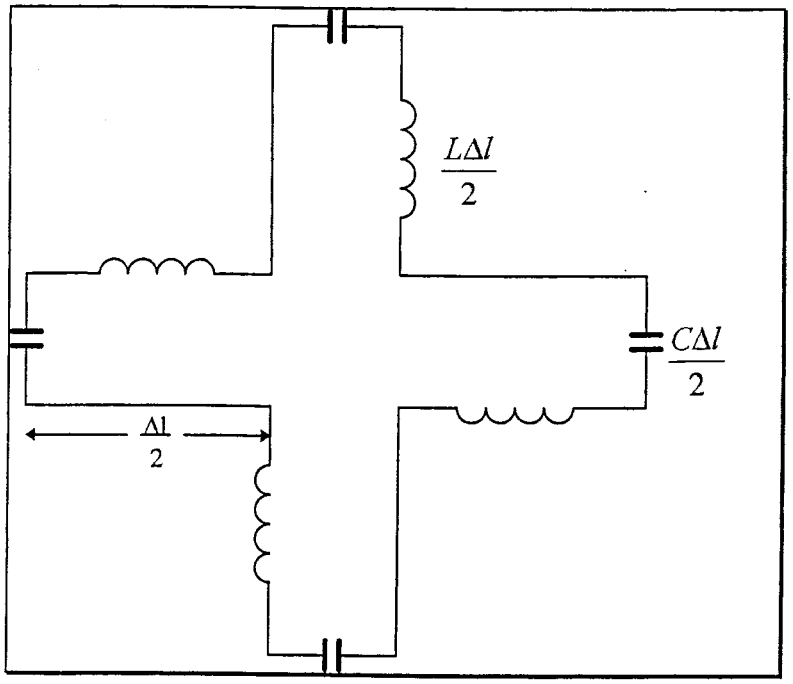


Figure 1.5 - The Two Dimensional Series Node

1.3 Two Dimensional TLM for Lossy and Inhomogeneous Materials

1.3.1 Stub Theory

In the field of electromagnetics two of the most important properties of a material are its permittivity, ε, and its permeability, μ. Similarly in acoustics the important properties are compressibility and density. These properties affect the propagation of waves within the material. The method for varying these properties within the TLM mesh is identical in both acoustics and electromagnetics, only the definitions change. For simplicity this discussion is focussed on electromagnetics.

Media with regions of varying permittivity/permeability exist throughout electromagnetics, common examples including the dielectric/plate interface of a parallel plate capacitor and the substrate (dielectric)/air interface in a microstrip line.

The two dimensional (2D) TLM shunt node represents an homogeneous medium of permittivity ε_0 and permeability μ_0. From ***equations 1.1*** and ***1.3***,

$$L \equiv \mu_0 \ , \qquad 2C \equiv \varepsilon_0$$

where L and C are respectively the distributed inductance and capacitance of the transmission line. In order to model a region of relative permittivity ε_r, (such that permittivity $\varepsilon = \varepsilon_0 . \varepsilon_r$), an extra capacitance must be introduced at the node. Similarly extra inductance is required to model a medium of permeability $\mu = \mu_r . \mu_0$. The extra capacitance/inductance is produced through the addition of stubs to the node. Capacitive stubs are usually used with shunt nodes and inductive stubs with series nodes.

1.3.2 Generation of Capacitance/Inductance Stubs

Consider the transmission line segment of ***figure 1.1***. Assuming the length of the line is short such that $\tan \beta x \approx \beta x$, a line segment of length x exhibits a complex impedance[7]

$$Z(x) = Z_0 \left[\frac{Z_L + Z_0 \, j\beta x}{Z_0 + Z_L \, j\beta x} \right] \tag{1.8}$$

If $Z_L = 0$, i.e. the end of the segment is short circuited, then $Z = j.Z_0 \beta x$, and the line exhibits an inductive impedance.
If $Z_L = \infty$, i.e. the end of the segment is open circuited, then $Z = -j.Z_0 \beta x$, and the line exhibits a capacitive impedance.
Thus the capacitance/inductance of a TLM node may be increased by adding an open/short circuited length of transmission line called a stub. The length of the stub is chosen to be $\Delta l/2$, thus ensuring that impulses scattered in to the stub and returned to the node are synchronised with the other impulses within the mesh.

Lossy materials, such that impulses passed between nodes are attenuated by a factor $e^{-\alpha \Delta l}$, may be represented by the addition of an infinite length or perfectly matched stub to the nodes[6]. Energy scattered in to this loss stub is absorbed and is not returned to the node.

1.3.3 Scattering at a Stub Loaded Shunt Node

Consider the stub loaded shunt node of ***figure 1.6***[8]. Impulses incident upon one of the four branch inputs encounter a reflection coefficient

$$\rho_{ii} = -\frac{y - 2}{y}, \qquad i = 1..4$$

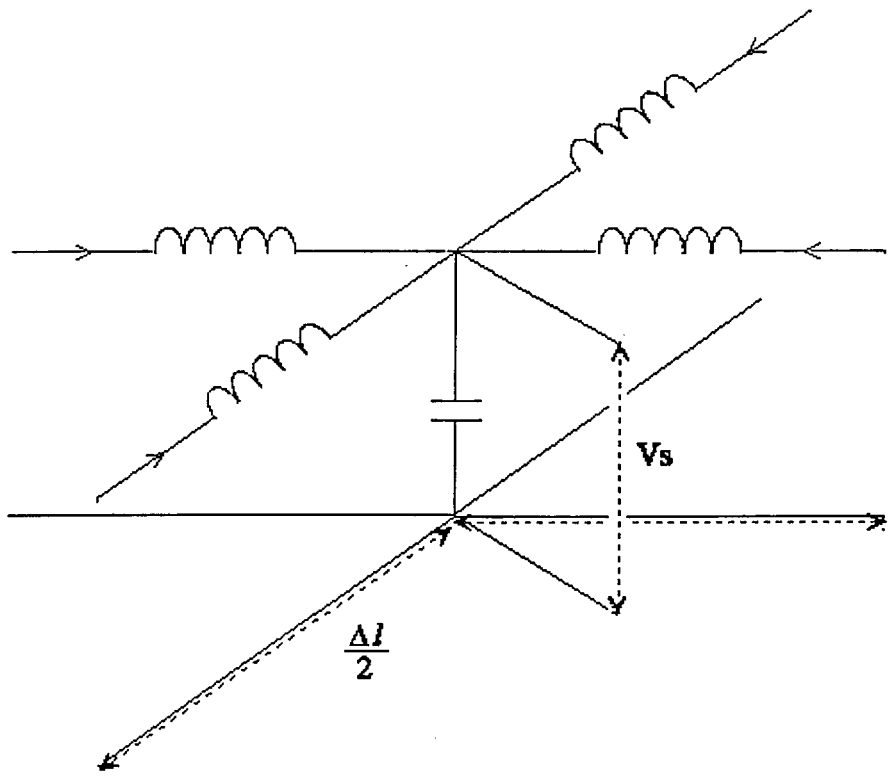


Figure 1.6 - Stub Loaded Shunt Node

and hence a transmission coefficient

$$T_{ki} = 1 + \rho_{ii} = \frac{2}{y}, \quad k \neq i$$

Impulses reflected from the permittivity stub encounter a reflection coefficient at the node of

$$\rho_{55} = \frac{2\,y_0 - y}{y}$$

This yields a transmission coefficient of

$$T_{i5} = 1 + \rho_{55} = \frac{2\,y_0}{y}, \qquad i \neq 5$$

Superposition of reflections and transmissions from all input ports gives rise to a scattering matrix[8] for the stub loaded shunt node of

$$\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_S \end{bmatrix}^r = \frac{1}{y} \begin{bmatrix} 2-y & 2 & 2 & 2 & 2\,y_0 \\ 2 & 2-y & 2 & 2 & 2\,y_0 \\ 2 & 2 & 2-y & 2 & 2\,y_0 \\ 2 & 2 & 2 & 2-y & 2\,y_0 \\ 2 & 2 & 2 & 2 & y_0-y \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_S \end{bmatrix}^i \qquad (1.9)$$

where V_S is the capacitive stub voltage and $y = 4 + y_0 + g_0$. y_0 is the capacitive stub normalised admittance and g_0 is the loss stub admittance. This can be reduced to an explicit form thus,

$${}_{k+1}V_n^r = \left\{ \frac{2}{y} \left[\sum_{m=1}^{4} {}_k V_m^i + y_{0k} V_S^i \right] \right\} - {}_k V_n^i \qquad (1.10)$$

Although the explicit form is less frequently quoted it offers an alternative to matrix calculation.

The stub loaded node of ***figure 1.6*** represents a medium of

$$\varepsilon_r \equiv 1 + \frac{y_0}{4}$$

It is clear that if no stubs are present ***equation 1.10*** reduces to the standard two dimensional scattering equation, ***equation 1.5***. Note that no alteration is required to the connect process as the capacitive stub voltage is returned to the same node in the next iteration and the loss stub voltage is absorbed. ***Equation 1.6*** still holds, with the additional term

$${}_{k+1}V_s^i(x, y) = {}_k V_s^r(x, y)$$

The introduction of stubs modifies the values of the attenuation and propagation constants for the TLM mesh[8]. This alters the form of the axial dispersion characteristic of the mesh, which becomes dependent upon y_0. Diagonal propagation remains frequency independent.

A mesh of series nodes may be loaded with inductive or permeability stubs of normalised impedance z_0 and a lumped element series resistance r_0. The properties and scattering matrix of the stub loaded series node may be derived in a similar manner to the above shunt node properties. y_0 is replaced by an impedance z_0, and y is replaced by $z = 4 + z_0 + r_0$. The scattering matrix has a similar form to that of ***equation 1.9***.

In a series node mesh the interface between regions of differing permittivity is treated as an anisotropic, partially reflective boundary. The reflection and transmission coefficients are determined by the ratio of the intrinsic impedances of the two regions. As with the boundaries of objects within the mesh the boundaries between media of different parameters are assumed to lie at a distance $\Delta l/2$ from the nodes. The same principles apply to the boundaries between regions of varying permeability in a shunt node mesh.

1.4 The Three Dimensional TLM Method

Two dimensional models are widely used in acoustic simulation, however in electromagnetics 2D problems are of limited importance. Most practical EM applications for TLM require the modelling of propagation within a three dimensional (3D) medium. A 3D TLM mesh is formed from a 3D Cartesian array of nodes. Each node must be capable of modelling 3 electric and 3 magnetic field components. This was originally accommodated by creating a 3D node from a combination of 2D shunt and series nodes[9]. In this distributed node the electric and magnetic field parameters are spatially distributed among the shunt and series nodes. Electric and magnetic walls must therefore be defined at different locations within the mesh to maintain their positioning $\Delta l/2$ from the nodes representing the relevant field components. To overcome these limiatations the distributed node was modified to model all field components at a single point, yielding the asymmetric condensed node[10]. This 3D node brings together the connections between the series and shunt nodes at a single point.

A considerable step forward in 3D modelling was provided in the form of the symmetrical condensed node (SCN)[11], ***figure 1.7***. Unlike the shunt and series nodes in which the scattering parameters are developed from equivalent circuits, the scattering matrix of the SCN was formed from a consideration of field parameter conservation within the region modelled by the node.

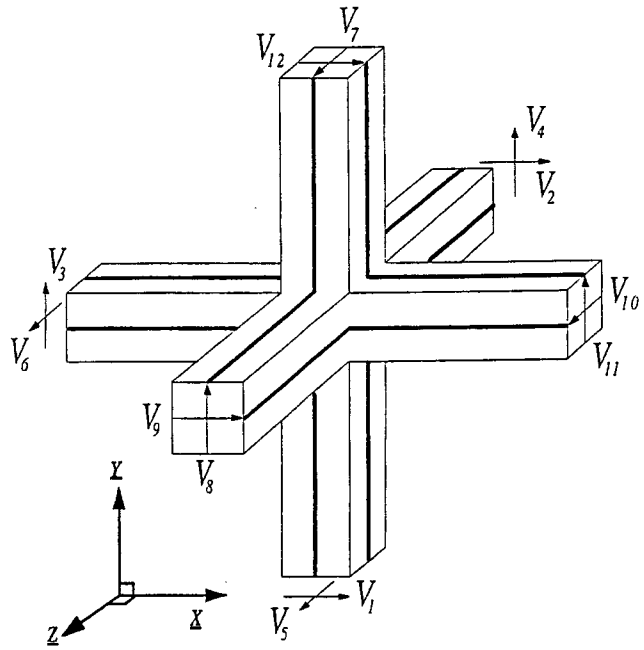


Figure 1.7 - The Symmetrical Condensed Node

This gives rise to a sparse 12 x 12 scattering matrix, ***equation 1.11***[11]. The 12 ports of the SCN are arranged such that 2 cross-polarised ports are presented at each of the 6

$$\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \\ V_9 \\ V_{10} \\ V_{11} \\ V_{12} \end{bmatrix}^r = \frac{1}{2} \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & -1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \\ V_9 \\ V_{10} \\ V_{11} \\ V_{12} \end{bmatrix}^i \qquad (1.11)$$

and

$$I_k = \frac{V^i_{ipj} - V^i_{inj} + V^i_{jni} - V^i_{jpi}}{Z_{ij} + Z_{ji}}$$

The characteristic impedance of a link line is given by

$$Z_{ij} = \sqrt{\frac{L_{ij}}{C_{ij}}}$$

It can further be shown[14] that in the case of a uniform mesh in which $\Delta x = \Delta y = \Delta z = \Delta l$ only two characteristic impedance parameters are required.

$$\begin{aligned} Z_{xy} = Z_{yz} = Z_{zx} = Z_p = \frac{Z}{A_1} \\ Z_{zy} = Z_{xz} = Z_{yx} = Z_n = Z\,A_1 \end{aligned} \tag{1.16}$$

Where Z is the characteristic impedance of the background medium

$$Z = \sqrt{\frac{\mu}{\varepsilon}}$$

and the parameter A_1 is given by

$$A_1 = \sqrt{\varepsilon_r \mu_r} \pm \sqrt{\varepsilon_r \mu_r - 1}$$

It is clear from the above that in the case of a background medium where $\varepsilon_r = \mu_r = 1$ the parameter $A_1 = 1$ and $Z_p = Z_n = Z_0$. The SSCN therefore reduces to the basic twelve port SCN.

1.5 Computational Efficiency in TLM

As TLM is applied to increasingly complex problems the run times on serial computers become prohibitive[17]. Serial computers are limited to processing a single operation at a time. Any increase in the number of operations required to perform a given function leads to a corresponding increase in the processing time required. Hence the introduction of more complex algorithms for inhomogeneous media, more

effective boundary conditions[18,19], graded meshes[20] and other refinements[e.g.21] to the basic scatter and connect techniques increase the time taken for each scatter or connect operation. The move to larger meshes, particularly in 3D, and finer meshes for studying more detailed phenomena increase the number of nodes and therefore the number of times the scatter-connect processes must be applied in each iteration. In order to maintain the usefulness of TLM when applied to complex models the run times must be reduced.

1.5.1 Signal Processing Techniques

The number of iterations required to obtain a solution can be reduced through the use of digital signal processing (DSP) techniques. The most commonly used in TLM are the Prony-Pisarenko method[22,23] (PPM) and the Auto-Regressive Moving Averages (ARMA)[22] technique. Both of these reduce the number of iterations required to achieve a solution to a given accuracy by interpolating future responses from knowledge of the previous behaviour of the system. Both may be used to good effect when frequency domain results are required, as they require fewer samples to obtain a solution than the traditional Fourier transform.

A wide range of high throughput DSP chips (e.g. C40, Sharc) is available. Each has an internal architecture optimised for high speed arithmetic. Such processors may offer a significant increase in performance for TLM applications[24]. However these processors are still constrained by the use of a serial internal architecture. While arithmetic instructions may be executed faster than on a PC, the same limitation of processing a single instruction at a time exists. In most cases, as with a PC, the instruction set of these processors is over specified for the requirements of a TLM processor.

1.5.2 Parallel Computing and TLM

The explicit nature of the scatter and connect processes allow them to be performed at all nodes simultaneously. For concurrent scatter or connect operations performed at any pair of nodes the input (V^i) and output (V^r) data sets for those operations satisfy Bernstein's Condition

$$\left(V_x^i \cap V_x^r\right) \cup \left(V_x^i \cap V_y^r\right) \cup \left(V_x^r \cap V_y^r\right) = 0$$

This states that the output of any given node within a single iteration is independent of the output of any other node within that iteration. One way to take advantage of this inherent parallelism would be to implement TLM on a parallel computer. This would produce an increase in computational performance as the scatter and connect processes could be applied simultaneously to the entire array in each iteration. Amdahl's law[25] may be used as an indicator of how suitable an algorithm is for parallel implementation, based on the fact that any operations that must be performed serially within an algorithm will limit the maximum speed up achievable through parallel execution. Thus

$$S \leq \frac{1}{f + \frac{(1-f)}{p}} \tag{1.17}$$

where S is the achievable speed up,
f is the fraction of operations that must be performed in serial, $0 \leq f \leq 1$
p is the number of processors in the parallel architecture

In TLM both the scatter and connect routines may be performed in parallel, i.e. $f = 0$. Amdahl's law then reduces to $S \leq p$, the only limit on the achievable speed up is the number of processors available. This implies that TLM is highly suited to parallel implementation. This equation, however, does not take in to account the limitations introduced through the choice of parallel architecture, or operations such as reading output data, which will increase f and therefore reduce S.

1.5.2.1 Parallel Architectures

The classification of parallel computers according to Flynn[26] places architectures in to one of four groups depending upon the number of instruction and data sets acted upon concurrently.

SISD - Single Instruction - Single Data. SISD computers have one processor and can only perform one instruction on a single set of data at any time. This category includes PCs, workstations and similar single processor systems.

SIMD - Single Instruction - Multiple Data. SIMD systems can perform a single instruction simultaneously upon many data sets, thus a single set of instructions need only be applied once to process a whole array of data. SIMD machines have difficulty executing conditional statements as they often lead to the formulation of multiple instruction streams.

MISD - Multiple Instruction - Single Data. This is the least practical and least common of the classifications. An MISD machine is capable of performing many instructions simultaneously upon the same data. Each instruction operates on a copy of the same data and is therefore incapable of affecting the results of the other processors.
MIMD - Multiple Instruction - Multiple Data. MIMD machines are the most powerful of all the classifications, as they are capable of executing multiple programs simultaneously.

Each of the basic types is illustrated graphically in ***figure 1.8***.

There are two other common architectures that do not fit readily in to Flynn's classification, the systolic array processor and the pipeline processor. In a pipeline processor a task is divided in to *P* subtasks, each of which is assigned a processing element (usually SISD). The output of one processor forms the input to the next, thus partial results are passed between the processors in a manner analogous to a production line. All processors are active concurrently thus improving throughput so long as the pipeline is continually fed new data.

A systolic array, from Johnson et al[27], is 'a grid like structure of special processing elements that processes data much like an n-dimensional pipeline. Unlike a pipeline, however, the input data as well as partial results flow through the array. In addition data can flow in a systolic array at multiple speeds and in multiple directions. Systolic arrays usually have a very high rate of I/O and are well suited for intensive parallel operations.' This class of systolic array may be termed data systolic to

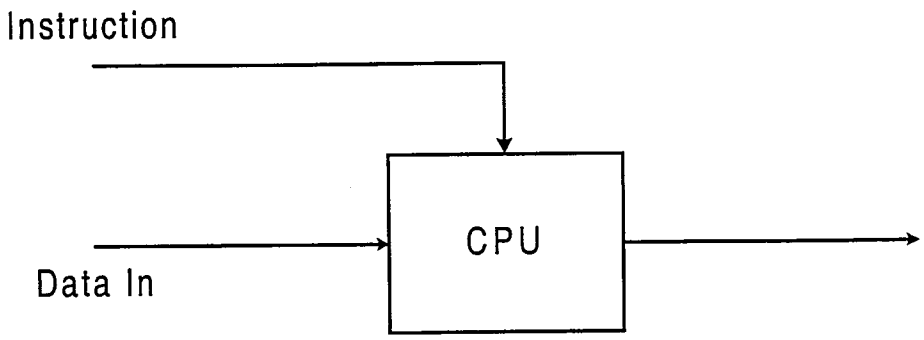


Figure 1.8a Single Instruction Stream, Single Data Stream (SISD) Architecture

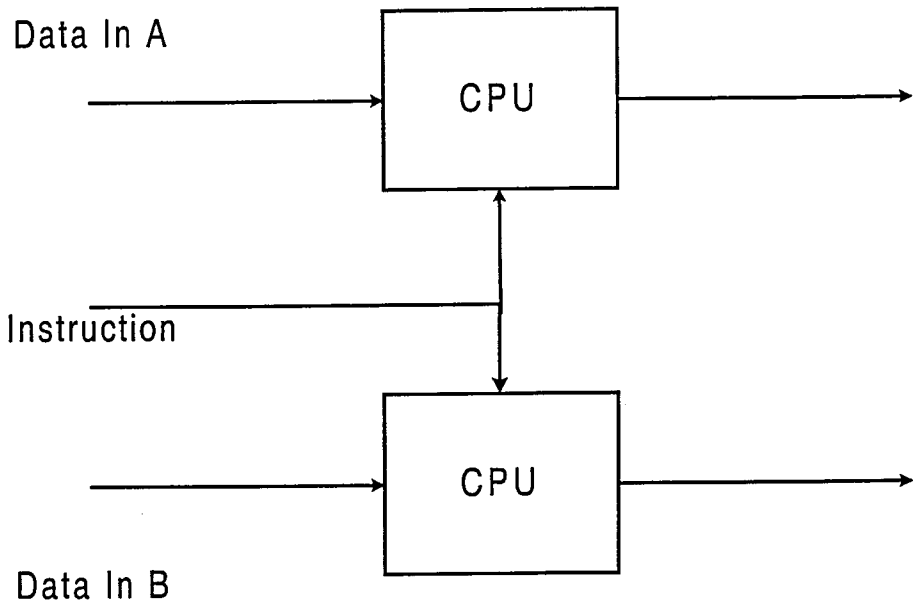


Figure 1.8b Single Instruction Stream, Multiple Data Stream (SIMD) Architecture

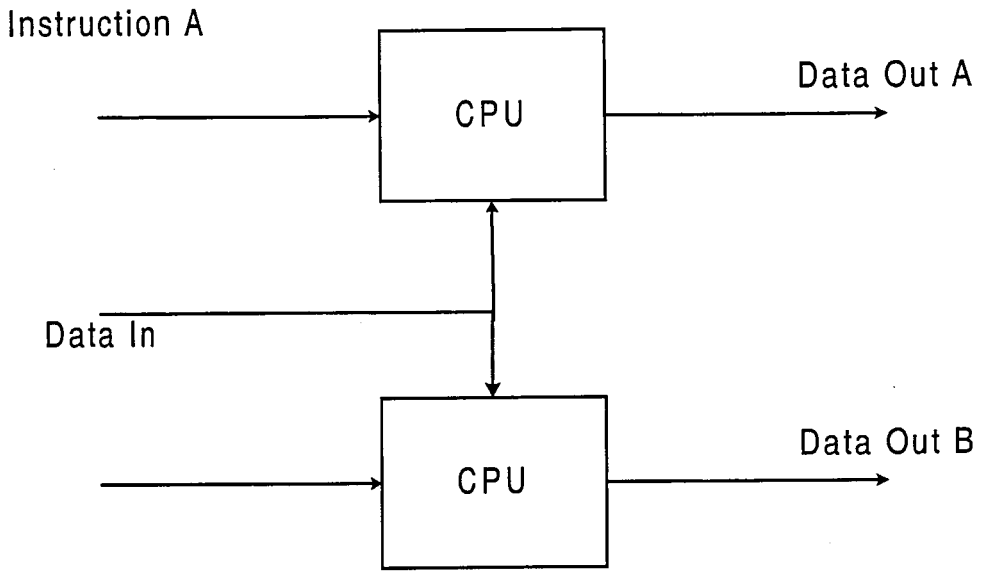


Figure 1.8c Multiple Instruction Stream, Single Data Stream (MISD) Architecture

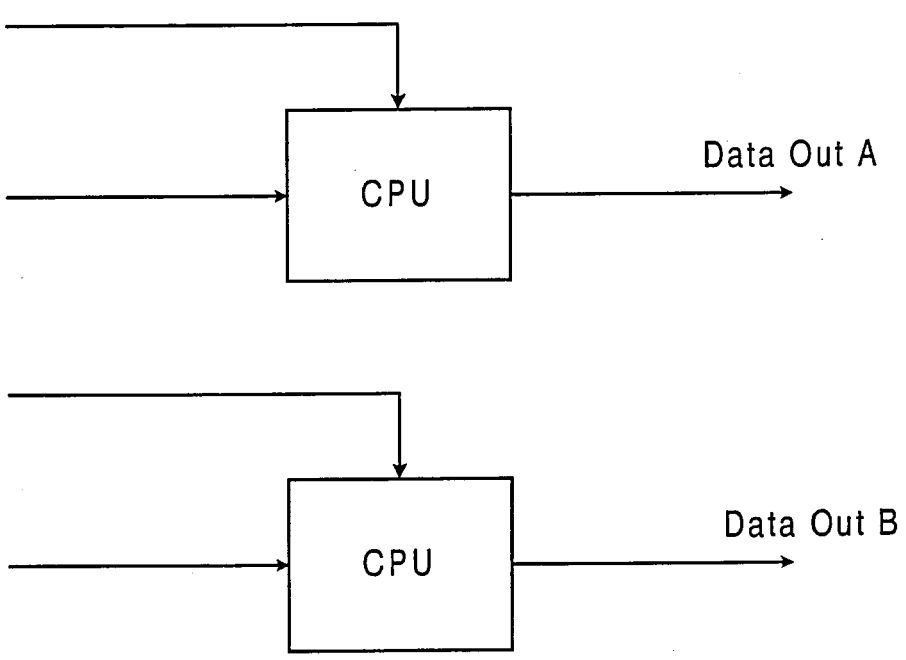


Figure 1.8d Multiple Instruction Stream, Multiple Data Stream (MIMD) Architecture

differentiate it from a more recent development in parallel processing, the instruction systolic array processor (ISP). In the ISP streams of instructions flow through an array of static data. A second stream of flags controls the application of the instructions to the required data locations. The ISP has the advantage that, unlike data systolic arrays, the instruction performed is independent of the location within the data array. This leads to more efficient coding and lower redundancy (ie. less inactive processors). This efficiency is gained at a price. Each processing element must be capable of performing a whole set of instructions. ISPs therefore generally require more complex processing elements than data systolic arrays.

1.5.2.2 Parallel Implementations of TLM

The idea of applying TLM on a parallel computer has been investigated by a number of groups in recent years. ***Table 1.1*** gives a summary of the methods used and results produced through these applications.

So *et al.*[28],[29] utilised the DECmpp12000 SIMD parallel computer and a number of HP workstations to implement both 2D and 3D TLM. A combination of parallel computing and signal processing were used to achieve a speed up factor of 34. However the low data transfer rate between the DECmpp12000 and the DEC5000/200 front end limited the speed up factor to 18 when a c.w. input was injected in to a single point in the mesh. Three dimensional arrays were formed by creating arrays of data at each processing element.

Dubard *et al.*[16] implemented TLM on the MIMD architecture of the CM-3 connection machine. A three dimensional stub loaded SCN mesh was created. The connect architecture of the CM-3 allowed a three dimensional mesh to be mapped directly on to the processor array. A speed up factor of 70 was achieved. Results were compared to a 20 MHz 80386 processor

Luthi *et al.*[30] tested the efficiency of TLM on a variety of parallel computers incorporating both SIMD and MIMD architectures. The results were used to derive an empirical formula linking execution times to the ratio of calculation time to communications time.

$$t_p = \frac{a\,n^2}{p} + b_n \tag{1.18}$$

t_p = execution time
a = unit calculation time
n = problem size
p = no. of processors
$b_n \propto$ communications overhead
The results suggest that those systems which use direct interconnects are more efficient than those using message passing systems with large communications overhead.

Tan and Fusco[31] implemented a stub loaded shunt node mesh on the bit serial SIMD architecture of the AMT DAP510. A small mesh was used in the test as the DAP510 has only a 32 x 32 array of processing elements. Larger meshes may be processed in 32 x 32 partitions. Execution time is proportional to the number of partitions in the mesh.

Parsons *et al.*[32] implemented TLM on a distributed array of SISD Sparc workstations using PVM (parallel virtual machine) software. Various mappings of a mesh on to the array were tested in order to study the effect of the mapping on the performance of the system. In particular the ratio of computation to I/O was varied. It was shown that this ratio has a strong effect on the speed up factor achieved. The effect was exacerbated by the serial data link between the machines.

Although the techniques, hardware and system complexity varies greatly between individual applications there are a number of general conclusions that may be drawn.

- None of the processors optimally matches the granularity of the processing elements with that of the problem. In the case of the AMT DAP510 the single bit processors require much manipulation of data between the processor and local memory. In the other cases, e.g. Dubard *et al*, the processors are capable of far more complex operations than the simple calculations required for TLM, therefore much of the system's potential is unused.
- With the exception of the work of Parsons *et al.* all applications map one processor to one node in the mesh. This limits the number of nodes in the mesh to the number of processors available. So *et al.* mapped a 3D mesh to a 2D connect scheme by mapping the third dimension to 1D arrays within the local memory of each processor, thus each processor maps to one node in each plane of the model. This restricts the number of nodes in the third dimension as local memory is limited.

REFERENCE	29,30	24	31	32	33
HARDWARE	DECmpp12000 DEC5000/200 HP Workstation	Connection Machine CM-3	Various	AMT DAP510	Sun Sparc Workstations
ARCHITECTURE	SIMD	MIMD	Various	SIMD	SISD/PVM
NO. OF PROCESSORS	8k	16k (8k Used)	Various	1024 (32 * 32)	12 (12 * 1)
SCATTER*	P	P	P	P	P/S
CONNECT*	P	P	Various	P	S
SOFTWARE	MPL/MPF	C*	Various	Fortran Plus	PVM
MAPPING	1-1 (2D) 10-1 (3D)	1-1 (3D)	Various	1-1 (2D)	Various
SPEED UP	34 x Plain 18 x Gaussian Input c.f. DEC5000	~70 x c.f. 20MHz 386 PC	$T_p=an^2/p + b_n$ t_p - Exec. Time a – Unit Calc. Time n – Problem Size p – No. of Processors b_n – Comms. Overhead	~10 x c.f. VAX9000	3.5 – 8.72 x c.f. Single Sun Sparc
EFFICIENCY	0.4% - 0.2%	0.85%	-	0.97%	29.1 – 72.7%
COMMENTS	Used arrays in local memory to map 3D mesh on to a 2D connect scheme. Integrated with OSA90/hope CAD software.	Full stub loaded 3D. The CM-3 connect scheme allows 3D arrays to be mapped directly.	Results suggest that TLM runs more efficiently on SIMD machines with direct interconnect as opposed to those using message passing.	Single bit processors with dedicated near neighbour interconnect. Performance is dependent upon model dimensions.	Studied effect of mapping on performance. Message passing is a major factor, partly due to the serial link used.

* P = Parallel Operation

S = Serial Operation

Table 1.1 – Comparison of Implementations of TLM on Parallel Computers

- TLM exhibits a very high ratio of I/O to computation, therefore applications require very high data bandwidths. The equation used by Luthi *et al.* to measure performance highlights the importance of this. As a 1 to 1 mapping is used in most cases, $n^2/p = 1$ and the dominating factor becomes the communications overhead, b_n. The work of Parsons *et al.* shows the importance of limiting the data bandwidth in distributed computing. It has been shown that the effectiveness of a given distribution of a mesh between a number of workstations is inversely related to the volume of data that must be transferred after each iteration.
- So *et al.* show the effect of communications overhead when data is injected in to or extracted from the mesh. The performance of their implementation on the DECmpp12000 falls by almost 50% when a continuous, Gaussian waveform is injected in to the mesh *c.f.* a single iteration impulse.
- Amdahl's law stated that for TLM the maximum achievable speed up is equal to the number of processors used. A rough measure of the success of each implementation can be gained by comparing the speed up achieved to the theoretical maximum. The *efficiency* row of ***table 1.1*** shows the actual speed up as a percentage of the theoretical maximum. This is not an exact figure as it does not take in to account the serial operations required for I/O etc. however these form only a very small percentage of the total number of operations performed. It is also recognised that speed up is often quoted with respect to a second machine as opposed to a single node of the parallel architecture. One reason for the very low efficiency demonstrated is that restricted data bandwidth prevents I/O operations from being performed in parallel, thus the ratio of I/O to computation increases dramatically.
- Each implementation uses a different programming language and each processor has its own executable format. Therefore code is not portable between implementations.
- In all cases the hardware requirements are impractical. Parsons *et al.* used 12 Sun Sparc workstations linked as a single Parallel Virtual Machine (PVM). While it is true that a network of workstations is commonly found in educational and research establishments, it is not necessarily true that these are all available simultaneously and may be taken over for a single experiment. In all other cases some or all of the hardware used is of limited availability, costly and inaccessible to most people.

Many of the problems listed above arise due to a mismatch between the requirements of TLM and the provisions of the chosen architecture. The TLM method consists of two distinct processes, scatter and connect. Each of these has separate requirements.

Consider firstly the scatter process. Whichever form of the scattering equation is used, the process consists of the application of a scattering equation to the input data of each node in the array. This would appear to be highly suited to solution on an SIMD machine on which the same instruction is applied simultaneously to a whole array of data. However the TLM array is inhomogeneous on a computational level. Sources, receivers and boundaries all require handling differently to the rest of the array. The approach favoured on most SIMD machines is to mask out any processors to which a given instruction, e.g. the injection of an impulse, does not apply. This can reduce the performance of a system considerably as demonstrated by So *et al.* There is a need for a degree of local autonomy[33] within the mesh, some processors must have the freedom to behave differently under certain circumstances.

The connect process, the application of ***equation 1.6*** to each node in the array, results in a flow of data between neighbouring processors. A systolic approach may therefore be expected to yield results. However none of the implementations reported utilise a systolic connect strategy. Again the need for a degree of local autonomy is present when dealing with boundaries. Data transfer is often the slowest part of a system, as demonstrated by computationally intensive processes where instruction fetching can lead to bottlenecks. In many of the implementations described above data must be transferred to a slower host system for visualisation or post-processing. Flynn[19] describes [parallel computers] as 'suitable for problems characterised by a high ratio of computing requirement to I/O requirement', however in TLM this is clearly not the case. Even the scattering matrix of the stub loaded SCN can be solved using only 54 addition/subtraction operations and 12 multiplications[34]. Most large scale parallel computers are designed to perform considerably more computation per data transfer therefore in many cases the data bandwidth required can not be provided by the chosen architecture. Flynn[19] again summarises this problem by saying that for SIMD machines 'latency in the instruction stream is often replaced by latency in the data stream caused by operand communication problems'. This is demonstrated by Luthi *et al.* who show that communication costs vary in inverse proportion to the processing rate in the systems they reviewed, thus as the performance of the individual processors improves the communications overhead increases to offset some of the increase in performance. It is clear therefore that while many systems may provide the required physical interconnections for TLM they can not provide sufficient bandwidth. This will limit the rate at which the computations are performed.

The architecture of the system poses an interesting trade off between the freedom to select arbitrary mesh geometries and throughput. The use of a large array of processors, where one processor is dedicated to each node in the model, allows the whole array, or large partitions of it, to be processed concurrently. However such arrays have a restricted input/output data bandwidth. Either the number of nodes in the model must be limited to the number of processors available or much of the performance increase achieved through parallel processing is lost through slow data transfer rates between partitions of the model. The use of only a small number of processors limits the size of partition of the model that may be processed concurrently, therefore reducing throughput. However the reduction in required bandwidth as the mesh partition sizes are made smaller makes the transfer of data between partitions of the model a realistic proposition. The advantage of the latter architecture is that by processing the model in smaller partitions and transferring data between them the restrictions on the size of the model imposed by the one to one mapping of the mesh on to the processor array may be removed. However the advantages of partially parallel operation are maintained. The high efficiency ratings of the TLM implementations by Parsons *et al* confirm the viability of this type of architecture.

1.6 Application Specific Processors for TLM

Software based TLM is an iterative process. Loops of instructions apply the scatter and connect processes to each node in turn. The parallel processing methods outlined in ***section 1.5*** reduce computation times by performing the scatter and connect routines many times simultaneously, thus reducing the number of times each loop is performed to process the whole mesh. One further way to increase performance is to reduce the time taken to perform each loop. It was shown in ***section 1.5.2.2*** that poor matching of processor granularity to that of the problem leads to computational inefficiency. If the processor granularity is too low then partial results must be manipulated to yield a final value, if it is too high then time is spent fetching and decoding complex instructions. The development of an application specific processor for TLM would allow the processor granularity to be matched to that of the problem, overcoming the loss of efficiency caused by a granularity mismatch. In the long term, systems with reduced computational complexity deliver higher performance at lower cost, the success of RISC (Reduced Instruction Set Computing) based processors in a wide range of computers confirms this principle. Two application specific approaches to reducing run times in TLM have been studied.

1.6.1 Review of Existing Application Specific TLM Processors

Existing application specific processor designs for TLM fall in to two distinct categories, complete systems on to which the TLM array is mapped and single processors which are utilised by software to perform the scatter operation. The following sections evaluate and compare the two approaches.

1.6.2 Single Node Coprocessor System

Saleh developed the coprocessor approach[35]. A single RISC processor was developed around the two dimensional, stub loaded shunt node algorithm. A host system runs the main TLM code and each time a scatter operation is encountered the data is passed to the TLM coprocessor. This performs a scattering computation and returns the scattered data to the host. The TLM array is defined in software, therefore any mesh size may be accommodated dependent upon the available memory. An instruction register and several control lines provide some control over the configuration of the processor for each node. All data regarding the composition of the array and the material properties at each node are held by the host system, the only data storage on the processor is a stack for holding partial results during calculations. Performance increase is achieved by performing the scatter calculations using optimised hardware. This removes the need for instruction fetch and decode cycles for these calculations. Software running on an LSI-11 computer achieved 62 node iterations per second, where as the same software on an LSI-11 equipped with the coprocessor achieved 1670 node iterations per second, a speed up factor of 27. These results were limited by the technology of the time, the processor was built from discrete ICs with high propagation delays. The LSI-11 uses a 16 bit data bus, therefore the processor utilised 16 bit floating point data with an 8 bit mantissa and 8 bit exponent. The numerical accuracy of the processor was low due to the limitations of the 8 bit mantissa.

1.6.3 Complete System

Gregory designed a complete application specific, parallel TLM system[36]. The system was developed around an array of bit serial processing elements, which performed a basic 2D scattering operation on their input data. A number of these PEs were arranged in an SIMD array on to which the TLM mesh was mapped with one node per

PE. All integer data regarding the energy within the mesh was stored on the array. The results produced were subject to delta modulation for error stacking, thus an N bit output took 2^N clock cycles to generate. One consequence of this is that throughput reduces exponentially with word length. It was shown that any word length above 11 bits would be less efficient than a theoretical software implementation on a 100 MHz serial processor. The PE was developed through full custom IC design. Although the system was never realised it was verified through simulation, with 2 interconnected nodes being tested.

1.6.4 Comparison of Coprocessor and Complete System Approaches

The system described in ***section 1.6.2*** and that of ***section 1.6.3*** both produced a performance increase over software implementations of TLM, however both systems retain some of the limitations imposed by traditional parallel architectures.

- The use of a coprocessor design reduces the restrictions on mesh size, however it limits the achievable throughput increase as the architecture does not utilise parallel processing. Gregory's design increases throughput both by improving the efficiency of the scatter operation and introducing parallelisation.
- An array implemented using Gregory's system is limited not only in size but in configuration as the interconnections between the PEs are hardwired.
- Saleh's processor greatly reduces data bandwidth as data is passed to only one node in each transaction. Gregory also reduces bandwidth through the use of a bit serial design. However the choice of bit serial algorithm forces the use of delta modulation, greatly increasing the number of clock cycles required to process a given word length.
- Both processors are limited to implementing a single TLM scheme, 2D shunt node in the case of Gregory and stub loaded 2D shunt node in the case of Saleh. There is no room for expansion, the limited reprogrammability of Saleh's processor only allows for placement of simple open and short circuited boundaries and the specification of material parameters within the given scheme. Gregory's processor fixes both the material parameters and the location of the boundaries.

It is clear from the above that neither system is ideally suited to TLM. The development of an application specific processing element increases the efficiency of the implementation by removing redundant computational steps. However as efficiency increases, flexibility decreases and the most efficient processors are limited to implementing a single form of the TLM algorithm. Saleh's approach produced a significant performance increase given the technology available at the time. However

with modern computer systems latency in data transfer, as opposed to computation rates, has become the overriding limitation on performance. It is therefore unlikely that such a system would produce similar performance increase over modern processors due to the high number of data transfers required. There is, therefore, a necessity for some degree of parallelisation. Gregory's system demonstrates a fundamental problem with parallel architectures for TLM. The number of PEs available limits the mesh size that may be implemented. With the exception of very large scale parallel computers, the number of PEs is limited. Gregory's system was developed around the concept of a chip containing a 3x3 array of nodes. With the potential to realise 2 or 3 chips per board and 2 boards per PC, an array of at most 54 nodes is possible. No details were given as to how the chips, or the boards, would communicate. Very few models of any practical interest could be solved using an array of this size. Modern fabrication techniques have increased the number of nodes which may be realised on the same area of silicon but as the number of nodes increases so the data bandwidth and pin count for each chip also rises, rapidly becoming the most significant limitation on array size.

1.7 Conclusions

Run times for software based TLM applications on serial computers are becoming prohibitively large. More efficient software schemes offer some improvement but the serial architecture of the PC places fundamental limits on the performance that may be achieved. Applications using parallel and distributed computing to take advantage of the inherent parallelism of TLM have been investigated. However a mismatch between the architecture of the parallel computer and the requirements of the TLM algorithm limit the efficiency obtainable. The mismatch problem may be overcome through the development of an application specific processor, using either a direct mapping of the TLM algorithm in to hardware or a reduced instruction set (RISC) approach. Existing application specific processors for TLM have failed to overcome the majority of the problems associated with parallel applications. Two classes of processor architecture exist for TLM, single node processors and large arrays. Both architectures reduce computational redundancy in the processing element, however neither architecture addresses the problems associated with the connect process and the mapping of the array in to hardware/memory. Single node processors handle the connect process as a software routine where as array based processors use hard wired near neighbour interconnects in a manner similar to an SIMD or systolic array. Many

of the problems associated with parallel implementations of TLM are caused by the connect process, such as bandwidth limitations restricting data I/O rates.

From this review of parallel and distributed processing implementations of TLM it is clear that existing solutions for increasing computational rates make inefficient use of the resources available. The literature highlights a number of issues relating to the design of TLM accelerators.

- The granularity of the TLM algorithm must be successfully mapped to hardware. This means both removing redundant elements from the computational hardware and providing sufficient bandwidth for the connect process.
- The chosen architecture should not limit the processor to a single form of the TLM algorithm or a single mesh configuration.
- The chosen architecture should be scalable to allow any mesh size to be implemented.
- The processor must be accessible. That is its use should not be prohibited through
  - Portability
  - Cost
  - Programming requirements

This thesis aims to demonstrate how each of these issues may be addressed through the development of a new class of TLM processor. Concepts are introduced for the efficient mapping of the scatter and connect routines in to hardware. These concepts are developed to give a description of a complete hardware based TLM accelerator. One key measure of the success of the new processor is its efficiency score, as defined in *Table 1* for previous implementations. However success must also be measured against resolving the above issues and producing an increased processing rate with respect to software based implementations of TLM.

The brief for the implementation of the new class of TLM processor is very open. A clear implementation strategy is therefore required to ensure a successful outcome. The strategy must break the work down in to a logical sequence of goals, the success of which may be measured by attaining certain milestones. There are many extensions to the TLM method, however it is not possible in the time scale of this thesis to address them all. The scope of the thesis is limited to the four main node schemes identified in this chapter. These are the shunt node, the stub loaded shunt node, the SCN and the SSCN. Between them they offer the ability to process a wide variety of media in both two and three dimensions. The implementation strategy for this project

is shown below.

OBJECTIVE	MILESTONE
1. Feasibility study	Proof that an application specific processor can achieve a higher throughput than existing computers
2. Definition of suitable design flow	Selection of hardware and software required for implementation
3. Development of a shunt node processor	Demonstration of a working shunt node scatter processor
4. Implementation of connect function	Demonstration of a parallel array of shunt node processors
5. Extension of scatter processor to other TLM schemes (stub loading, SCN, SSCN)	Demonstration of working processors for the four main TLM schemes

References

[1] Kron, G 'Equivalent Circuit of the Field Equations of Maxwell', Proc. IRE, Vol.32, pp.289-99, 1944

[2] 'Numerical Techniques for Microwave and Millimeter-Wave Passive Structures', ed. T. Itoh, J. Wiley & sons, New York, 1989

[3] Johns, P.B and Beurle, R.L 'Numerical Solution of 2-Dimensional Scattering Problems Using a Transmission-Line Matrix', Proc. IEE, Vol.118(9), pp.1203-08, 1971

[4] Dunlop, J and Smith, D.G 'Telecommunications Engineering, 2nd Edition', Chapman and Hall, London, 1989

[5] Coates, R; deCogan , D and Willison, P.A 'Transmission Line Matrix Modelling Applied to Problems in Underwater Acoustics', Proc. IEEE OCEANS '90, 24 September 1990

[6] Johns, P.B 'The Transmission Line Matrix Method of Waveguide Analysis' PhD Thesis, University of Nottingham, 1973

[7] Johnson, H and Graham, M 'High Speed Digital Design', Prentice Hall, 1993

[8] Johns, P.B 'The Solution of Inhomogenous Waveguide Problems Using a Transmission Line Matrix', IEEE Trans. On Microwave Theory and Techniques, Vol.22(3), pp.209-215, 1974

[9] Akhtarzad, S and Johns, P.B 'Solution of 6 Component Electromagnetic Fields in Three Space Dimensions and Time by the TLM Method', Electronics Letters, Vol.10(25), pp.535-537, 1974

[10] Amer, A 'The Condensed Node TLM Method and its Application to Transmission in Power Systems', PhD Thesis, University of Nottingham, 1980

[11] Johns, P.B 'New Symmetrical Condensed Node for the Three Dimensional Solution of Electromagnetic Wave Problems by TLM', Electronics Lett. Vol.22, pp.162-64, 1986

[12] Krumpholz, M and Russer, P 'On the Dispersion in TLM and FD-TD', IEEE Trans. On Microwave Theory and Techniques, Vol.42(7), pp.1275-79, 1994

[13] Tong, C.E and Fujino, Y 'An Efficient Algorithm for Transmission Line Matrix Analysis of Electromagnetic Problems Using the Symmetrical Condensed Node', IEEE Trans. On Microwave Theory and Techniques, Vol.39(8), pp.1420-1424, 1991

[14] Trenkic, V 'The Development and Characterisation of Advanced Nodes for the TLM Method', PhD Thesis, University of Nottingham, 1995

[15] Trenkic, V; Christopoulos, C and Benson, T.M 'New Symmetrical Super-Condensed Node for the TLM Method', Electronics Letters, Vol.30(4), pp.329-30, 1994

[16] Trenkic, V; Christopoulos, C and Benson, T.M 'Theory of the Symmetrical Super Condensed Node for the TLM Method', IEEE Trans. On Microwave Theory and Techniques, Vol.43(6), pp.1342-48, 1995

[17] Lawton, S; Ward, D.D; Cloude, S.R and Dawson, J.F 'Hybrid Time Domain Modelling for Automotive EMC', IEE 2nd Int. Workshop on Computation in Electromagnetics Digest, pp.275-78, 1994

[18] Morente, J.A; Porti, J.A and Khalladi, M 'Absorbing Boundary Conditions for the TLM Method', IEEE Trans. On Microwave Theory and Techniques, Vol.40(11), pp.2095-2099, 1992

[19] Simons, N.R.S and Bridges, E 'Method for Modelling Free Space Boundaries in TLM', Electronics Lett. Vol.26(7), pp.453-455, 1990

[20] Al-Mukhtar, D.A and Sitch, J.E 'Transmission Line Matrix Method with Irregularly Graded Space', IEE Proc. Vol.128(6), part H, pp.299-305, 1981

[21] Duffy, A.P; Herring, J.L; Benson, T.M and Christopoulos, C 'Improved Wire Modelling in TLM' IEEE Trans. On Microwave Theory and Techniques, Vol.42(10), pp.1978-1983, 1994

[22] So, P.P.M; Eswarappa, C and Hoefer, W.J.R 'Parallel and Distributed TLM Computation with Signal Processing for Electromagnetic Field Modelling', Int. Jnl. Num. Mod : Elect. Networks, Devs. And Fields, Vol.8, pp.169-185, 1995

[23] Dubard, J.L; Benevello, O; Pompei, D; Le Roux, J; So, P.P.M and Hoefer, W.J.R 'Acceleration of TLM Through Signal Processing and Parallel Computing', Int. Conference on Computation in Electromagnetics, IEE, pp.71-74, 1991

[24] Stothard, D 'The Implementation of TLM on a Fast DSP Processor', Awaiting Publication, 1999

[25] Swaminathan, M; Sarkar, T.K 'A Survey of Various Computer Architectures for Solution of Large matrix Equations' Int. Jnl. Num. Mod : Elect. Networks, Devs. And Fields, Vol.8, pp.153-168, 1995

[26] Flynn, M.J 'Very High Speed Computing Systems', Proc. IEEE Vol.54(12), pp.1901-1909, 1966

[27] Johnson, K.T; Hurson, A.R and Shirazi, B 'General Purpose Systolic Arrays', IEEE Computer pp.20-31, Nov. 1993

[28] So, P.P.M and Hoefer, W.J.R 'Optimization of Microwave Structures using a Parallel TLM Module', Progress in Applied Computational Electromagnetics, 10th Annual Review, pp.546-53, 1994

[29] So, P.P.M; Eswarappa, C and Hoefer, W.J.R 'Transmission line Matrix Method on Massively Parallel Processor Computers', Progress in Applied Computational Electromagnetics, 9th Annual Review, pp.467-74, 1993

[30] Luthi, P.O; Chopard, B and Wagen, J-F 'Wave Propagation in Urban Microcells : a Massively Parallel Approach Using the TLM Method', Applied Parallel Computing in Physics, 2nd International Workshop, pp.408-18, 1995

[31] Tan, C.C and Fusco, V.F 'TLM Modelling Using an SIMD Computer', Int. Jnl. Num. Mod : Elect. Networks, Devs. And Fields, Vol.6, pp.299-304, 1993

[32] Parsons, P.J; Jaques, S.R, Pulko, S.H and Rabhi, F.A 'TLM Modelling Using Distributed Computing', IEEE Microwave and Guided Wave Letters, Vol.6(3), pp.141-42, 1996

[33] Li, H and Stout, Q.F 'Reconfigurable SIMD Massively Parallel Computers', Proc. IEEE, Vol.79(4), pp.429-43, 1991

[34] Trenkic, V; Christopoulos, C and Benson, T.M 'Efficient Computation Algorithms for TLM', 1st Int. Workshop on Transmission Line Matrix (TLM) Modelling : Theory and Applications, pp.77-80, University of Victoria, 1-3 Aug., 1995

[35] Saleh, A.H 'A Dedicated Processor For Solving TLM Field Problems', PhD Thesis, University of Nottingham, 1982

[36] Gregory, S 'Design of a Single Bit Processor for TLM Using Full Custom IC Design', Dissertation (BEng), University of Nottingham, 1989

2 - Digital Arithmetic Systems Design

2.1 Introduction

Much of the performance increase gained through the use of application specific computing results from a reduction in the number of redundant computational elements within the processor. In order to efficiently map the TLM algorithm to hardware an understanding must be gained of the way in which arithmetic operations are performed on binary data. This chapter presents a review of binary arithmetic and discusses the development of digital arithmetic systems from the perspectives of processing rate and error minimisation. The specific issue of quantisation noise in discrete systems is raised.

2.2 An introduction to Digital Arithmetic

There are four principle operations in the familiar decimal arithmetic system. They are addition, subtraction, multiplication and division. The multiplication/division of two long numbers is commonly broken down into a sequence of smaller multiplication/division operations with addition or subtraction of the partial products used to generate a final sum. More complex functions such as square roots or calculus may generally be approximated to any given accuracy by iterating algorithms composed from these four main operations[1].

2.2.1 Number Representation in the Binary System

In a digital computer numbers are stored as sequences of binary digits, either 0 or 1. In the same way as decimal arithmetic uses columns to represent units (10^0), tens (10^1), hundreds (10^2) etc., so the binary system uses columns representing 2^0, 2^1, 2^2 etc. with the least significant bit (l.s.b) written on the right. Thus the binary number 10011_2 has the decimal equivalent $1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 16 + 2 + 1 = 19_{10}$. Fractional numbers are formed by extending the number to the right beyond the fractional point[ϖ]. These columns then represent 2^{-1}, 2^{-2}...$2^{-\infty}$. For example the binary

[ϖ] As we are not dealing with decimal representations the term decimal point is inappropriate. The term fractional point will be used instead for the delimiter between the integer and fractional parts of a number.

number 1011.1001_2 has the decimal equivalent 11.5625_{10}.

The negative form of a number is defined as that value which must be added to a number to obtain a sum of 0. Negative numbers may be represented in a variety of ways, the most common of which are discussed below.

- *Sign and Magnitude* - The magnitude of the number is represented in standard binary format as discussed above. An extra bit, the sign bit, is included at the most significant (left hand) end of the number. If the sign bit is 0 the number is positive, if the sign bit is a 1 the number is negative.
- *Ones Complement* - Negative numbers are formed by inverting the bits in the positive number.
- *Twos Complement* - To form the negative of a number in 2s complement notation all the bits of the number are inverted and 1 is added to the result. The 2s complement system works on the principle of using fixed word lengths to represent numbers. Any overflowing bits are ignored. 2s complement notation has the advantage that the sign bit is automatically maintained during calculations. This is unlike sign and magnitude, where the signs of the operands must be compared to create the correct sign for the result.

2.2.2 Fractional Data Representation

There are two commonly used methods for representing fractional data in binary arithmetic. The simplest uses a fixed word length similar to the integer notation used above. It is noted that an integer is simply a number in which the fractional point lies at the extreme right of the number. Using a fixed length representation the fractional point may be implied at any location. So long as all numbers use the same fixed reference point notation the simple arithmetic rules used above may be used to perform calculations. The term 'fixed point' is often used with reference to a numbering scheme in which the fractional point lies at the extreme left of the number. In this case the number is wholly fractional.

A more complex, but more flexible representation is floating point notation. This uses two numbers to represent a single number in the form $A = a2^e$. a is the mantissa and e is the exponent. A negative exponent implies a wholly fractional number. The main advantage of floating point is that it is able to represent a much broader range of numbers than integer or fixed point. However the hardware requirements of floating point processors are greatly increased as both mantissa and exponent are involved in

calculations[1]. Floating point representation does not represent the panacea of numerical inaccuracy. The range of numbers that may be represented is greatly increased through the addition of the exponent. However the precision to which each number may be represented is still limited by the number of bits available to the mantissa. Consider a 32 bit fixed point number and a 32 bit floating point number. The floating point number is able to express a larger range of values. However in some cases the fixed point number can represent a given value to a higher accuracy as only some of the bits of the floating point number (typically 24 for a 32 bit number) are given to expressing the mantissa.

2.2.3 Block Floating Point Representation

The range of fixed point numbers of a given word length may be extended using a block floating point scheme. This scheme applies a single exponent value to a whole array of fixed point data. Should an overflow occur the exponent may be incremented and the whole array right shifted by one bit. Similarly the exponent may be decremented and the whole array left shifted, for example to increase the number of significant bits stored from the result of a calculation. This scheme is commonly used for calculations such as Fourier transforms of large arrays[14], where data growth can be significant. Because the exponent is the same for all data in the array there is no need for the comparison and shifting operations required in true floating point. A block floating point processor may therefore make use of simple fixed point arithmetic hardware. Additional hardware is required to detect overflow or loss of significant bits and shift the data array when necessary.

2.2.4 Rules for Binary Arithmetic

In binary arithmetic there are two principle functions, addition and subtraction. A diverse set of algorithms exists for multiplication and division, which will be discussed later. These are mostly formed from addition and subtraction operations and basic logic functions.

2.2.5 Addition

The addition of two binary digits has a maximum value of 10_2, hence both a sum and carry output are required. These are defined by the truth tables in ***table 2.1(a,b)***.

Input A is read across the top row of the table, input B is read from the first column of the table. The intersection of the selected row and column provides the output.

+	0	1
0	0	1
1	1	0

(a)

+	0	1
0	0	0
1	0	1

(b)

Table 2.1 - Sum (a) and Carry (b) Function Truth Tables for a Binary Adder

It is clear from above that the sum function is formed from the exclusive OR (symbol ⊕) of the inputs while the carry is formed from the logical AND of the inputs. A further input formed by the carry out from a previous adder stage can also be introduced to provide a 'full adder'. The full adder is defined by the logic equations

$$\text{Sum} = (A \oplus B) \oplus C_{in}$$
$$\text{Carry} = (A \,.\, B) + (A \,.\, C_{in}) + (B \,.\, C_{in})$$

Chains of full adders may be formed by linking the carry out of each adder to the carry in of the adder whose inputs form the next most significant bit of the sum. This basic parallel adder, a section of which is shown in ***figure 2.1***, is known as a ripple carry adder. As the calculation of each bit of the output depends upon the carry bit from the

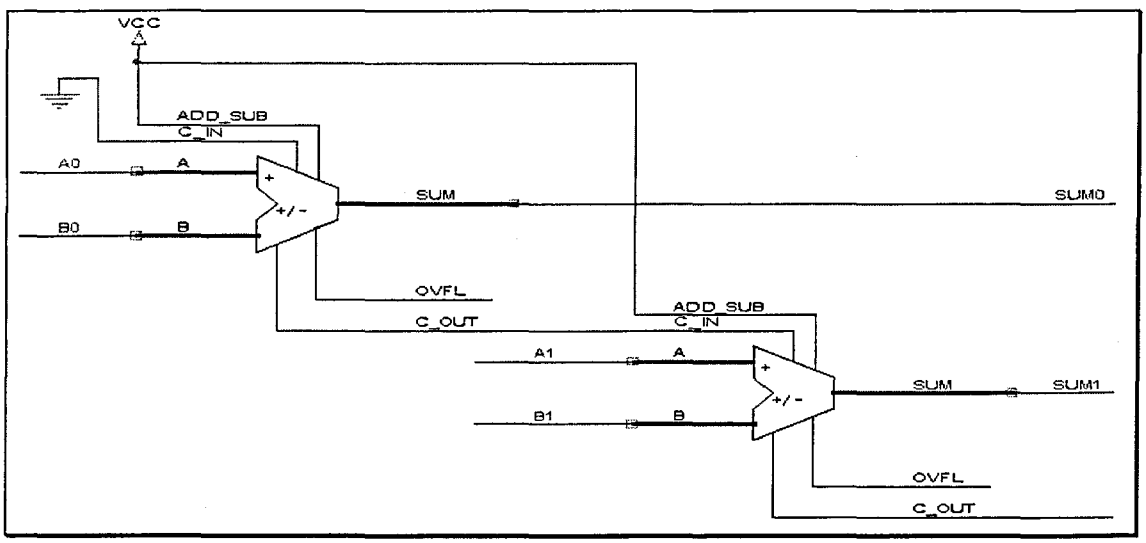


Figure 2.1 - First Two Stages of a Ripple Carry Parallel Adder

previous stage the sum ripples through the chain of adders from l.s.b to m.s.b. This type of parallel adder is inefficient as only one bit of the sum is actively being produced at any one time. For a chain of N adders, each with a propagation delay of t_p ns, the ripple carry adder may take up to $N.t_p$ ns to produce a sum.

The ripple carry adder may be improved through the use of a look ahead system[1]. Consider two functions, G_i and P_i, derived from the A and B inputs of a full adder and defined by the truth table

A	B	G_i	P_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

G_i - carry generate - is true when the inputs to the adder guarantee the generation of a carry output.
P_i - carry propagate - is true when C_{in} is propagated as an identical C_{out}.

The full adder may thus be defined by the logic equations

$$\text{Sum} = P_i \oplus C_{in}$$
$$\text{Carry} = G_i + (P_i \,.\, C_{in})$$

The carry function may be expanded for each bit in the adder as

$$C_1 = G_0 + (P_0 \,.\, C_0)$$
$$C_2 = G_1 + P_1 \,.\, (G_0 + (P_0 \,.\, C_0)) \text{ etc.}$$

From the preceding table G_i and P_i may be substituted by

$$G_i = A_i \,.\, B_i$$
$$P_i = A_i \oplus B_i$$

Yielding the final result

$$C_1 = A_0 \,.\, B_0 + A_0 \,.\, C_0 + B_0 \,.\, C_0$$
$$C_2 = A_1 \,.\, B_1 + A_1 \,.\, A_0 \,.\, B_0 + A_1 \,.\, A_0 \,.\, C_0 + \ldots \text{ etc.}$$

As long as the initial input to the carry chain is known all further values may be derived. Although the logic for producing the carry bits for the more significant bits of the sum can be quite involved it is generally faster than the ripple carry method.

2.2.6 Subtraction

Binary subtraction is most commonly performed by making use of the relationship

$$A - B \equiv A + (-B)$$

It has already been shown that there are several ways of representing negative numbers in binary. The 2s complement method is most commonly used as it does not require any extra processing to generate the correct sign bit in the result. 2s complement subtraction is performed by inverting all the bits in the number to be subtracted and setting the initial input to the carry chain to 1. The two inputs are then added using either a ripple carry or look ahead carry adder as described above. A fixed word length must be maintained. If one operand is shorter than the other it may be made up to the same length by adding copies of the sign bit to the left (m.s.b) end of the number.

2.2.7 Multiplication

Binary multiplication algorithms may take many forms[1]. The simplest form requires a combination of addition and shift operations. Consider the calculation

$$\begin{array}{r} 5.0 \\ 4.0\times \\ \hline 20.0 \end{array}$$

which in binary becomes

$$\begin{array}{r} 101.0 \\ 100.0\times \\ \hline 10100.0 \end{array}$$

Two registers are required to store the multiplicand (MD) and the multiplier (M). A double length register initially holds M in its lower half and a partial product (PP), initially set to zero, in its upper half. If the least significant bit of the double length

register is a 1 the value of MD is added to PP, maintaining a fixed word length, otherwise 0 is added to PP. The value stored in the double length register is then shifted right using an arithmetic shift i.e. maintaining the sign of PP. This add and shift routine is repeated *W* times for a *W* bit long multiplier following which the double length register holds the product of MD and M. The double length register accumulates the sum of the partial products formed by multiplying MD by each bit in M. The product of two numbers of arbitrary lengths *m* and *n* will have a maximum length $m + n$.

While the above method is simple and functionally correct it can be slow, particularly where both multiplier and multiplicand contain many bits. Faster multipliers may be formed using asynchronous switching networks or large arrays of adders[1]. However these require considerably more logic resources. With the advancement of very large scale integration (VLSI) techniques, array multipliers have become more feasible. However other options for decreasing the computation time for multiplication operations are available.

As shown above, a multiplication may be considered as a sequence of smaller multiplications with accumulation of the partial results used to form the product. Consider the binary multiplication 110 x 111. This may be written

$$(2^2 \times 2^2) + (2^1 \times 2^2) + (2^2 \times 2^1) + (2^1 \times 2^1) + (2^2 \times 2^0) + (2^1 \times 2^0)$$

A cursory examination of the above reveals the product $(2^2 \times 2^1)$ occurs twice. Where this happens the multiplication need only be performed once and the partial product left shifted one place before addition. In longer calculations where many partial products may be repeated this technique can provide a considerable reduction in processing time. Any partial product that is repeated 2n times may be performed once and the result left shifted n places prior to addition. Similar systems may be derived by performing single operations wherever consecutive groups of bits form a given pattern. The aim of all these schemes is to improve efficiency by reducing the number of operations performed.

Many other, more complex, schemes also exist. Their inclusion is beyond the scope of this thesis, however references[1,2] are provided which offer a starting point for anyone wishing to research the subject in more depth.

2.2.8 Division

Binary division in its simplest form has two variations, restoring and non-restoring[1]. Restoring division uses a technique similar to decimal long division. The divisor is aligned with the m.s.b of the dividend and subtracted. If the result is positive the m.s.b of the quotient is set to 1. If the result is negative the m.s.b of the quotient is set to 0 and the divisor is added to the dividend to restore it to the value it held prior to the subtraction. In both cases the divisor is then shifted one place to the right w.r.t the dividend and the process is repeated to produce subsequent bits of the quotient. Following the subtraction (and restoration as appropriate) performed when the least significant bits of the divisor and dividend are aligned the quotient is resolved and the divisor holds the value of any remainder. In non-restoring division if subtraction results in a negative dividend the quotient bit is set to 0 and in the next cycle the divisor is added to the dividend. In longer calculations the non-restoring method offers a significant reduction in the number of operations performed. As with multiplication, division may be performed faster by using large parallel switching networks.

2.2.9 Bit Serial Binary Arithmetic

Thus far the discussion of binary arithmetic has assumed that all bits of all arguments of an operation are available prior to calculation. This is not always the case, many systems produce data as serial streams of individual bits. Bit serial operation is generally slower than parallel operation. Its main advantage is that, as only a single bit of each operand is acted on at any given time the hardware required is considerably reduced.

A basic bit serial adder/subtractor is the shown in ***figure 2.2***. It has been shown ***(section 2.2.4)*** that 2s complement notation allows subtraction to be performed as an addition operation. Therefore only a single full adder is required. For subtraction one input is inverted and the initial input to the carry chain is set to '1'. This is equivalent to adding the 2s complement of the inverted input. Both operands must be of the same length. This is achieved by padding the most significant bits of the shorter number with copies of the sign bit. Zeros are used to pad unsigned numbers. The operands are presented to the adder bit serially from l.s.b to m.s.b. The carry formed by each pair of operands is fed back to form the carry input signal for the calculation

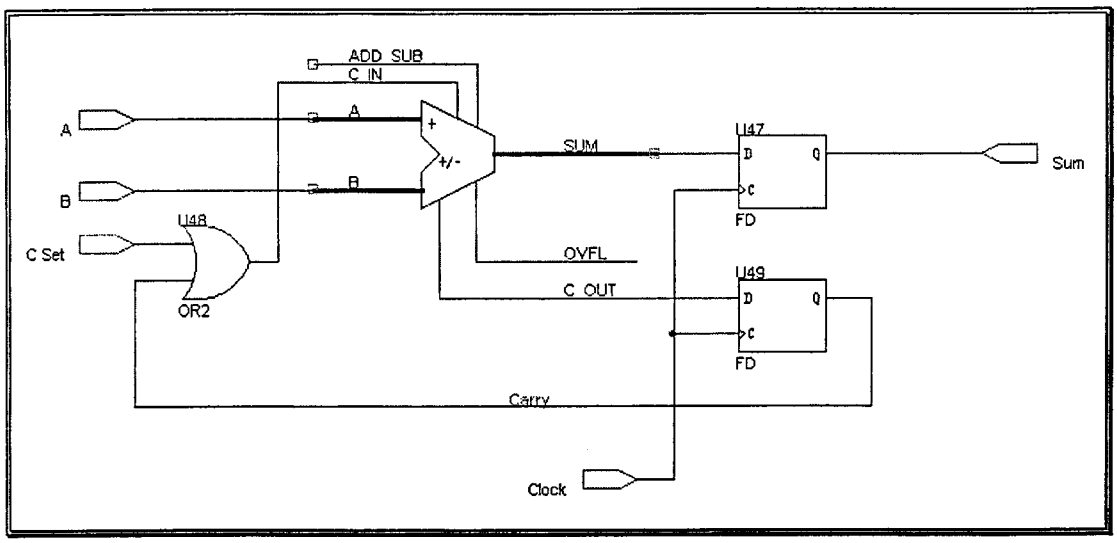


Figure 2.2 - A Bit Serial Adder/Subtractor

of the next bit. The addition or subtraction of two W bit long words requires W clock cycles. The final carry output forms the m.s.b of the sum except in 2s complement arithmetic where the final carry bit is discarded to maintain a fixed word length.

There are two forms of bit serial multiplication/division. Firstly there is genuine bit serial operation where both operands are presented bit serially. The second and simplest form is serial-parallel, where all bits in one or both of the operands are known prior to operation. Serial-parallel multipliers may make use of either serial or parallel adders. The use of bit serial adders leads to very compact multipliers, however due to the very large number of additions that need to be performed these circuits are slow.

A class of fast serial-parallel (FSP) multipliers has evolved which reduce computation times by using only full adders and basic combinational logic[3]. An FSP multiplier based upon the carry-save add-shift (CSAS) technique is shown in ***figure 2.3***[4].

The multiplier is presented in parallel to the system while each bit of the multiplicand is in turn broadcast throughout the system. For an M bit long multiplicand the first M clock cycles are used to calculate the partial products from the multiplicand. These are output from P0 in ***figure 2.3***. A further M clock cycles are required to produce the remaining bits of the product from the data held in the adder chain. It has been shown that the computation time for a CSAS based FSP multiplier can be significantly reduced if the data held in the adder chain after M clock cycles is processed using a single parallel addition. This technique has been further extended to operate on 2s complement data via an implementation of the Baugh-Wooley algorithm[5]. The

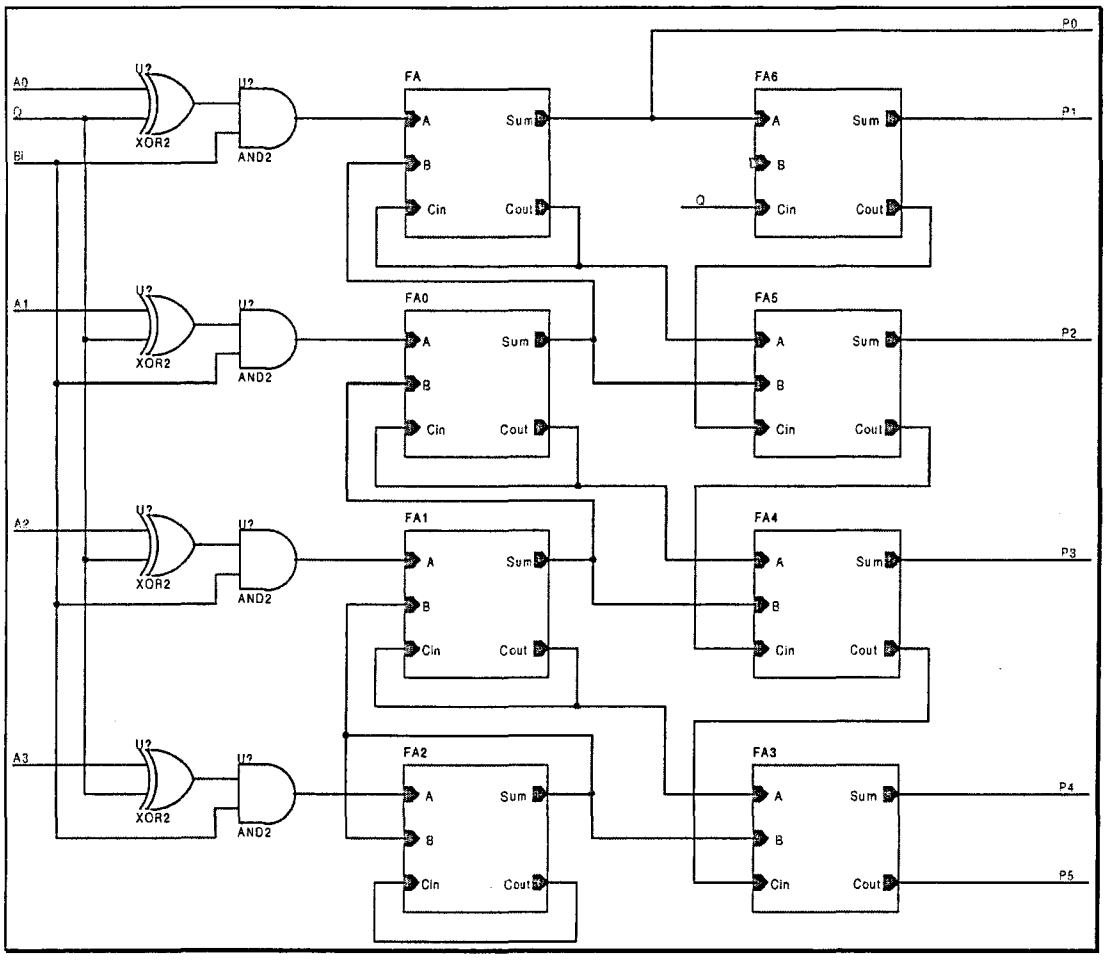


Figure 2.3 - A Fast Serial-Parallel Multiplier

hardware cost of the FSP multiplier is greater than that of a similar circuit employing bit serial addition as both the adder chain and an M bit parallel adder are required. However the reduction in computation time achieved by replacing M-1 serial additions with a single clock cycle, parallel adder operation can considerable, particularly when long multiplicands are used.

More complex techniques are required for multiplication or division where both operands are presented bit serially. A single bit slice of a serial multiplier, after Ienne and Viredaz[6], is shown in ***figure 2.4***. M-1 such slices are required to multiply two M bit numbers with some additional logic required for the most and least significant bit slices. With minor adjustment the system can be made fully modular, i.e. all bit slices are identical, thus facilitating simple extension to any word length. 2M clock cycles are required to produce an output. The sign bits of the inputs must be preserved at the inputs during the latter M clock cycles or the data sign extended to a length of 2M bits. This design is particularly compact and has a low latency. It is however only one of many serial multiplier designs.

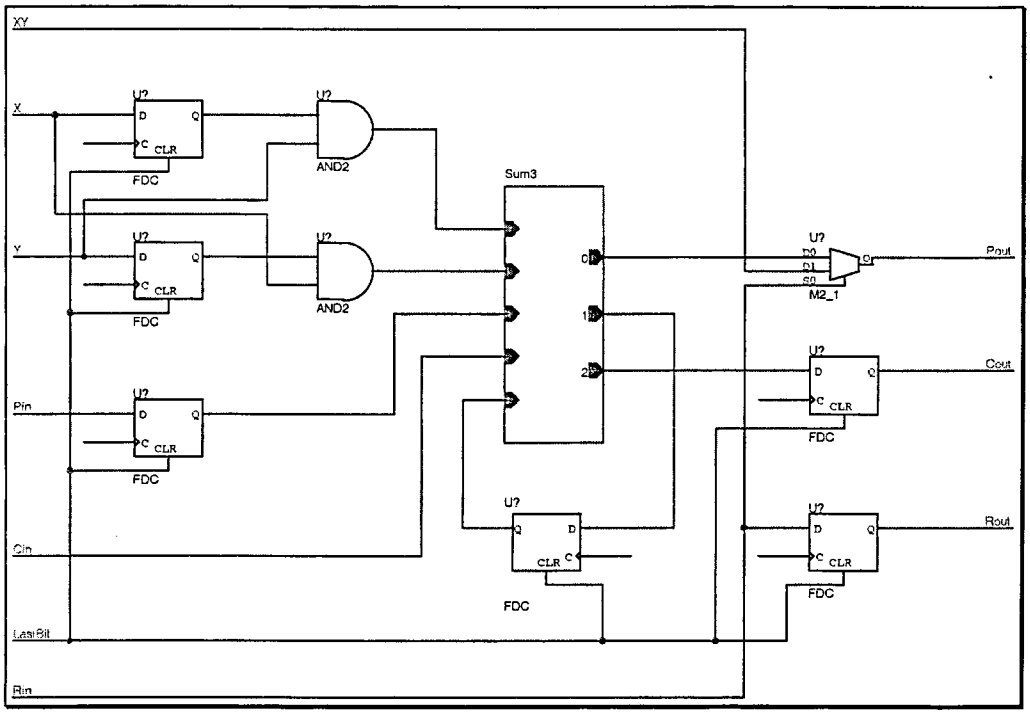


Figure 2.4 - A Bit Serial Multiplier Slice

2.3 Errors in Digital Arithmetic

In manual arithmetic numbers may be written down to any given accuracy as long as the value is known to that accuracy. Digital systems usually operate to a fixed word length due mainly to the need to compartmentalise memory into bytes and to match data widths to bus widths. The use of a fixed word length can lead to the occurrence of errors in binary arithmetic. It is impossible to express any number absolutely without specifying an infinite number of significant figures. All calculations therefore inherently contain inaccuracies. There exists a threshold above which the inaccuracy becomes too large and must be classed as a mathematical error. This threshold is different for each application and is determined by the requirements of the solution. The term error is used in the remainder of this chapter. The primary sources of error[7] are quantisation, truncation and overflow. A less obvious source of error arises from the process of normalisation in floating point.

2.3.1 Quantisation Errors

Quantisation is most commonly used to refer to the process of sampling an analogue signal. A quantised system is any system that can only take one of a fixed set of values. Quantisation errors result from the fact that any number represented by a

particular system may only be expressed to an accuracy equal to the smallest number which may be expressed by that number system[8]. Thus for an integer the quantisation step is 1, any number can only be expressed to an accuracy of ±0.5 in integer notation (unless the number is a perfect integer). Quantisation errors occur whenever a fixed word length is used to represent a number, regardless of the base of the number system used. Quantisation errors limit the accuracy with which a given real event can be modelled mathematically. If the word length is sufficiently large, over 12 bits is commonly used as a rule of thumb[11], the errors tend to be randomly distributed and uncorrelated. Correlation is dependent upon the input signal. A widely varying input signal, e.g. white noise, will exhibit much lower correlation than an impulse or other low frequency input[9]. The effect of uncorrelated quantisation noise may be modelled by placing an additive white Gaussian noise (AWGN) source in series with the original signal source. The quantised signal is given by

$$s_n = s_c + e_n$$

Where s_n is the quantised value,
s_c is the original signal
e_n is an instantaneous AWGN sample

It can be shown[11] that the noise source has a variance $\frac{1}{12}L^2$ and a mean value of 0.5L, where L is the value of the least significant bit of the quantised word. If a rounding algorithm is used to create the sample the mean value of the noise is zero. The variance is unchanged through the use of rounding. If the word length is too short then the errors become correlated with the data and can no longer be represented by simple statistical means. An accurate description of the errors under these conditions requires prior knowledge of the distribution of the input data and the calculations being performed.

If integer data is used the least significant bit has a fixed value, independent of the word length. Both the mean and the variance are predefined, having the values –0.5 and 0.2083 respectively. In fixed point notation the mean and variance are dependent upon the word length. For a word of length W the mean has a value

$$2^{-1} \times 2^{-W} = 2^{-(W+1)}$$

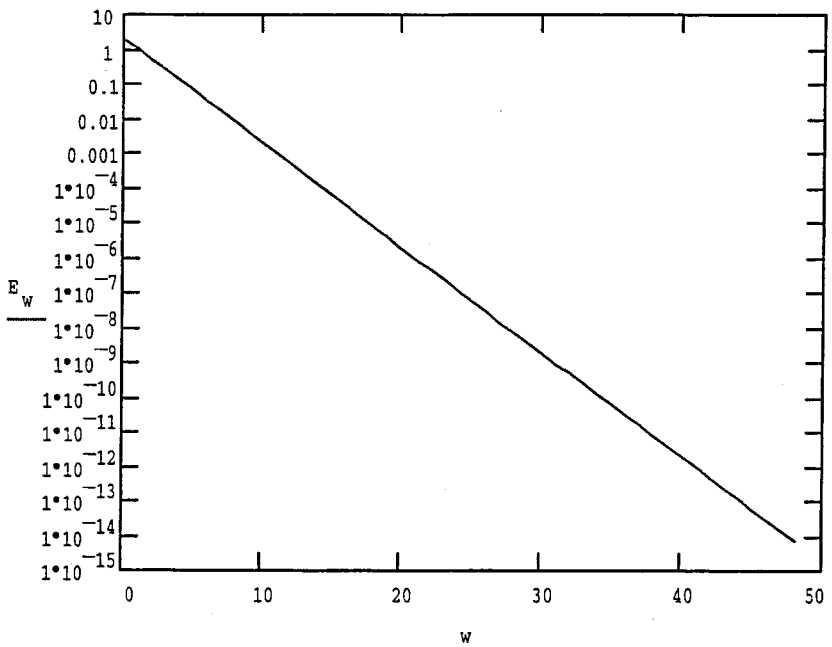
While the variance has a value of

$$\frac{1}{12}\times 2^{-2W}$$

The maximum number expressible in fixed point notation is 1.0, therefore if we express the mean error as a fraction of the maximum expressible value it remains unchanged at $2^{-(W+1)}$. Consider the mean error in integer notation expressed as a fraction of the maximum expressible value. For a word of length W this is given by

$$E_w = \frac{2^{-1}}{2^W} = 2^{-(W+1)}$$

It is clear that the magnitude of the mean error for a fixed word length given as a fraction of the largest number expressible in that word length is the same in both fixed point and integer representation. This function is plotted in *figure 2.5.*



Quantisation gives rise to a signal to mean quantisation noise ratio (SNR) of

$$SNR \cong 20\log N \quad dB$$

where N is the number of bits used to represent the signal. Thus the SNR increases by 6 dB for each extra bit used.

2.3.2 Truncation Errors

Consider the summation of 10011000.0110 with 110.011000111. Before the summation can take place the fractional points of the two words must be aligned. To the left of the fractional point the smaller operand is padded out with zeros. However to the right of the fractional point the smaller operand must be truncated to .0110 in order to maintain a fixed word length. This represents a loss of half of the significant bits of the value. Truncation may be considered as a process of re-quantising a number to fit within a given representation. As such the statistical treatment of truncation errors is as for quantisation errors (***section 2.3.1***). Errors of this type mainly occur in mantissas of floating point numbers where the fractional points can not be shifted without a loss of data. Truncation is not limited to addition but is also prevalent in multiplication. Consider the multiplication of two m bit numbers. The result is a 2m bit long number. For integers, truncation of this number by removal of the lower m bits results in a meaningless value. Systems that require multiplication with a fixed length result use fixed point representation (***section 2.2.2***). In this case the result after truncation is meaningful but is subject to quantisation noise. Each calculation within the system that requires a number to be truncated represents a source of AWGN. As the noise is uncorrelated each source may be added in series to produce a total noise figure for the system. The effect of truncation may be reduced through rounding, which gives a mean noise value of zero.

2.3.3 Overflow Errors

An overflow occurs when the result of an arithmetic operation is too large to fit in the target memory location[10]. Consider the fixed length addition of two unsigned numbers 1001 and 1110. The result is 10111, which, to a fixed word length of 4 bits yields the incorrect result 0111. Addition operations with two operands may produce one overflow bit (see ***table 2.1(a)***). However the multiplication of two W bit long integers may result in a $2W$ bit long product, a potential overflow of W bits. Overflow errors are particularly damaging as they affect the most significant bits of the data.

Overflow in fixed length integer data may be overcome by right shifting the data and moving the fractional point. This technique is used in the block floating point scheme introduced earlier. The same technique is used to correct overflow in the mantissa of

a floating point number. Data normalised in this way is subject to quantisation noise due to the truncation of the least significant bits of the word.

Fixed point notation is commonly used where a large number of multiplications are to be performed. Recall that fixed point numbers are wholly fractional. The product of two numbers less than one is itself always less than 1. The product can therefore never extend to the left of the fractional point, the extra bits are formed at the least significant end of the word. The product may then be truncated to a word length of W to prevent overflow. Although the truncation adds noise (***section 2.3.2***) it is less damaging than either overflow or truncation in integer data, both of which result in a meaningless value.

2.3.4 Normalisation Errors

An often overlooked error is encountered through the process of normalisation. Floating point numbers are often stored such that the mantissa is wholly fractional with no leading zeros or copies of the sign digit. This format maximises the number of significant bits that may be stored. Problems may occur when the word length allows numbers to be stored to a greater precision than they are initially specified to. Consider the value $0001010*2^6$. This may be normalised to yield $10100000*2^2$. This implies an accuracy of 8 significant bits. However only the first four bits of data are significant. Normalisation errors must be carefully controlled, as any operation where one operand contains non-significant bits will propagate the error. Hence during repeated operations the proliferation of non-significant bits due to normalisation errors can corrupt large amounts of data. Normalisation errors are particularly difficult to detect unless the expected result of a calculation is known. Errors of this type are most common when calculations require data to be truncated and later re-normalised. In this case the truncated bits are not restored but are replaced with zeros. The exponent acts as a gain function, amplifying the noise created by the truncation operation[10]. Normalisation errors can not easily be quantified statistically as they are highly correlated with the data. However a careful examination of the calculations to be performed will highlight areas in which these errors are likely to occur.

2.3.5 Reliability

The reliability of a system is defined as 'the probability that a system will not fail within a time t given that it was working correctly at time 0'[10]. The failure of a system is 'any deviation of [the system] from its specified correct behaviour'[10]. A

failure is caused by the existence of an error or incorrect output from a module or subsystem. Again the term error must be taken to imply an inaccuracy above a certain threshold. Assuming that the hardware of the system is sound, the main source of error for an arithmetic unit is the input data. If the system is fed operands for which it can not resolve the current operation it will fail. Some failures are catastrophic, such as an attempt to divide a number by zero. The result of this operation is undefined[ϖ] and hence the system will fail. There are three stages[10] in fixing an error - detection, diagnosis and repair.

- *Detection* - The first stage in repairing an error is determining that the error exists. Some errors, e.g. normalisation error, are inherent in a given operation, therefore the proliferation of these errors may be monitored each time the operation takes place. Other errors such as overflow are dependent upon the input data, therefore the simplest form of detection is to apply a set of rules for checking the input data.
- *Diagnosis* - Once an error has been detected the nature of the error must be determined. The definition of the rules used for error detection can help in diagnosis.
- *Repair* - Errors must be prevented from propagating through the system. Repairable errors include overflow and truncation errors in floating point. Both of these errors may be fixed by shifting the mantissa and incrementing or decrementing the exponent accordingly. Truncation errors may be irreparable if shifting the mantissa to recover the lost bits would lead to overflow. In this case the error may be minimised by using a rounding algorithm. Truncation and overflow in integer data are at worst irreparable. In this case detection should occur at the inputs to the arithmetic unit and the operation should be prevented from taking place.

2.4 Performance Issues

The performance of a system is usually quoted in terms of the number of operations performed per second (ops). It has been shown[1] that multiplication and division may be performed using multiple add/subtract and shift operations. In 2s complement add and subtract are the same operation, therefore a 2s complement arithmetic unit could be constructed purely from adders and basic logic. However the basic add and shift techniques may require many cycles to perform a single operation and therefore limit

[ϖ] The result can not be taken as infinity as the divisor is only known to be zero to within the specified number of significant bits.

performance. In order to increase performance the time taken for each operation must be reduced.

The use of more sophisticated multiplication and division algorithms can significantly improve performance. However as a general rule the hardware required to perform these algorithms increases in complexity as the processing time decreases. This rule is not limited to multiplication/division. A good example of this is the extra logic required in producing a look ahead carry parallel adder. The introduction of error checking and repair logic may also degrade the performance of a system, however this must be weighed against the time lost in repeating calculations should an unchecked error occur. High performance is clearly often obtained at the cost of system complexity and size.

2.5 Conclusions

The principles of binary arithmetic have been presented. The four basic operations - addition, subtraction, multiplication and division, have been discussed for both parallel and serial operation. It is clear that numerous methods exist for performing each of these algorithms. The choice of method for a given application is a trade off between performance, resource availability and reliability.

The design of digital arithmetic systems is clearly a complex issue. The required reliability and performance must be determined beforehand. A system that is used to perform a few operations before its state is checked will need less error checking and repair software/hardware than a deep space probe which has to work unchecked for years. Similarly a PC graphics card must be capable of performing calculations many times faster than a simple calculator. In each case the requirements must be clearly stated and the choice of hardware made accordingly.

While it would appear from above that floating point notation gives rise to fewer irreparable errors than fixed point notation it is considerably more difficult to implement, requiring logic to process both the mantissa and the exponent. It has been shown that despite the increased range available to floating point numbers they are subject to quantisation noise in the mantissa in the same way as fixed point numbers. Block floating point representation combines the range of floating point with simplicity of fixed point. The accuracy of a block floating point representation lies between that of fixed point and that of floating point. The simplicity of fixed point

calculations means they often give higher performance than floating point operations, however they offer lower precision and reliability. The existence of a number of commercially successful fixed point DSP chips (e.g. ADSP21XX, Motorola 5630X) is evidence of the fact that, with careful algorithm design, fixed point notation can provide sufficient accuracy even for demanding signal processing applications.

References

[1] Pirsch, P 'Architectures for Digital Signal Processing', John Wiley & Sons, New York, 1998

[2] Oklobodzija, V.G; Villeger, D and Liu, S.S 'A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers using an Algorithmic Approach', IEEE Trans. On Computers, Vol.45(3), pp.294-305, 1996

[3] Dadda, L 'Some Schemes for Parallel Multipliers', Alta Frequenza, Vol.34, pp.349-356, 1965

[4] Sunder, S; El-Guibaly, F and Antoniou, A 'Two's Complement Fast Serial Parallel Multiplier', IEE Proc. Circuits Devices Syst., Vol.142(1), pp.41-44, 1995

[5] Baugh, C.R and Wooley,B.A 'A Two's Complement Parallel Array Multiplication Algorithm', IEEE Trans. Comput., Vol.33, pp.1045-47, 1983

[6] Ienne, P and Viredaz, M.A 'Bit Serial Multipliers and Squarers', IEEE Trans. Comput., Vol.43(12), pp.1445-1450, 1994

[7] Nelson, V.P and Carrol, B.D (eds) 'Tutorial : Fault Tolerant Computing', IEEE Comp. Soc. Press. Los Angeles, USA 1987

[8] Porat. B 'A Course in Digital Signal Processing', J. Wiley and Sons, New York, 1997

[9] Oppenheim, A.V and Schafer, R.W 'Digital Signal Processing', Prentice Hall, 1975

[10] Higgins, R.J. 'Digital Signal Processing in VLSI', Prentice Hall, 1990

3. A Data Parallel Application Specific Processor for TLM

3.1 Introduction

Chapter 1 highlighted the fact that run times for serial computations of TLM are becoming excessively large. The development of dedicated processors for the TLM method has, in the past, produced an increase in performance with respect to software applications. However recent advances in computer technology have increased significantly the performance of desktop PCs and workstations and therefore the rates of computation that may be achieved by these systems. The feasibility of an application specific processor for TLM is dependent upon advances in the technology used to implement the processor. The application specific processors of Saleh[1] and Gregory[2], respectively built using discrete components and small scale integration, are made obsolete by the very large scale integration (VLSI[3]) techniques used to create modern 300+ MHz processors. This chapter documents the development of an application specific processor for the TLM method developed using current technologies. The aim of this work is twofold. Firstly the performance of the design is compared to software running on a personal computer to determine the feasibility of the application specific approach to reducing run times in the modern computing environment. The second aim is to evaluate suitable implementation strategies for a new TLM processor.

3.2 Design Methodology

The development of an application specific processor may be divided in to two key areas[4].

- Algorithm development - The specific form of the TLM algorithm implemented can have significant effects on the performance of the processor. Decisions have to be made as to which features the processor will implement e.g. boundaries, stub loading and two or three dimensional meshes.

- Hardware development - Choice of implementation technology - hardware mapped or RISC and specific architectural issues e.g. single node or array, floating point or integer data.

3.2.1 Algorithm Development

The increased efficiency of a hardware mapped approach arises mainly from a reduction in the number of operations which must be performed in the application of a given algorithm. The mapping of an algorithm directly to the hardware of a device reduces the need for instruction fetching and decoding cycles, instruction stack operations and data management (caching, storing of partial results etc.) operations. The form of the algorithm to which the hardware is mapped has a bearing on the efficiency of the design. The algorithm must therefore be developed with some knowledge of the way mathematical constructs are performed in hardware[5]. The design of digital arithmetic systems was discussed in ***chapter 2***. The hardware required to implement adders, multipliers and other typical mathematical functions varies a great deal. As such it may be necessary to trade off one form of an algorithm for another requiring more calculations but where each calculation is in itself less complex. The choice of data representation and word length is also significant. The hardware requirements of a floating point system are greater than those of a fixed point system. The requirements of a bit serial fixed point system are less than those of a data parallel system. A careful choice of algorithm can reduce both the number and complexity of operations performed by the hardware. Many calculations produce partial results that must be stored until required. A single flip flop is sufficient to store one bit of data. Registers may be formed by linking a number of flip flops with a common enable line. However if many results must be stored the consumption of resources and the complexity of the logic required to access the correct register can increase rapidly. An efficient design must keep the storage of partial results to a minimum, utilising results as they are produced.

The TLM method takes many forms in both two and three dimensions, each of which has a unique algorithm. The two dimensional shunt node algorithm makes a good starting point in the development of a TLM processor.

- The shunt node was used as the basis for the processors of both Saleh and Gregory. Developing the new processor around the shunt node offers some continuity. This is important when comparing performances to test the feasibility of the current design.

- The algorithm is relatively simple. However the scatter and connect processes used form the basis for all other TLM schemes. The ideas developed with the shunt node could therefore be expected to form a solid base for the implementation of more complex schemes.

3.2.2 Hardware Considerations

The implementation of a new circuit in silicon is a costly process. The design must be verified before it is committed to hardware, requiring many hours of skilled design and rigorous testing. The design process is usually performed on paper or on a computer, rarely involving actual hardware. Once a circuit has been designed and verified there are three typical routes to implementation.

1) Discrete components mounted on a PCB or wire wrap board. This approach, as used by Saleh, is relatively low cost for small designs. Wire wrap has the flexibility that components may be easily moved or changed. The main disadvantages are that discrete components of this nature tend to have high propagation delays and power consumption and are also physically large, thus limiting the size of circuit which may be created.
2) Custom IC design, as used by Gregory. A custom built IC offers a high speed, single component solution. However the design process is complex, requiring a detailed knowledge of gate level design and fabrication techniques, and mistakes are impossible to rectify following implementation without rebuilding the entire circuit. The equipment required for fabrication is very expensive and production is usually limited to specialist companies. This approach is best suited to high volume production following a significant prototyping phase.
3) Field Programmable Gate arrays (FPGAs)[6]. An FPGA is a silicon chip composed of a large array of logic cells, which may range from single gates to multiple input look up tables (LUTs). The function and interconnection of the cells is usually defined by loading an appropriate bitstream in to the device, allowing any combination of logic functions to be mapped in to the array. Bitstreams, binary files containing programming data for the device, are generated from either schematics or textual ('netlist') descriptions of the circuit to be implemented. FPGAs are low cost and in most cases can be reprogrammed any number of times, the new configuration overwriting the old one. Libraries of macros, logic configurations designed to perform common functions, are available for many FPGAs. Complex circuits are

created by linking these macros together. This approach therefore combines the flexibility and simplicity of discrete components with the speed and size advantages of a custom IC in a low cost, low development time solution. For bulk applications FPGAs prove considerably more expensive than custom built ICs, therefore their main advantage is in prototyping. Design faults that would require a costly remanufacture in an IC require only a reprogramming of the FPGA.

It would appear that FPGAs offer an ideal design solution for developing a TLM processor. The potential exists for transferring the design to a custom IC following testing.

FPGA architectures vary between manufacturers in two ways, the type and granularity of the logic cells and the programming method. In much the same way as an efficient processor must map the granularity of its architecture to that of the problem, so the FPGA must map the granularity of its logic cells to that of the logic to be implemented. FPGAs may have a coarse or a fine grain architecture. Fine grain architectures typically use a 'sea of gates' construction in which the device consists of a large array of logic gates. These are usually NAND or NOR. The programmer has control over the interconnections between the gates. Coarse grain architectures are built around an array of more complex cells. These may consist of combinational logic. However modern FPGAs have been developed around SRAM based look up tables (LUTs). The principle behind the LUT is that a single output combinational logic circuit may have its N inputs in any one of 2^N states. Each state will produce either a '0' or a '1' output. The combinational logic is replaced with a 2^N bit deep SRAM that uses the logic inputs as its N address lines. Each state of the inputs therefore addresses a different location in the memory. The memory is then programmed with the required pattern of 1s and 0s to produce the output defined by the logic mapped within the LUT. There are three main advantages to this approach.

- Large logic circuits, with many levels of logic and therefore large propagation delays, may be reduced to a single SRAM LUT with a single propagation delay.
- The logic that is mapped to the SRAM may contain any combination of gate types, where as sea of gates arrays usually consist of only one type of gate.
- The logic functions performed by the FPGA may be changed by simply writing new data to the SRAM cells, thus providing unlimited reprogrammability.

The shunt node algorithm requires only additions, subtractions (which may be implemented using 2s complement addition) and a divide by two, which may be achieved by right shifting the data one bit. The main components of the processor are therefore adders. While these may be developed from a gate level architecture a coarser granularity may reduce propagation delays and therefore provide better performance.

The programming method is more a matter of convenience. Some technologies allow programming with the FPGA *in situ* whereas others require special equipment. Some require less loading time or data storage. The choice of programming method only becomes an issue when specific factors in the circuit layout require the use of one method. For example an FPGA in an embedded system must program itself from a PROM on its circuit board at power up. Prototyping systems require programming from a host machine. The nature of the host will define whether the bitstream is presented in a serial or a parallel format.

3.3 Xilinx XC4000 FPGAs

The Xilinx XC4000[7] family of FPGAs is built around an array of configurable logic blocks (CLBs). Each CLB consists of two four input and one two input LUTs, denoted F, G and H respectively. These may be combined to produce a LUT with up to 9 inputs. Each of the four input LUTs has an associated flip flop for intermediate data storage. A single full adder/subtractor may be compressed in to two four input LUTs, each with a propagation delay of around 5ns. It is therefore possible to build very high speed adders and subtractors with the XC4000 CLBs. The Xilinx devices also use dedicated carry bit routing for efficient design. The CLB architecture and the availability of larger components with high pin counts make the Xilinx XC4000 FPGAs ideal for developing the TLM processor.

The design flow for these devices is shown in ***figure 3.1***. A sequence of software packages synthesise a generic, gate level netlist in the EDIF netlist format from a high level text description or a schematic. Proprietary, vendor specific software from Xilinx, called M1, translates this netlist in to a physical layout via the MAP and place and route (PAR) routines. The combinational logic in the netlist is partitioned by the software, where possible, in to groups of 4, 5, 8 or 9 input, single output functions, each of which is

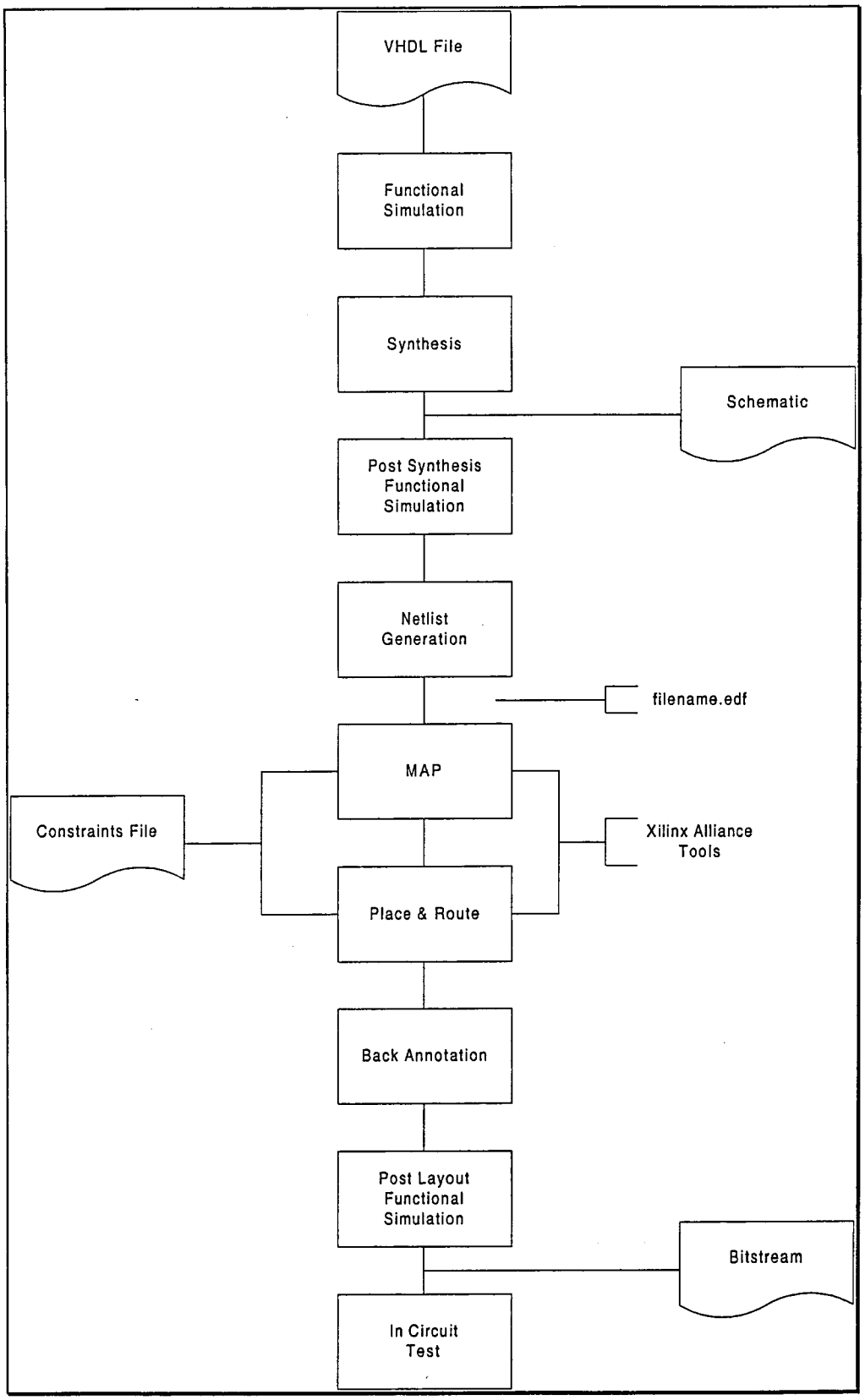


Figure 3.1 - Design Flow for a Xilinx FPGA

mapped to one or more LUTs within a CLB. These CLBs are then placed within the array so as to minimise routing delays and signal skew. A graphical representation of the device is provided allowing the layout to be edited manually at the CLB level. This is useful for adjusting critical logic or routing where guaranteed performance is required. The device is programmed by downloading a bitstream generated by the M1 software. The bitstream defines the contents of the LUTs and the individual SRAM bits, which control the programmable interconnect routing matrix. The device may be programmed *in situ* over a data bus or at power up from an EPROM mounted on the circuit board. Loading the bitstream takes from a few hundred milliseconds to a few seconds for a single device depending on its size.

Back annotation tools can be used to re-write the EDIF netlist to include information on the propagation delays within the device. This post layout netlist can be incorporated in to a schematic, which may include external components. This allows the interaction of the FPGA with, for example, a bus interface to be tested. With the aid of simulation software the performance of the final implementation can be predicted with a very high degree of accuracy. A medium sized design can be taken from a schematic, through verification to an operational FPGA device within a few hours. Any design that fails to meet the specified criteria can be overwritten with a new bitstream, allowing development costs to be minimised.

3.4 Numerical Representation in the TLM Processor

In ***chapter 2*** it was shown that the way in which numbers are represented in a system could have a significant impact upon the behaviour of the system. A balance must be struck between accuracy, range and processor complexity. The three main numeric representations used in digital systems are

1) Fixed Point – Also referred to as integer data. The accuracy of fixed point arithmetic is strongly dependent upon the word length used. Calculations may be subject to quantisation noise, truncation errors and overflow. The hardware required to implement fixed point arithmetic is simple and exhibits a high throughput.
2) Floating Point – Represents each number using a mantissa and an exponent. Floating point representation is able to express a much larger range of values than a fixed point

number of the same length. Accuracy is less dependent upon word length as the exponent allows the data to be scaled, preserving significant bits which would have been lost in fixed point arithmetic. Floating point hardware is complex as operations must be performed on both the mantissa and the exponent. High throughput floating point arithmetic units may be realised at a high silicon cost.

3) Block Floating Point – This method is used for calculations on arrays of fixed point data. A single exponent is applied to the whole block. The data can be scaled as in floating point. Block floating point is less accurate than floating point as numbers can not be scaled individually. It is more resilient to truncation errors and overflow than fixed point and exhibits a greater range. The arithmetic is carried out on fixed point numbers, therefore the hardware requirements are low. They are greater than for fixed point due to the need for shifters but lower than floating point.

TLM requires the repeated application of a calculation to an array of data. Any inaccuracy introduced through the calculation will be exacerbated by its repeated application. As a consequence of this the accuracy of the numeric representation within the processor must be high. If the processor is to demonstrate an increased throughput with respect to a PC the arithmetic logic must be fast. High speed implies a fixed point representation. The accuracy of fixed point arithmetic is dependent upon the word length used. A longer word length provides a lower mean quantisation noise but produces a larger, slower circuit.

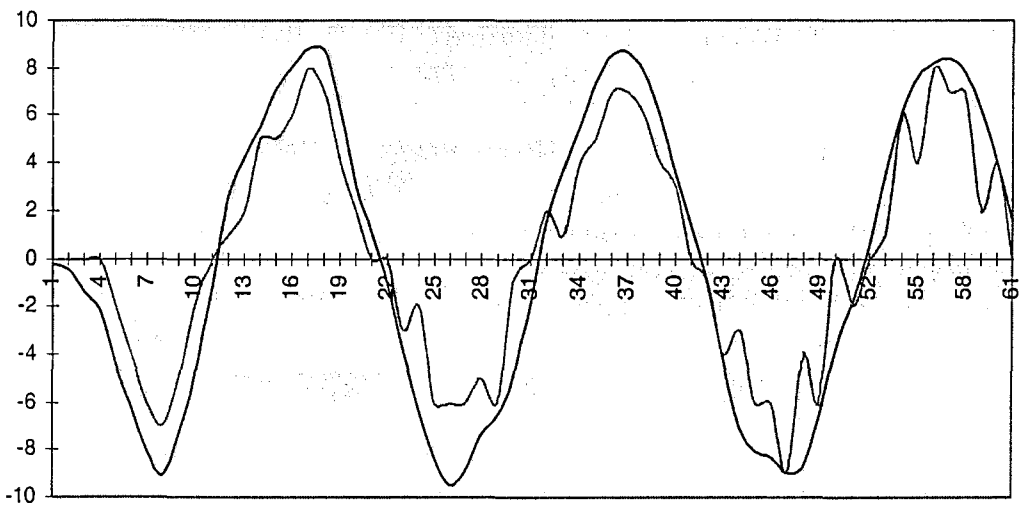


Figure 3.2a - Comparison of Integer and Floating Point Results, Short Integer Word Length

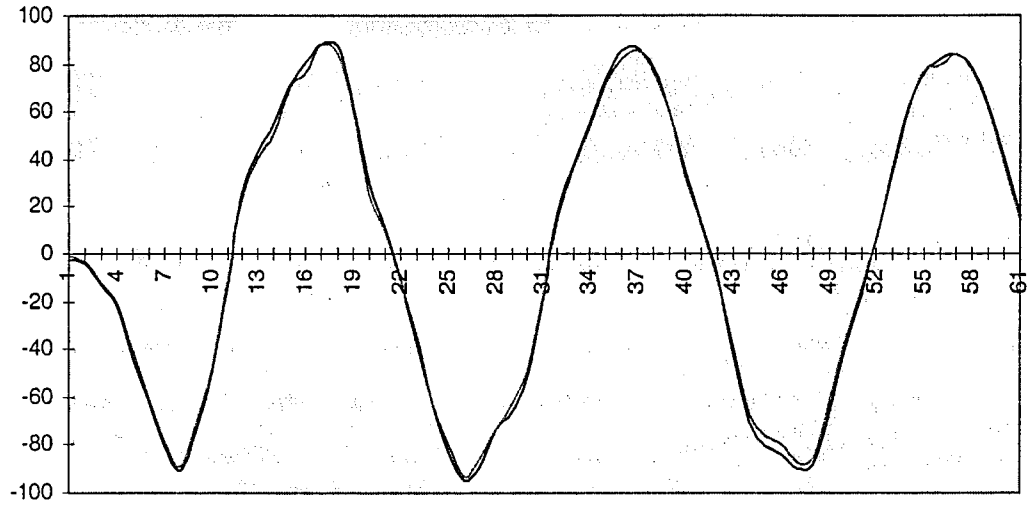


Figure 3-2b - Comparison of Integer and Floating Point Results, Medium Integer Word Length

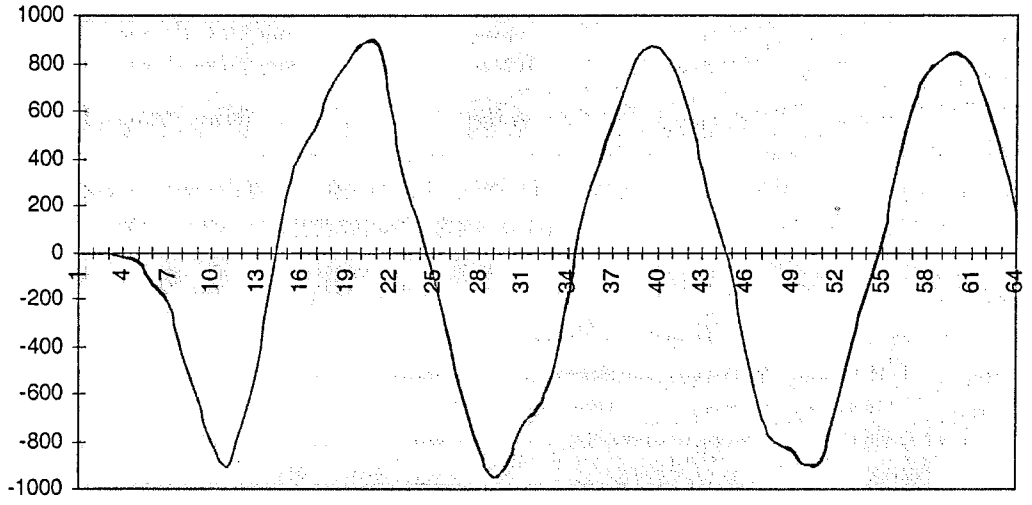


Figure 3-2c - Comparison of Integer and Floating Point Results, Long Integer Word Length

Figures 3.2(a-c) show the effect of varying the size of the word length in a TLM simulation written in C++. These results were obtained by exciting a 100 x 100 node mesh at the center node with a continuous sine wave. The blue line shows the result of the simulation performed using 32 bit floating point arithmetic. The magenta line shows the results of the same simulation using integer arithmetic. ***Figure 3.2a*** clearly shows the effect of quantisation noise on the integer data. Significant distortion of the output has

occurred. Although quantisation noise is present in the floating point data it has a much lower mean value. As the word length is increased, so the difference between the floating point and integer results decreases. The integer quantisation noise has a fixed mean level (see ***chapter 2***). However the signal to noise ratio (SNR) is greatly increased. The SNR may be maximised by normalising the integer input such that it occupies all the available bits of the input word. As demonstrated in ***figure 3.2c***, there is a negligible difference between the fixed and floating point results for a 32 bit word length.

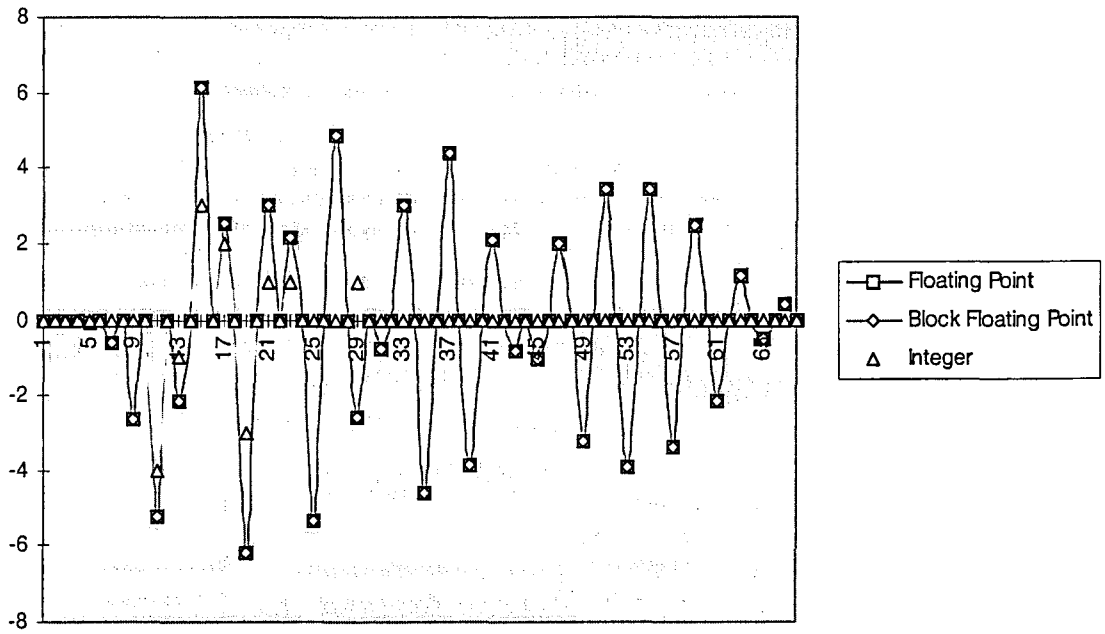


Figure3.3 - Comparison of Simulation Results Using Various Numeric Formats

Figure 3.3 shows the application of a block floating point scheme to TLM. The output was generated from a low amplitude impulse excitation to the center of a 100 x 100 node mesh. It is clear from the diagram, which shows a section of the first 200 iterations, that by the time the output node is reached the integer data has decayed to a level where it is consumed by noise and no longer accurately represents the desired output of the simulation. However the block floating point and floating point outputs are indistinguishable. The output of the block floating point scheme has been re-normalised to yield the same exponent for each output point. The exponent was shifted four times during a 10000 iteration simulation. This yields a very small computational overhead with respect to an integer calculation but provides a significant improvement in the accuracy of the results.

Each TLM model will have a different set of requirements. In some cases the increased accuracy and range of floating point may be deemed a necessity. These however are exceptions. It has been shown that given a sufficiently large word length, fixed point arithmetic can exhibit an accuracy very close to that of floating point. The use of a block floating point scheme extends this accuracy over a range of numbers comparable to floating point for a small increase in hardware costs. In the majority of cases the accuracy provided by a block floating point scheme is greater than that offered by the model itself. Limitations on boundary placement and geometry and similar effects of spatial quantisation will sometimes generate a greater inaccuracy in the output of a model than the effects of numerical quantisation.

The effects of quantisation noise may be further limited when necessary at the output of the TLM mesh. Excitations in the TLM mesh are band limited to minimise dispersion. Typically the maximum frequency of the excitation is one tenth of the sampling rate (thus giving 10 nodes per wavelength). However the quantisation noise is white noise; it contributes an equal power at all frequencies. The noise bandwidth extends from 0 Hz up to the Nyquist frequency. Assuming 10 nodes per wavelength the noise bandwidth is therefore 5 times greater than the signal bandwidth. ***Figure 3.4*** shows the effect of applying a very simple 5 point, finite impulse response, low pass filter to the integer data

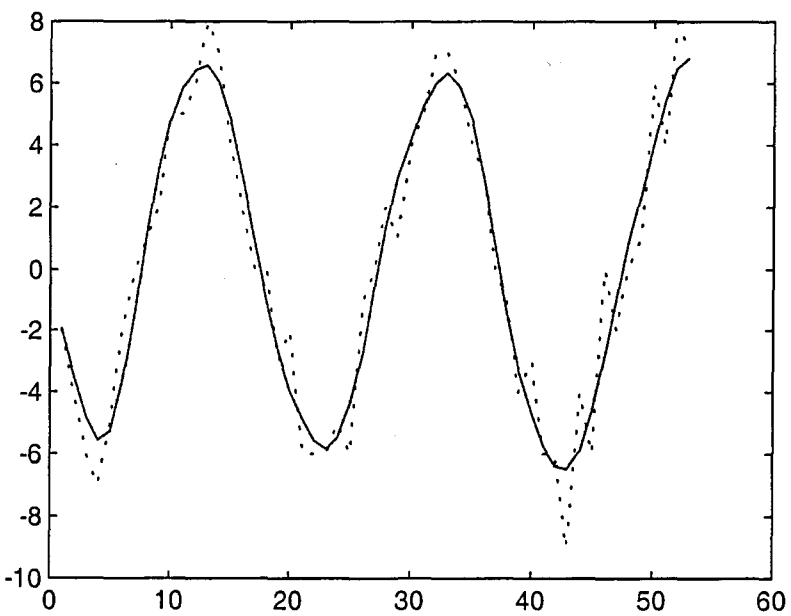


Figure 3.4 - Effect of Filtering the Output of a Noisy TLM Mesh

output of ***figure 3.2a***. The effect is to limit the noise bandwidth, greatly improving the SNR. The output of the filter, the solid line, shows the recovered sine wave. Filtering performed on the output of the TLM mesh may be used to reduce the noise power and minimise the effect of quantisation noise.

The significant reduction in hardware costs offered by integer arithmetic over floating point arithmetic makes it an attractive choice. It is also reasonable to expect an integer arithmetic unit to demonstrate a higher throughput. The use of a block floating point scheme further extends the range of integer data without significantly affecting either of the above properties. If integer data is to be used in the TLM processor a suitable word length must be chosen. The decision must be based on a trade off between hardware requirements, speed and accuracy. The quantisation step, the smallest change in value that may be represented by a given numeric representation, may also be an issue. The TLM processor must be able to accurately model the smallest changes that may occur in the system under consideration.

32 bit integer data was chosen for the TLM processor for a number of reasons.

- 32 bit data offers a range of 2^{32} = 4294967296 numbers. The quantisation step as a fraction of the full scale value is 2^{-32} = 2.33e-10.
- The mean truncation error as a fraction of the full scale value is 1.16e-10. This offers a full scale signal to mean quantisation noise ratio (SNR) of 99dB.
- 32 bits represents a standard bus width for many systems. This is an important consideration in the integration of the processor in to a complete system.

The dynamic range and quantisation step were considered sufficient for most applications in TLM. The loss of range with respect to a floating point solution may be compensated for by an increased throughput due to the simplified circuit architecture. The range may be extended using a block floating point scheme. The job of monitoring the data and scaling the array would be best performed by a host system.

The use of integer data with a fixed dynamic range does not restrict the implementation of non-linear models. While the dynamic range of the 32 bit data may not appear to suit the non-linear system under consideration it must be remembered that the model is a user generated entity. As such its parameters may be adjusted to scale the input data to the non-linearity under examination. When using the term integer it is often assumed that the bit positions are fixed to represent $2^W \ldots 2^1, 2^0$. However through careful choice of the

model parameters the bit positions may assume an arbitrary range of consecutive powers of 2. Models requiring small value inputs do not necessarily suffer a decrease in range or accuracy.

There are certain classes of problem which are less affected by the use of fixed point data. Problems involving fully enclosed meshes or meshes with constant sources are less affected as the total energy within the mesh is maintained. As such the range of values of interest tends to be smaller. Integer arithmetic is less suited to those problems where small amplitude effects are important. This class of problems is not excluded from making use of the processor. It is however necessary to give a little more consideration to selecting a suitable word length. The use of a block floating point scheme extends the range of the processor. However this is most useful for tracing decaying waveforms such as impulses. The inability to scale individual data words makes the scheme ineffective for problems where a large range of must be represented simultaneously. In this case the increased range must be provided through an increased word length. Such problems are relatively uncommon.

3.5 Design of the TLM Processor

The introduction of computer aided engineering (CAE) has allowed designers to lay out circuits on a computer and study their operation via simulation without the need to build any physical hardware. Performance and operation can therefore be fully tested before the design is transferred to hardware. A useful extension to CAE is the hardware description language (HDL)[8]. The HDL allows a circuit to be described either structurally or behaviourally using a high level language similar to many programming languages. The behaviour of the circuit thus described can then be verified through simulation. The key advantage of this approach is that the initial HDL description is technology independent, removing the need to know in the early stages of design which technology will be used for the final implementation of the circuit. The HDL may be used as a technique for the rapid prototyping of algorithms, following which a circuit is built using traditional techniques. A more powerful technique, that of logic synthesis[9] uses computer software to generate a circuit from an HDL description. The synthesis package may generate a schematic for entry in to a CAE package for testing and

simulation. It may also generate a text based netlist file, detailing all the components in the circuit, their properties and their interconnections.

There are two main design methods used with hardware description languages, behavioural modelling and structural modelling. Behavioural modelling may be considered as 'black box' modelling. The HDL is used to define the relationship between the inputs and outputs of the system with no consideration as to its architecture. It is useful if the designer has some idea of the architecture required from the behavioural model as completely abstract descriptions will lead to abstract and inefficient circuits. Structural modelling defines the system in terms of its component parts and their interconnections. Structural models are usually hierarchic. The bottom level must contain a behavioural description of each component. For example, an arithmetic unit may be modelled as a collection of adders, which are in turn modelled as individual NAND gates. The bottom level of the hierarchy will then be a behavioural description of a NAND gate.

3.5.1 VHDL Description of the 2D Shunt Node

There are several hardware description languages available. These include Verilog, Altera's AHDL and ABEL. One of the most popular languages is the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language or VHDL[10]. Using VHDL to model the circuit and simulate its operation, a number of different forms of the TLM algorithm may be tested. Changing the form of an algorithm in a text file is much faster than redesigning a circuit. VHDL therefore facilitates rapid prototyping of systems. As no predefined structure exists for the TLM processor, and its behaviour is defined by the shunt node equation, ***equation 1.4***, behavioural modelling seems the most appropriate approach. However, as stated above, the form of the equation affects the development of the circuit. The matrix form of ***equation 1.4*** is difficult to represent in an HDL. Expanding ***equation 1.5*** for each branch of the node would lead to a series of four equations which may lead to the synthesis of four adders all producing the same summation of the input values. A more efficient form of the shunt node equation for synthesis is

$$
\begin{aligned}
S &= \frac{1}{2}\left(V_1^i + V_2^i + V_3^i + V_4^i\right) \\
V_1^r &= S - V_1^i \\
V_2^r &= S - V_2^i \\
V_3^r &= S - V_3^i \\
V_4^r &= S - V_4^i
\end{aligned}
\tag{3.1}
$$

The form of ***equation 3.1*** ensures that the circuit synthesised from the VHDL will contain only one construct to form the sum, S. ***Figure 3.5*** gives a complete VHDL description of the circuit using ***equation 3.1***.

Logic synthesis is a very specific procedure. The HDL description must clearly specify all properties of the circuit to be synthesised. The circuit defined therefore by the set of equations presented in ***figure 3.5*** will not include any error checking or correction such as rounding. Such constructs, if desired, must be manually added after synthesis or included in the HDL file.

3.5.2 Logic Synthesis

The synovation synthesis package was used to generate a generic EDIF netlist from the VHDL description of ***figure 3.5***. This netlist was then translated in to a schematic for the Veribest CAE package, the top level architecture of which is shown in ***figure 3.6***. The adders use a ripple carry. The first two bits are summed and the carry is passed on to the next section of the adder, while simultaneously the sum is passed to the next adder. This produces a pipeline effect, considerably reducing the time to generate the total sum, S. Shifting the sum one bit to the right performs the divide by two. Extra logic is required to maintain the word length and the sign of the sum. The input data is also routed to the inverting inputs of the second set of full adders. These produce the outputs using 2s complement addition. The operation of the system is entirely mapped in to hardware. No instruction stream is required; the only control signal required is a clock to sample the output of the system.

```
USE STD_DAZIX_PRIMITIVES.ALL;
USE STD_DAZIX_OPERATIONS.ALL;
USE STD_DAZIX_STANDARD.ALL;

ENTITY TLM_NODE IS
PORT (VI1, VI2, VI3, VI4       : IN  INTEGER RANGE -35565 TO +35565;
      VR1, VR2, VR3, VR4 : OUT INTEGER RANGE -35565 TO +35565;
      CLOCK                    : IN  BIT);
END TLM_NODE;

ARCHITECTURE BEHAV_TLM_NODE OF TLM_NODE IS

BEGIN

PROCESS (CLOCK)

VARIABLE S = INTEGER RANGE -35565 TO +35565;

BEGIN

IF (CLOCK = '1') AND (CLOCK'EVENT) THEN

S := 0.5*(VI1+VI2+VI3+VI4);
VR1 <= S-VI1;
VR2 <= S-VI2;
VR3 <= S-VI3;
VR4 <= S-VI4;

END IF;

END PROCESS;

END BEHAV_TLM_NODE;
```

Figure 3.5 - VHDL Description of a TLM Node

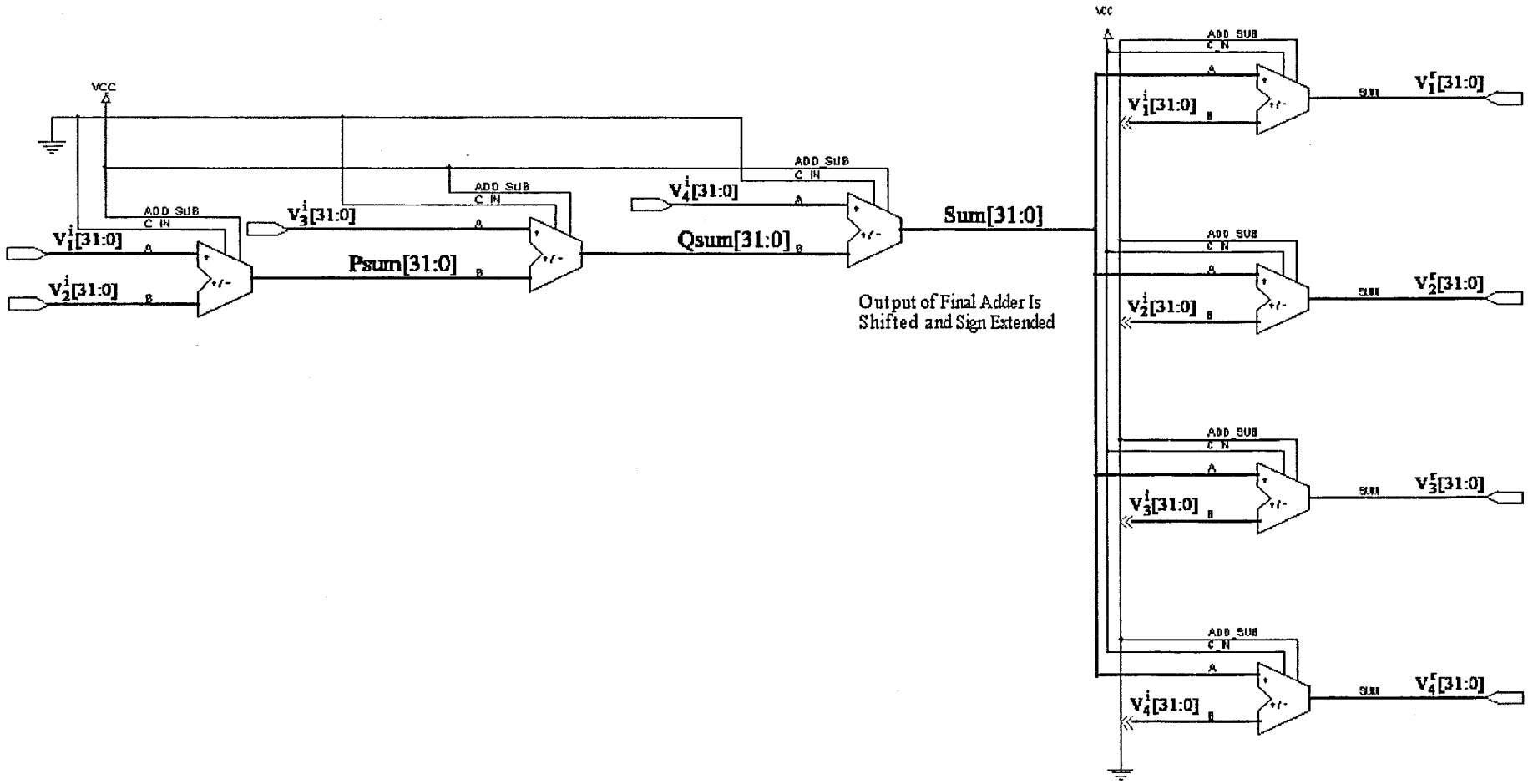


Figure 3.6 - Architecture of the Shunt Node Processor

3.5.3 Testing and Simulation

The synthesised gate level schematic may be simulated within Veribest. Timing data used by the simulator is based upon the propagation delays of the individual gates created by the synthesis package. The Xilinx place and route process compresses a number of gates in to a single CLB, considerably reducing propagation delays. Therefore timing data extracted from the schematic may potentially be greater than that of the final implementation. Simulation at this stage therefore facilitates only basic functional testing.

The schematic serves a more important purpose in that it may be used to fix parameters for the place and route (PAR) software. By setting properties attached to the components in the schematic the designer can specify maximum delays allowable for a given path, fix the location to which components are placed within the FPGA or mark out groups of gates to be partitioned together in a single CLB. Of particular importance is the ability to fix the device pins to which particular input/output nets are routed. This simplifies board level design. In this case pin location constraints were used to route the clock pin to a dedicated clock buffer input. Due to the large number of pins required by the design one of the Xilinx JTAG test port pins was also defined as a data input.

The fully annotated schematic is then translated in to a netlist in either the Xilinx Netlist Format (XNF) or the EDIF format. The Xilinx M1 software package is used to implement the netlisted design on an FPGA. First, the design is mapped to the target device, i.e. the logic is divided up and functions are compressed in to CLBs. The CLBs are then placed and routed within the device. This process may be optimised to within user defined guidelines. The layout can be tested for failed routings and the delays within the chip can be analysed via criteria such as pin to pin timings, clock to output timings etc. From this data the Xilinx timing analyser will deduce a maximum clock rate for the circuit. The nature of the programmable interconnect structure, which routes nets between fixed points means that all path lengths within the device are specified to a high degree of accuracy. As the properties of the CLBs are equally well defined the timing data generated by the timing analyser is highly accurate. User defined parameters can be used to study the effects of varying external conditions such as temperature.

The design of ***figure 3.6*** was placed and routed to produce a bitstream for an XC4025pg191. No specific criteria were attached to the design and the place and route tools were left to determine the most efficient layout based upon default parameters. Post layout timing analysis predicts a maximum clock rate of 5.75 MHz. As the circuit performs a scattering operation on its inputs on each rising clock edge, a single processor is theoretically capable of 5.75×10^6 scattering operations per second.

Once the Xilinx timing analysis has been performed the original schematic may be back annotated. The circuit's simulation netlist is rewritten to include the propagation delays calculated by the software. The circuit was simulated, including the actual device propagation delays, to ensure that no timing conflicts existed. ***Figure 3.7*** shows the results of the post layout simulation.

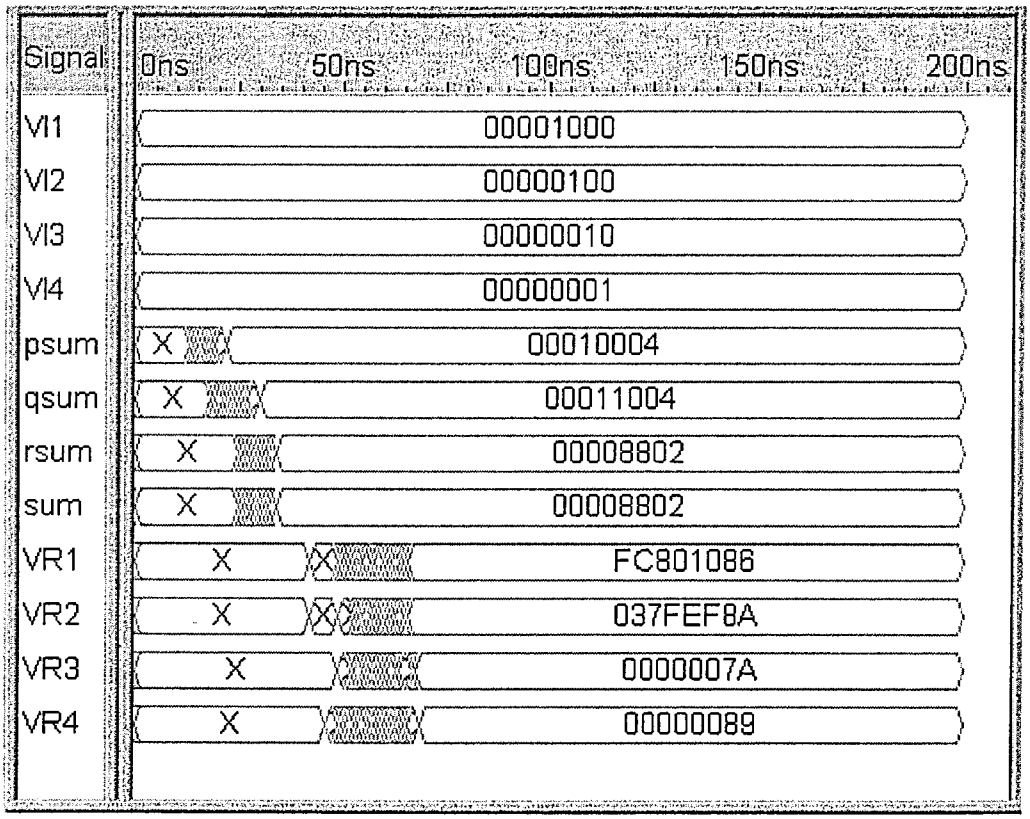


Figure 3-7 Simulation Results for the TLM Processor

3.6 Design Development

Thus far the design process has developed around a single node, however it is possible to connect a number of single node processors to form a structure similar to an SIMD array. The output pins of each node in the array are connected directly to the inputs of its near neighbours to form a 2D array. The Xilinx XC4025 has sufficient logic resources to allow the design of a 3 x 3 node array on a single chip. Such a design was modelled in VHDL using a structural description whose bottom level components were the single node processor developed above. This design was simulated to test its functionality and then synthesised in to an XNF netlist using the procedures outlined in ***figure 3.1***. As there are insufficient pins on the Xilinx to route all the outputs only a single branch of one node was used for input and output. All other branches at the edge of the model were looped back such that they simulated electric ($\rho = 1$) walls. Timing analysis of the synthesised circuit performed by the M1 software indicates a maximum clock rate of 1.31 MHz. Therefore each node in the array would perform 1.31×10^6 scattering events per second and the 9 node system would be capable of performing 11.7×10^6 scattering events per second.

3.7 Discussion

The development of an application specific processor for the two dimensional TLM shunt node has been presented. The performance of the processor implemented on a Xilinx XC4025 FPGA, ignoring the effects of any external circuitry, has been calculated as 5.75×10^6 scattering operations per second. This is an order of magnitude greater than software applications at the time of writing[11]. This represents a significant increase given that it results purely from the direct mapping of the TLM algorithm into hardware. When a limited degree of parallel processing is introduced, by extending the processor to an array of 3 x 3 nodes, the throughput approximately doubles. The figures were calculated via simulation and from the Xilinx M1 timing analyser, as the resources were not available to produce a working design in hardware. However the nature of the modular logic and fixed routing paths on the Xilinx ensure the accuracy of the information produced.

These figures clearly demonstrate that, despite increases in desktop computer performance, the idea of an application specific processor for TLM is still viable. However the design developed is far from ideal.

Although the parallel implementation presented here is of little practical importance it demonstrates that the processor may be used as part of a larger array to achieve greater throughput. Practical models in TLM often contain many tens of thousands of nodes. However, the number of nodes in the processor array is constrained by limitations of the technology on which it is implemented.

- There is a physical upper limit to how many nodes may be placed on a single chip. Large chips, with densities of 10^5 - 10^6 gates, could contain many more than nine nodes. However limitations in packaging technology restrict the number of pins available for input/output, thus restricting the size of array which may be implemented without using sophisticated interconnection strategies. A single node requires four 32 bit wide inputs and four 32 bit wide outputs, a total of 256 pins. For an M x N array of nodes, 2 x (M+N) x 32 pins are required. The array of nine nodes developed above therefore requires 384 pins. This is the minimum number of pins required to give access to all points on the edges of the array, any data routed from within the array would require further resources. A typical, large FPGA, the XC4025, has 256 pins available to the user. This is just enough to implement a single node. Even this is only possible through the use of a non-dedicated I/O pin on the Xilinx as a data input. Although the logic resource available in FPGAs is increasing rapidly, technological limitations mean that packaging size and pin counts are rising at a much slower rate. This is a fundamental limit on the technology and one that is not likely to be overcome in the near future[12]. Complex interconnection strategies may be developed[13]. Indeed the single node processor may be used as the processing element in a full SIMD array like the DECmpp12000, however these architectures have been shown in ***chapter 1*** to have their own disadvantages. Large arrays may be formed by linking several FPGAs together, however this incurs further routing delays through input/output buffering and increased path lengths and therefore degrades performance. A multiplexed input/output is another option however this may limit performance as each calculation would require multiple clock cycles.
- The word length used in the calculation is fixed by the width of the logic. Integer data has been used as this considerably simplifies the arithmetic logic required. A 32 bit word length was chosen as this provides a wide dynamic range with a high full scale signal to noise ratio. 32 bit data buses are standard in most microprocessor systems,

thus the use of 32 bit data also eases integration in to such a system. The processor can be developed to operate on any word length within pin limitations, however any changes must be made at the HDL level and the processor must be resynthesised.

- When an array of nodes is formed using fixed, hard wired interconnects, as above, there is no access to the total incident energy at each node. This is a consequence of the simple interconnect strategy. This value is commonly used for visualisation purposes.

3.8 Conclusions

The development of an application specific processor for TLM has been presented. A single two dimensional shunt node processor has been designed using the VHDL hardware description language and synthesised for fabrication on a Xilinx XC4000 series field programmable gate array. A design incorporating nine of these nodes in to a 3 x 3 node array has been developed and analysed.

This work represents the conclusion of milestones 1 and 2 as defined in ***chapter 1***. An application specific processor has been demonstrated to provide an order of magnitude performance increase over existing software based solutions. A suitable design flow has been identified by which future processors may be developed.

Comparison of Saleh's processor with the current processor shows that Saleh achieved a speed up of 27 times where as the processor detailed above produces a performance increase of an order of magnitude. This reflects both the rapid increase in performance of desktop computers and the limitations of the synthesised design.

While it has been shown that an application specific approach to reducing run times in TLM applications is still theoretically viable, the system developed here fails to address many of the aims laid out in chapter 1. The processor does not address the issue of reducing data bandwidth. Consequently the system offers scalability only at a high resource price. A practical system developed around this processor would be both costly and large, violating two of the accessibility requirements for the processor. Only a single form of the TLM algorithm is implemented.

The nine node array provides a speed up of 2.03, thus giving it an efficiency score of 22%. This compares favourably with the large scale parallel processor implementations reviewed in ***chapter 1***. The efficiency calculation assumes that sufficient bandwidth is available to supply all inputs within the allowed clock period of 76μs. This equates to a bandwidth of 159 $Mbits^{-1}$ $node^{-1}$.

The design flow identified for implementing designs with the Xilinx FPGA is thus. The circuit is specified using VHDL. The VHDL description is synthesised to produce a gate level netlist description of the circuit. If necessary this is converted to a schematic to allow properties to be added or modified. Functional simulation may be performed at this stage to ensure the synthesised design operates as expected. The netlist is then passed to the Xilinx place and route tools. These divide the logic in to small (typically ≤ 4 input, 1 output) units, which are then placed within the CLB array on the chip. The Xilinx timing analyser inspects the routing between logic blocks and produces accurate timing information. This information is added (back annotated) to the functional netlist. The design can then be re-simulated to ensure no timing conflicts exist. A bitstream is then generated to program the FPGA.

The predictability of the fixed routing network within the FPGA allows the timing analyser to give very accurate results. Post layout simulation can thus be assumed to be a very good approximation to the behaviour of the actual device under the conditions specified in the simulator. Both supply voltage and temperature effects can be varied to simulate the device under a range of conditions.

The implementation path chosen allows freedom of design. It also permits thorough design verification through simulation at all stages in the development process. The Xilinx XC4000 series FPGA has been chosen as a target device due to its low cost and reprogrammability. The internal architecture of the Xilinx device has a granularity closely matched to that of the main components in the shunt node, thus satisfying the first of the conditions laid out in ***chapter 1***. Where, as in this case, components are not available for physical device testing the Xilinx implementation software provides accurate timing data. This data may be incorporated in to a post layout simulation to test for timing violations. Processing rates of an order of magnitude greater than current software applications have been predicted for a single node. This demonstrates the viability of the approach taken in the development of the processor. VHDL offers a useful starting point in the development of the TLM algorithm in to a form suitable for

hardware implementation. However unless the code is written in a precise way it is hard to control the output of the synthesis software. This may potentially lead to a decrease in efficiency.

The development of the TLM processor has allowed a suitable implementation path and target technology to be defined. However the development of a complete working system capable of sustaining an increase in performance requires a redesign of the processor. This must take in to account the limitations highlighted above and in previous chapters.

References

[1] Saleh, A.H 'A Dedicated Processor For Solving TLM Field Problems', PhD Thesis, University of Nottingham, 1982

[2] Gregory, S 'Design of a Single Bit Processor for TLM Using Full Custom IC Design', Dissertation (BEng), University of Nottingham, 1989

[3] Price, T.E 'Introduction to VLSI Technology', Prentice Hall, New York, 1994

[4] Hosticka, B.J 'Digital Signal Processing Algorithms and VLSI Design', Proc. Of the 1st International Workshop on Digital Signal Processing Techniques Applied to Space Communications, ESTEC, Noordwijk, Netherlands, pp.191-97, 1988

[5] Pirsch, P 'Architectures for Digital Signal Processing', McGraw Hill, New York, 1998

[6] Read, J.W ed. 'Gate Arrays', Collins, London, 1985

[7] 'The Programmable Logic Data Book', Xilinx Inc. 1999

[8] Brewer, M and Hartenstein, R ed. 'Computer Hardware Description Languages and their Applications', North-Holland, Amsterdam, 1981

[9] DeMicheli, G 'Synthesis and Optimisation of Digital Circuits', McGraw Hill, New York 1994

[10] Perry, D 'VHDL', McGraw Hill, New York, 1991

[11] Jaycocks, R 'Private Communication' 1994

[12] Cypher, R 'Theoretical Aspects of VLSI Pin Limitations', SIAM Journal of Computing, Vol.22(2), pp.356-378, 1993

[13] Babb, J; Tessier, R and Agarwal, A 'Virtual Wires: Overcoming Pin Limitations in FPGA-based Logic Emulators', Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines '93 (FCCM '93), April 1993.

4 - A Bit Serial Scatter Processor

4.1 Introduction

The data parallel TLM processor presented in ***chapter 3*** demonstrates the problems associated with the very high bandwidth requirements of a TLM processor. Limitations on routing resources and the availability of I/O pins make the processor effective only in a large scale parallel array with a complex interconnect strategy. From the literature review in ***chapter 1*** this type of structure has been demonstrated to be inefficient. This illustrates an important but often overlooked point regarding TLM, that while the calculations performed are relatively trivial, the efficient routing of data between nodes is a complex problem.

A physical implementation of the processor developed in ***chapter 3*** consumes a significant percentage of the resources available within a large FPGA. As the routing resources within the FPGA are consumed, successive routing operations are forced to use increasingly less efficient paths. This can increase the propagation delays within the device and limit the maximum clock rate. This chapter presents an alternative processor design, which uses a bit serial architecture to reduce the bandwidth and resource requirements of the processor. The performance of the new design and the implications of a bit serial architecture are discussed.

4.2 Development of a Bit Serial Architecture

The processor developed in the previous chapter makes inefficient use of the resources it consumes. The major components of the processor are 32 bit wide ripple carry adders. A graphical representation of the operation operation of a ripple carry adder, ***figure 4.1***, reveals the inefficiency of their design.

The first full adder generates the sum and carry from the least significant bit (LSB) of its inputs. The sum bit is output while the carry is used to generate the sum from the second full adder. This forms the second bit of the parallel sum. This process continues between successive full adders to form the full parallel sum.

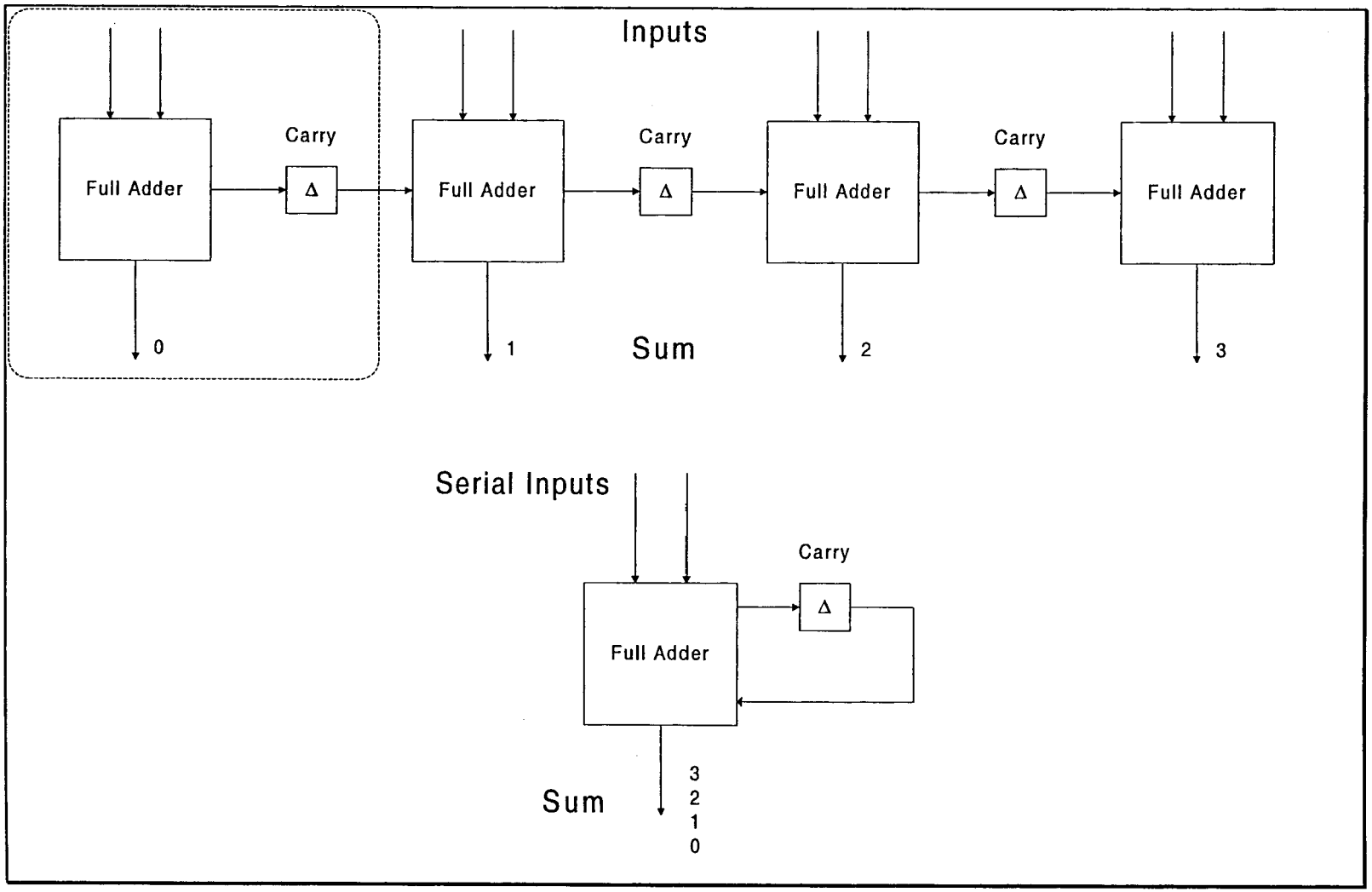


Figure 4.1 - Operation of a Ripple Carry Adder

The key elements of the operation of the parallel adder are

- Data flows in one direction through the full adder chain. Each full adder is dependent upon the carry output of its predecessor, thus they form a one dimensional pipeline.
- At any given time, the parallel adder is calculating only one bit of its output sum.

The principle of circuit folding[1] may be used to determine where the operation of multiple processing units may be implemented within a single unit using pipelined or time division multiplexed inputs. The application of this principle to the ripple carry adder demonstrates that its operation is identical to that of a single bit full adder. The chain of adders is folded on to itself. Bit n of the input is delayed by $n\Delta$, where Δ is the delay of the carry loop. The application of the folding concept to the TLM processor gives the architecture of ***figure 4.2***. The order of the summation of the inputs has been altered. The adder chain forming the total incident energy has been reduced to two stages as opposed to three in the data parallel node. This is because each stage contributes to the mean noise at the output. Reducing the number of stages therefore reduces the mean noise. The operation of the circuit remains unchanged.

Data is presented to the adders bit serially, starting with the LSB. The divide by two is performed by discarding the first bit of the sum, simulating a right shift. As it takes three clock cycles for the sum to be produced the inputs must be delayed in reaching the subtractors. Flip-flops are used as delay elements. The bit serial and data parallel architectures are mathematically identical. This becomes clear if the width of the data parallel adder operands is reduced to 1 bit and the circuits are compared. However the bit serial design has several advantages.

- Each 32 bit wide adder is replaced by a single full adder, leading to a considerable reduction in resource requirements.
- The processor is mapped to the TLM algorithm at a lower level of granularity, thus further reducing computational redundancy. Many algorithms exhibit such exploitable architectures when considered at a bit level[2].
- By using bit serial input/output the required bandwidth is significantly reduced. The number of device pins required by each processor is reduced from 256 to 11.
- The word length is no longer dependent upon the width of the adders, thus enabling the use of arbitrary word lengths.
- A single pin is sufficient to output the total energy incident upon the node for visualisation purposes.
- The ripple carry adder design receives all 32 bits of all four inputs simultaneously. The rate at which a sum is produced is dependent upon the rate at which data can flow between the stages of each adder. The time taken for the output to settle to a

steady state, which governs the maximum clock rate, is difficult to determine as quasi-steady states may appear. In the bit serial design a clock signal gates through each bit of data, thus ensuring predictable performance and eliminating the possibility of latching incorrect data.

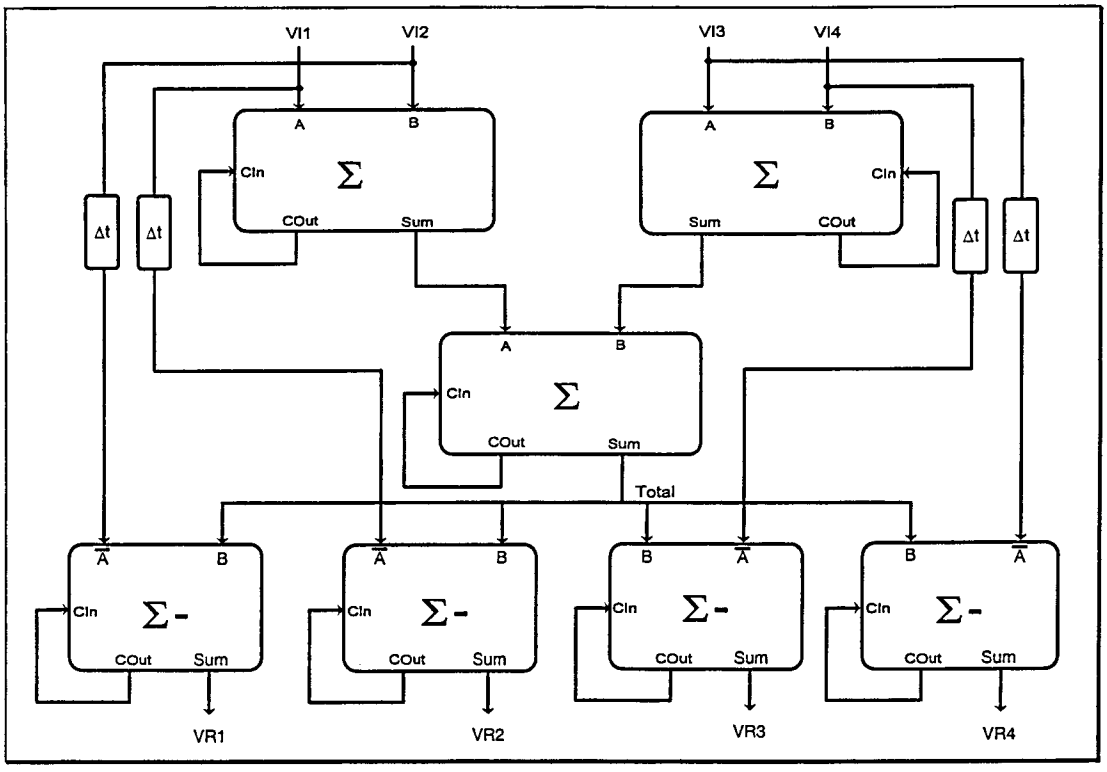


Figure 4.2 - Architecture of the Bit Serial Scatter Processor

4.3 Implementation

A bit serial scatter processor has been developed using the design flow defined in ***chapter 3***. Again the architecture of the Xilinx FPGA provides a close match to the granularity of the TLM processor. A structural VHDL description was developed from ***figure 4.2*** using a full adder as the basic component. This simplifies the code as the full adder is only defined once. The full adder is defined using boolean equations for the sum and carry functions to aid the synthesis tool in mapping the functions to the LUTs. Code listings are given in the ***appendix***.

The code has been synthesised, placed and routed on a Xilinx XC4013 FPGA. Analysis of the routed design reveals that the implementation makes inefficient use of the resources available. This is due to an intermediate stage in the synthesis procedure where the VHDL is mapped to a generic gate level description. The structure

developed at this stage does not take into account the internal structure of the Xilinx device and therefore does not translate well into an optimised Xilinx netlist. Each full adder synthesised from the generic netlist description requires 3 CLBs. However it is possible to create a full adder in a single CLB, using one four input LUT to generate the sum and the other four input LUT to generate the carry. For this reason the processor was implemented manually via the EPIC graphical editing interface. This initially presents the user with a schematic representation of the interior structure of the FPGA device, in this case an XC4013. The user may then select individual CLBs and define the contents of the LUTs via graphical means or Boolean equations. The interconnections between the CLBs may be manually defined by selecting individual programmable interconnect points within the routing matrix to guide a signal between source and sink. Although the routing may be performed automatically once source and sink pins are defined, better results were achieved using manual routing. The final design is composed of just 8 CLBs and 11 I/O pins, a considerable reduction in resources *c.f.* the data parallel design of ***chapter 3***. The sum and carry LUTs are defined using the Boolean equations

Sum.	$F = (F1 \oplus F2) \oplus F3$
Carry.	$G = (G1*G3)+(G2*G3)+(G1*G2)$

Where F and G define the outputs of the F and G LUTs, F1/G1 is input 1, F2/G2 is input 2 and F3/G3 is the carry bit formed in the previous clock cycle. For the subtractors the carry equations are modified thus.

Sum.	$F = (F1 \oplus F2\sim) \oplus F3$
Carry.	$G = (G1*G3)+(G2\sim*G3)+(G1*G2\sim)$

Where ~ indicates a logical inversion. F1/G1 is the sum of the inputs and F2/G2 is the delayed input signal.

The Xilinx timing analyser calculates a maximum clock rate of 100 MHz for the manually implemented circuit. As each clock cycle produces one bit of output data this gives a calculation rate of 3.03×10^6 scattering events per second for a single processor using 32 bit data. This is comparable with the 5.75×10^6 scattering events per second achieved by the data parallel design. Note that due to the discard bit produced during the divide by two operation, 33 clock cycles are required to produce a 32 bit output. The slight reduction in the rate of calculations compared to the data

parallel design is balanced by the reduction in circuit complexity and required bandwidth.

4.4 Expansion of the Bit Serial Architecture

The reduced bandwidth and resource requirements of the bit serial design free resources within the FPGA to introduce more advanced features to the processor. The finite storage capacity of a computer forces the TLM mesh to have a finite size, therefore the introduction of boundaries within the mesh is of particular importance. Traditionally boundaries within the TLM mesh are treated as part of the connect process[3], ***equation 1.7.*** However the modification of the data by the reflection coefficient of the boundary may be implemented within the scatter process thus removing all computational content from the connect process. This ensures that even in a coprocessor architecture all computational steps are mapped to hardware for maximum efficiency. The scatter equation thus becomes

$$ {}_{k+1}V_n^r = \rho_n \left\{ \frac{1}{2} \left[\sum_{m=1}^{4} {}_k V_m^i \right] - {}_k V_n^i \right\} \tag{4.1} $$

where ρ_n is the reflection coefficient of branch n. The connect process for that branch then becomes

$$ {}_{k+1}V_n^i(p,q) = {}_kV_n^r(p,q) \tag{4.2} $$

In order to prevent the complication of the current design through the introduction of multipliers this discussion is restricted to the implementation of boundaries with reflection coefficients of $\rho = 0, +1, -1$. Each node in the array must be aware of the location and reflection coefficient of any local boundaries. This may be achieved in one of two ways.

- Special boundary nodes are developed and placed within the array. The location of these nodes fixes the location of the boundaries
- An homogeneous array of nodes is created where all nodes are capable of representing all boundary conditions. Each node is informed of its local boundary conditions dynamically at run time.

Of these, the latter is the preferable option as it allows the arbitrary placement of boundaries within the array. This conforms to the aim laid out in ***chapter 1*** stating that the processor should not limit the configuration of the TLM mesh. Even a fixed array of processors may be reconfigured to provide a wide range of modelling environments. Each node must be informed of the boundary conditions at each branch at any given time. This may be performed as part of the initialisation of the array, giving a static configuration for that simulation. A more interesting solution is to pass the boundary data to the node in each iteration, thus allowing boundaries to move or dynamically change their properties during the course of the simulation.

The simplest way to pass local boundary data to each branch in each iteration is to include it in the incident data word. Four possible states are catered for in each branch of each node. These are no boundary, or a boundary with a reflection coefficient of $\rho = 0, +1$ or -1. The type of boundary may therefore be specified by adding a two bit code to the beginning of each data word. The code is interpreted by the processor according to ***table 4.1***.

CODE	BOUNDARY
00	None
01	$\rho = +1$
10	$\rho = -1$
11	$\rho = 0$

Table 4.1 - Boundary Representation using a 2 bit Code

An external control signal indicates to the processor which part of the incident data is boundary code. This data is then used to set two flags for each branch within the processor, the active low zero flag (ZF~) and the inversion flag (IF). The first two conditions, no boundary and $\rho = +1$ require no action by the scatter processor as the data is unchanged by the boundary. When $\rho = 0$, ZF~ is asserted and the output from the subtractor for that branch is set to zero using a logical AND operation. The case of $\rho = -1$ makes use of the relationship

$$-(A-B) \equiv (B-A)$$

The boundary code '10' sets IF which is used to select which input to the subtractor is inverted. The inversion is created using an exclusive OR (XOR) operation. Any binary value XORed with '1' becomes inverted, whereas any binary value XORed

with '0' remains unchanged. One input is XORed with the active high IF, and is thus inverted. The other input is XORed with the complement of IF and thus remains unchanged. When IF is negated the second input becomes inverted. Using 2s complement notation the inverted input is subtracted from the non-inverted input. Swapping the inverted input using IF therefore changes the output of the subtractor from (A - B) to (B - A).

4.5 A Boundary Equipped Scatter Processor

The boundary method defined above has been built in to the scatter processor. As with the basic architecture a manual implementation via the EPIC editing facility was chosen for efficiency. The additional hardware required to produce a boundary equipped scatter processor is shown in ***figure 4.3***.

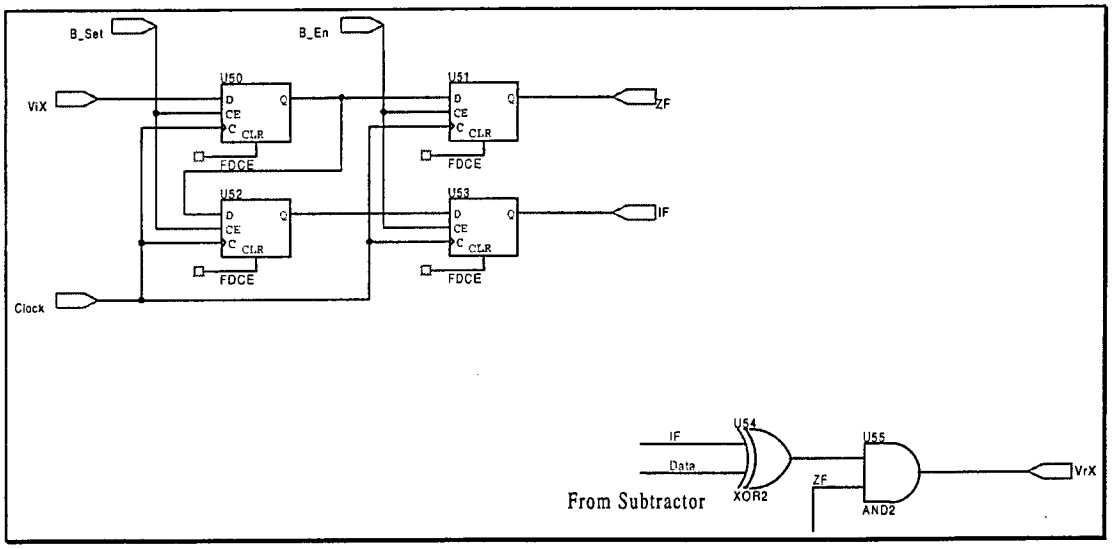


Figure 4.3 - Additional Logic for Boundary Implementation

The Boolean equations to define the sum and carry logic in the subtractor LUTs become

Sum. $F = (((F1 \oplus IF) \oplus (F2 \oplus IF\sim)) \oplus F3), \quad H = F*ZF\sim$

Carry. $G = ((G1 \oplus IF)*G3)+((G2 \oplus IF\sim)*G3)+((G1 \oplus IF)*(G2 \oplus IF\sim))$

By XORing one input with IF and the other with the logical inverse of IF, the logical state of the flag defines which input is inverted. This determines the output of the subtractor. The sum requires 5 inputs and therefore makes use of the H LUT to

provide the 5th input. A *control* signal, generated externally, is used to set the carry bits in the adders/subtractors for the first bit of each new word. The *B_Set* signal, generated from *control*, enables the flip flops which store the boundary data, thus this signal is active for the first two bits of each word. A third signal, *B_En*, also generated internally from *control* delays the activation of the new state of the boundary data. This is required as, due to the pipelined nature of the processor, the boundary data changes while some stages of the pipeline are still processing the previous input word. The boundary state associated with the old data must be preserved until this data has cleared the pipeline. This technique also prevents the activation of intermediate values formed as the boundary registers are serially loaded. As the boundary flags apply only to the subtractors the new data may enter the adder stages of the pipeline before the boundary data changes. This eliminates the need for wait states between words and ensures the flow of data through the pipeline is uninterrupted.

Timing data from the Xilinx software shows that the introduction of the boundary logic reduces the maximum clock rate of the circuit to 80 MHz. Thus a single processor with 32 bit input data would perform 2.28×10^6 scattering events per second. The reduction in performance *c.f.* the basic bit serial architecture is due in part to the increased propagation delays of the circuit from the addition logic used and also to the increase in word length caused by the addition of the boundary code.

4.6 Testing

The reduced resource requirements of the bit serial scatter processor permit it to be fabricated on a Xilinx XC4010 FPGA. This is a smaller component than that required for the data parallel design. Alongside the functional simulation testing introduced in ***chapter 3*** a physical implementation of the processor has been tested.

Testing[4] has two facets, test generation, obtaining test data to confirm the circuit operation, and test verification, proving that the test results confirm the correct operation. Exhaustive test vectors must cover all possible combinations of inputs, thus for a circuit with N inputs, 2^N test vectors are required. Good testing design utilises the principle of partitioning, the so called 'divide and conquer' approach. This simplifies testing by dividing the circuit in to smaller sub-circuits, each of which may be tested individually. Partitioning aids fault location by reducing the number of components in the circuit under test (CUT) and preventing the propagation of faults

through the circuit. Efficient partitioning also reduces the number of inputs to each test partition, thus reducing the number of test vectors required for an exhaustive test.

4.6.1 Testbench Design

A controlled method of entering the test vectors and collating the results is required. A testbench, a physical or software based ('virtual testbench') device that surrounds the CUT, may be used to provide controlled test conditions. A virtual testbench has been developed by researchers at Loughborough University around a Xilinx XC4000 prototyping board using Viewlogic's Labview™ software. The XC4000 board contains an XC4010 device with connections for downloading the bitstream from a host system and testing configuration via the Xilinx readback facility. Access is provided to all I/O pins, 88 of which are connected via a PC-DIO-96 interface card to a PC running the testbench software.

The testbench is controlled through a simple user interface. Text input boxes are used to enter the locations of the configuration and input and output vector files. The configuration file tells the testbench which signals are connected to which port of the I/O card while the vector files store the test data to be input to the system and the results produced by the test. A critical signal may be selected, that is one that is updated after the others. This may be necessary particularly in the case of a driving clock signal.

4.6.2 Testing Strategy

When partitioning the circuit for testing it is important that

- Sub-circuits maintain their functional integrity. That is, the circuit is split in to its component functions and not simply in to smaller groups of logic.
- The number of inputs in each partition is minimised in order to limit the number of test vectors required.

The boundary equipped scatter processor divides in to four test partitions

- Full adder
- Subtractor
- Boundary flag logic
- Delay pipeline

The logic blocks can be functionally tested via schematic entry and simulation, however the testbench checks that the functionality of the design has been retained throughout the partition, place and route procedure. This is particularly important in this case as the design was placed and routed manually. It is possible to place and route each partition individually for testing, however the bit serial design provides an abundance of spare I/O pins which may be used to generate test points allowing each partition of the circuit to be tested *in situ*. The inputs and outputs of each sub-circuit may be routed to spare pins to provide individual control without the need for creating separate FPGA bitstreams for each partition.

By reducing the number of inputs in each partition all possible input states may be tested. As the boundary flag and control partitions contain sequential logic the range of test vectors must be limited to all valid input sequences, as the number of potential bit sequences is infinite. Valid sequences are defined by the length of the sequential logic pipeline. In the case of the boundary data only combinations of 2 bits need be tested. Test results are included in the appendix.

Having confirmed the functionality of each partition the circuit as a whole must be tested. The test vectors used to test each partition are designed to evaluate the circuit under all possible input conditions. However in the case of the whole processor the test data is required to confirm the synchronisation between the control signals and the flow of data through the processor, therefore real 16 bit input data was used and tested under all boundary conditions. The word length of the data is largely irrelevant in that the arbitrary choice of 16 bits does not affect the validity of the testing. Robust testing requires that the test data satisfy certain requirements.

- All inputs and outputs are tested with both zero and non-zero data.
- All inputs and outputs are tested with both positive and negative data.

Figure 4.4(a-d) show a graphical representation of some of the test results. It is clear from these results that the circuit behaves as expected under all boundary conditions. The testbench does not allow the clock rate to be varied and therefore does not allow the maximum clock rate of the CUT to be evaluated.

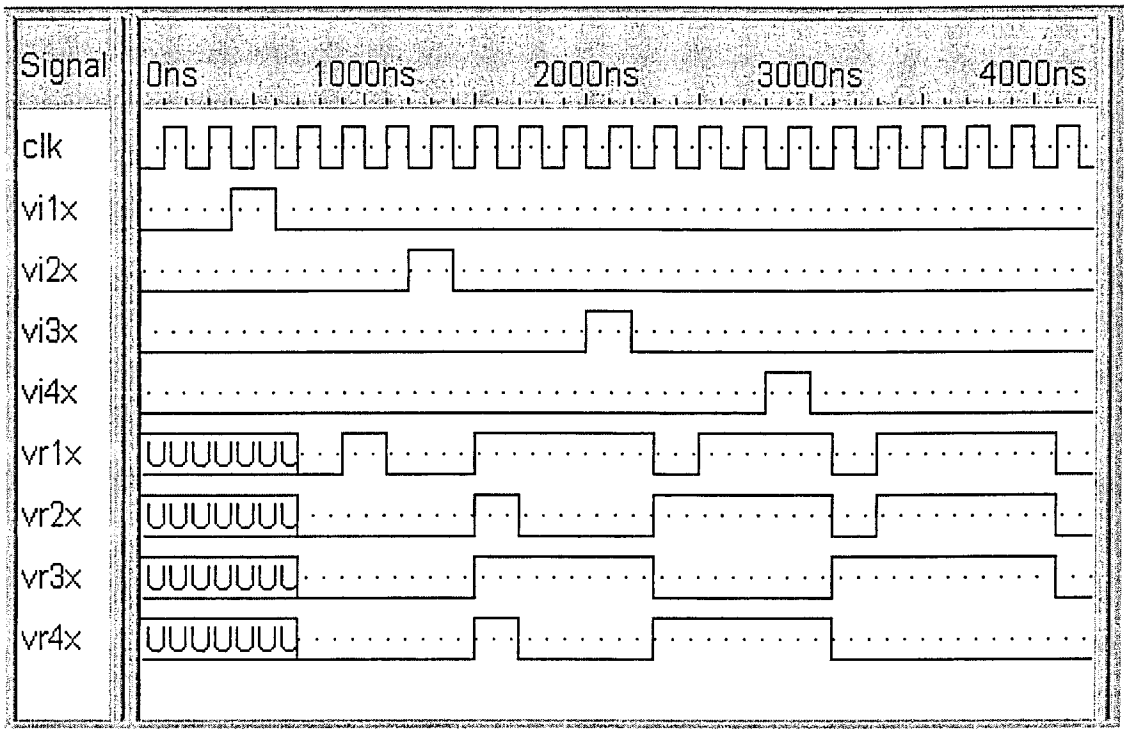


Figure 4.4(a) - Normal Operation, No Boundaries

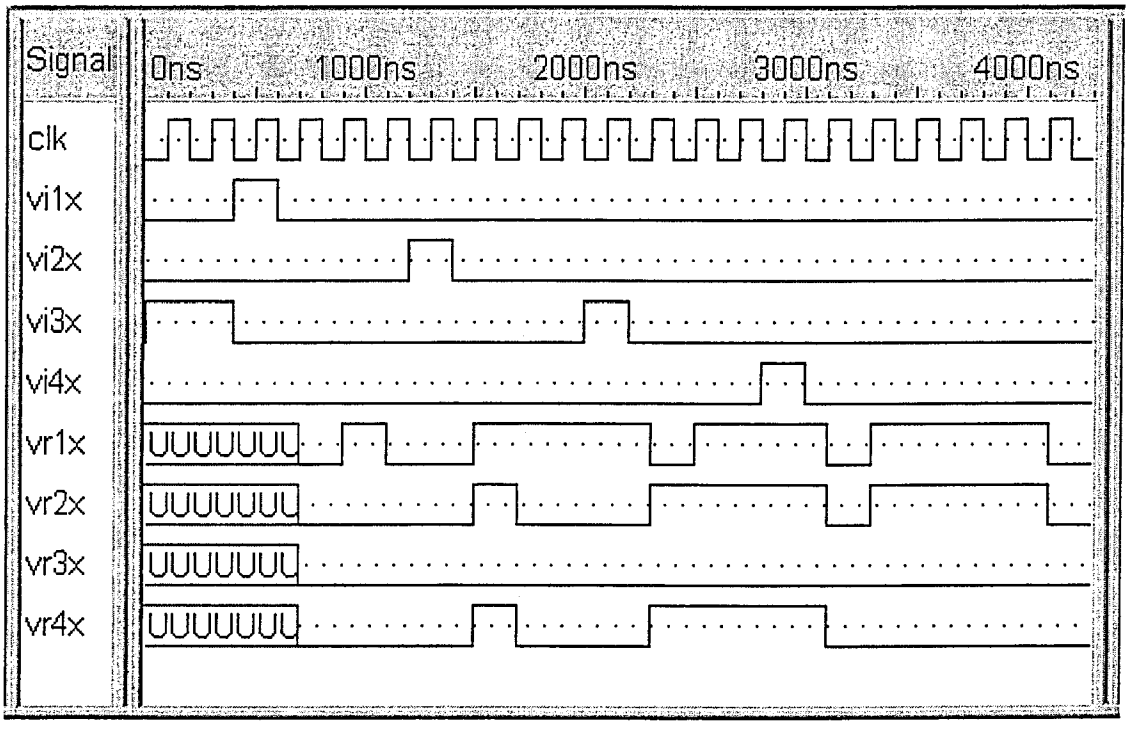


Figure 4.4(b) - ρ= 0 Boundary on Branch 3

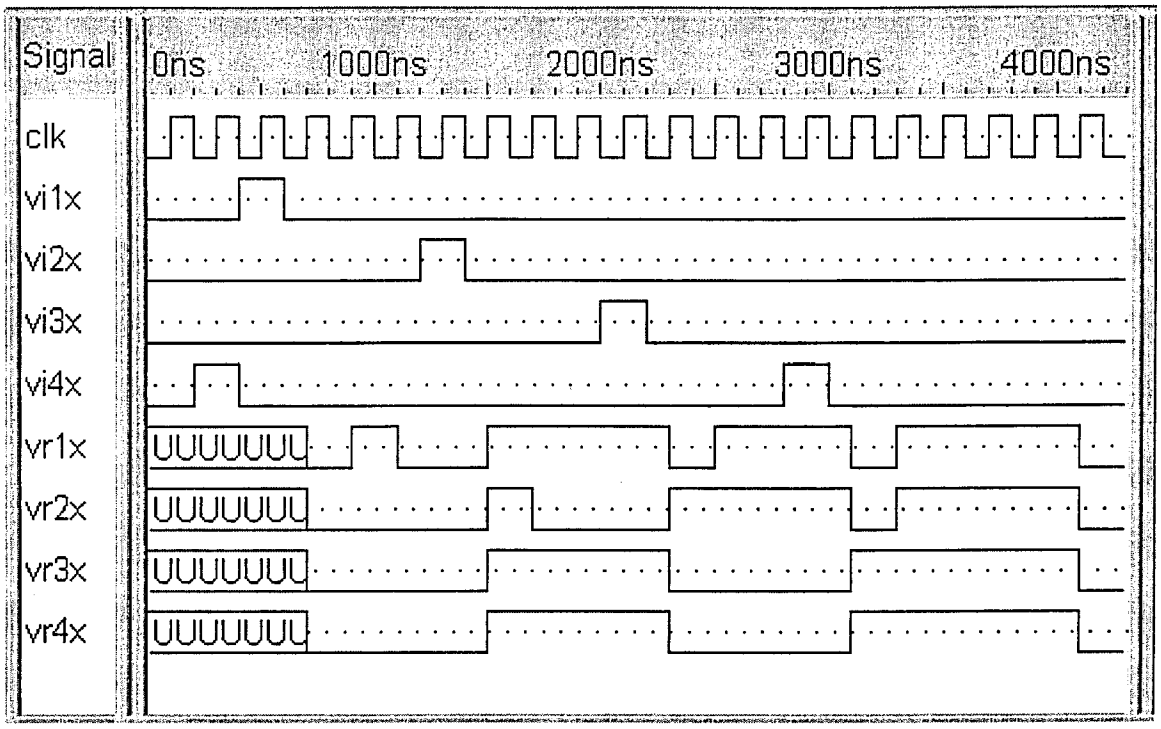


Figure 4.4(c) - ρ= -1 Boundary on Branch 4

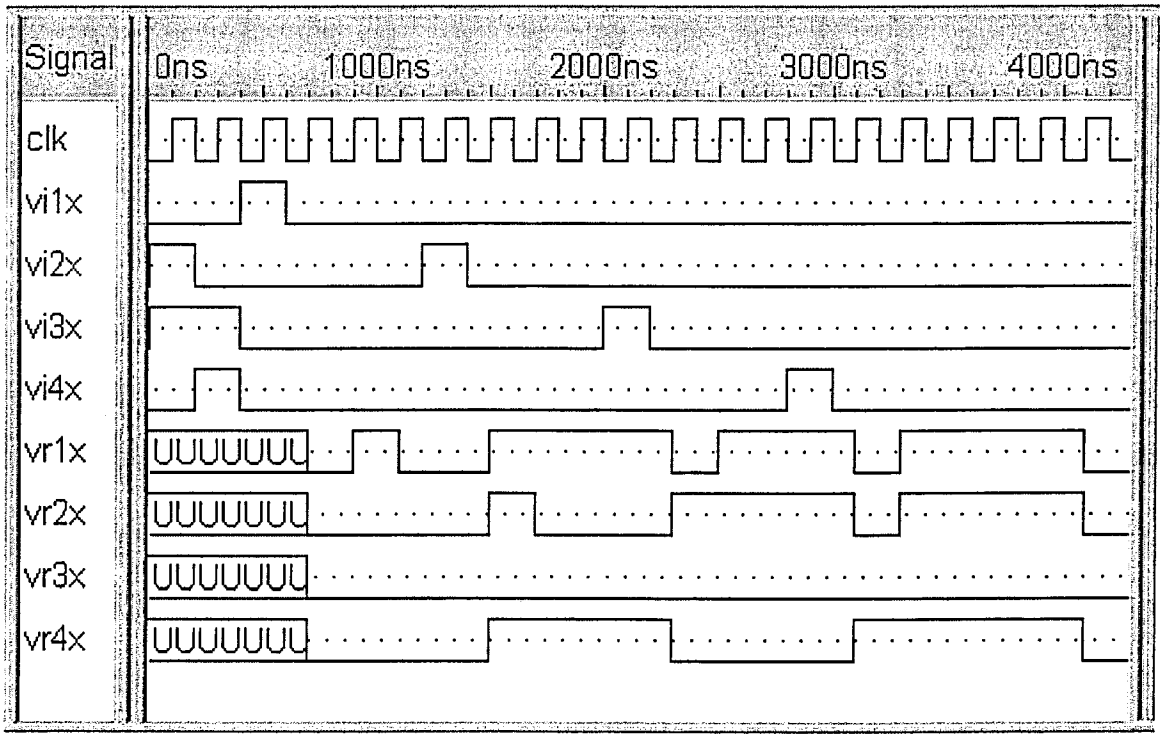


Figure 4.4(d) - All Boundary Conditions Active Concurrently, branch 1 : no boundary, branch 2 : ρ = 1, branch 3 : ρ = 0, branch 4 : ρ = -1

4.7 Discussion

The bit serial, boundary equipped processor presented in this chapter offers considerable advantages over previous data parallel designs. An 86% reduction in pin count, with a similar reduction in FPGA resources, has been achieved over the data parallel processor. The processing rate for a single node remains of the same order of magnitude as that of the parallel processor.

Mapping the node to a lower level structure within the TLM algorithm has produced a more compact processor with less computational redundancy. The bandwidth requirements of the processor are more closely matched to those provided by potential hosts, reducing the need for a complex interconnect strategy.

The bit serial design allows a single architecture to process data of arbitrary length without the need for reconfiguration. The trade-off between precision, dynamic range and throughput becomes a software, as opposed to a hardware issue. The ability to view the total energy at each node at the cost of only a single pin makes possible the visualisation of impulses propagating within the array. This is coupled with the ability to apply simple boundary conditions at arbitrary points within an homogeneous array of processors. The implementation of simple boundaries is independent of the connect process. This provides greater flexibility in the deployment of the processor. The addition of boundary data to each word increases the word length and therefore reduces throughput. For the majority of nodes within a model no local boundaries exist, therefore the addition of the boundary data provides no new information, however it must be included to preserve synchronisation with those nodes for which local boundaries are present. The extended word length and extra logic required to process the boundary data give rise to a 27% reduction in throughput *c.f.* a non boundary equipped processor. This is also due to the use of the H LUT, which adds an extra layer to the logic. The propagation delay, t_p, of this layer is encountered in each bit calculation and therefore produces a compound delay of $W.t_p$ over the whole calculation. Despite the increased delay the node is still capable of higher throughput than software based implementations

During the divide by two the data is shifted one bit to the right and the resulting fractional part, the least significant bit, is discarded. In the data parallel processor architecture of ***chapter 3*** the word length is maintained via an arithmetic shift, each bit discarded from the right is replaced by a copy of the sign bit on the left. In the bit

serial architecture the word length is not maintained during the division, rather it is left up to the connect routine to produce a copy of the sign bit.

As with the data parallel architecture, no provision is made for preventing data overflow. However, the maximum energy input to the mesh during a simulation may be calculated and the word length chosen such that this value may be accommodated. The total incident energy at any one node can never exceed the total energy within the mesh therefore the calculations will not generate overflow. The provision for arbitrary word lengths offered by the bit serial architecture makes this a realistic solution. Due to the nature of TLM the tendency is for the impulses within the mesh to diminish as the calculation progresses. However in closed systems, particularly those with constant input sources, the energy at any point within the mesh can rise considerably above the peak value of the input, thus the word length must be carefully chosen to prevent overflow in these cases without unnecessarily limiting performance. As with the data parallel design, inaccuracies due to quantisation noise may be limited through the use of a block floating point scheme. This is easier to perform at a system level by a host PC than to build in to individual processors.

It is possible to develop a floating point processor using bit serial inputs to reduce the required bandwidth per clock cycle. However the need for examination of the exponents and the shifting of data to align decimal points requires, at least in part, a data parallel approach. The data must therefore be converted to parallel within the processor. It has been shown that even a simple fixed point processor consumes a large proportion of the resources within an FPGA. A floating point processor would require even more resources per node. By using the bit serial, fixed point node a reduction in resources of approximately 80% is achieved. The advantages obtained from a reduction in resource requirement therefore force the use of a fixed point scheme.

The bit serial processor achieves several of the goals laid out in chapter 1. The bit serial architecture demonstrates an improved match between the granularity of the TLM algorithm and that of the processor. In particular the significant reduction in pin count per processor reduces the need for a complex interconnect strategy. The reduction in logic resource requirements of the bit serial design makes possible the creation of larger arrays on a single FPGA. Thus a large array could be produced on a small, cheap device. The processor also represents the delivery of milestone 3, a new design of shunt node scatter processor. The potential for a more accessible hardware accelerator for TLM is apparent.

References

[1] Huang, A 'Computational Origami: The Folding of Circuits and Systems', Applied Optics, VOL.31(26), pp.5419-5422, 1992

[2] Fetwells, G; Serra,R and Stahl,J 'On the Interaction Between DSP-Algorithms and VLSI-Architecture', Proc. Of the 1st International Workshop on Digital Signal Processing Techniques Applied to Space Communications, ESTEC, Noordwijk, Netherlands, 1988

[3] Johns, P.B and Buerle, R.L 'Numerical Solution of 2 Dimensional Scattering Problems using a Transmission Line Matrix', Proc. IEE, Vol.118(9), pp.1203-1208, 1971

[4] Williams, T.W and Parker, K.P 'Design for Testability - A Survey', Proc. IEE, Vol.71(1), pp.98-112, 1985

5. A Parallel Architecture for the TLM Processor

5.1 Introduction

The bit serial architecture TLM processor detailed in the previous chapter provides many significant advantages over previous data parallel designs. These advantages include the reduction of circuit size and pin count and the ability to implement simple boundary conditions. However, as with previous hardware implementations of TLM, the TLM connect procedure has not so far been considered in any detail. The literature review of ***chapter 1*** indicates that the connect procedure is often overlooked as a source of inefficiency.

Data transfer has overtaken processing as the main source of latency in modern computers[1]. It is therefore to be expected that an externally attached coprocessor architecture as used by Saleh[2] would be ineffective in a modern computer. A method of connecting multiple processors to form a parallel array is needed to boost performance significantly. The pipeline architecture of the bit serial processor makes it difficult to use hardwired interconnects. This is because output is generated while the input is still being processed. If the processor were placed in a traditional SIMD or systolic array some form of buffering would be required to store the results until the neighbouring nodes were ready to accept new input data. It is possible to use a FIFO as a delay pipeline, allowing the serial transfer of data without additional clock cycles. However it would be difficult to ensure the boundary data was added to the beginning of each word. The depth of the FIFO would have to be matched to the word length to remove any redundancy. This would preclude the use of variable word lengths built into the design of the processor. The limitations on mesh size imposed through the use of a hardwired array would also still be present.

In fact, a closer inspection of the data transfer process within the TLM mesh yielded a better design of connect hardware in much the same way that a review of the scatter process has led to a more efficient scatter processor.

5.2 Requirements of the Connect Process

Each scattering event within a 2D TLM mesh generates four output data words, one for each branch of the node. The connect routine, stated in ***equation 1.6***, presents these four data words as the inputs to the nodes adjoining the respective branches of the node at which the scattering took place. ***Equation 1.6*** holds true in all but the following circumstances.

- When a boundary is present on a branch the connect process is altered for that branch thus
    - ♦ The scattered data is returned in the next iteration as an input to the node from which it was scattered.
    - ♦ The value of the scattered data may be altered at the boundary due to a non-unity reflection coefficient.
- At a partially reflecting boundary a fraction, ρ, of the data is reflected as though a boundary of reflection coefficient ρ were present. The remaining $(1-\rho)$ is transmitted as though no boundary was present. Both the reflected and transmitted components may be attenuated by the boundary by reducing the reflection and transmission coefficients by the desired amount.
- At the boundary between regions of differing permittivity/permeability where internodal reflections may occur.

The design of the bit serial TLM processor allows these exceptions to be simplified when implementing boundaries. The common method of implementing boundaries within TLM, stated in ***equation 1.7***, is to perform any modification of the reflected value due to a non-unity reflection coefficient as part of the connect process. However the TLM processor developed in ***chapter 4*** performs this modification as part of the scatter process. Thus, from above, the connect process becomes a straightforward matter of routing the data to the required node. The routing may be simplified by considering only the class of reflecting boundaries for which the transmission coefficient is zero. Each scattered data impulse is then routed to a single destination.

While the above represent the requirements for the connect process to satisfy the TLM equations, the nature of the scatter processor itself imposes requirements upon the connect hardware. The connect process should support the arbitrary variability of some model parameters introduced by the bit serial node design, these being

- Data word length

- Boundary positioning
- Mesh geometry

Removing the limitation on mesh size imposed by the standard one node per processor mapping used in SIMD architectures would allow the processor to be applied to a whole range of large mesh problems in fields such as acoustics and electromagnetics. Such problems have previously been denied the use of high speed computing solutions.

5.3 System Design

Impulses within the TLM mesh travel with a finite velocity, therefore the principle of causality[3] states that in a finite time impulses generated by a scattering event can only influence other events within a finite distance. In TLM the time period of interest is Δt, the iteration time. After this time the influence of the scattering event extends over a distance Δl. Thus within a single iteration a scattering event will affect only the nodes directly adjacent to the scatterer, see ***figure 5.1a***. If scattering occurs concurrently at any number of nodes within the same row of the mesh, the influence of the scattering events after a time Δt covers only the row on which the scattering takes place and the two adjacent rows, see ***figure 5.1b***.

Conceptually, unsophisticated software solutions for TLM store all the incident mesh data in an array and use a separate temporary array to hold the scattered data. The data in the incident array is swapped for the data in the temporary array (the swap array) at the end of each iteration. If the scatter process is performed at all nodes before the connect process is performed then both arrays must be large enough to store data for all nodes in the mesh. However if the scatter and connect processes are performed concurrently within a single row of the mesh then causality demonstrates that the swap array need only be large enough to hold three rows of the model. Beyond this the current scattering event has no influence and the scattered data can be written back to the incident array to form the incident data for the next iteration. This is a minimal form for the swap array. No smaller causal array can prevent data being overwritten.

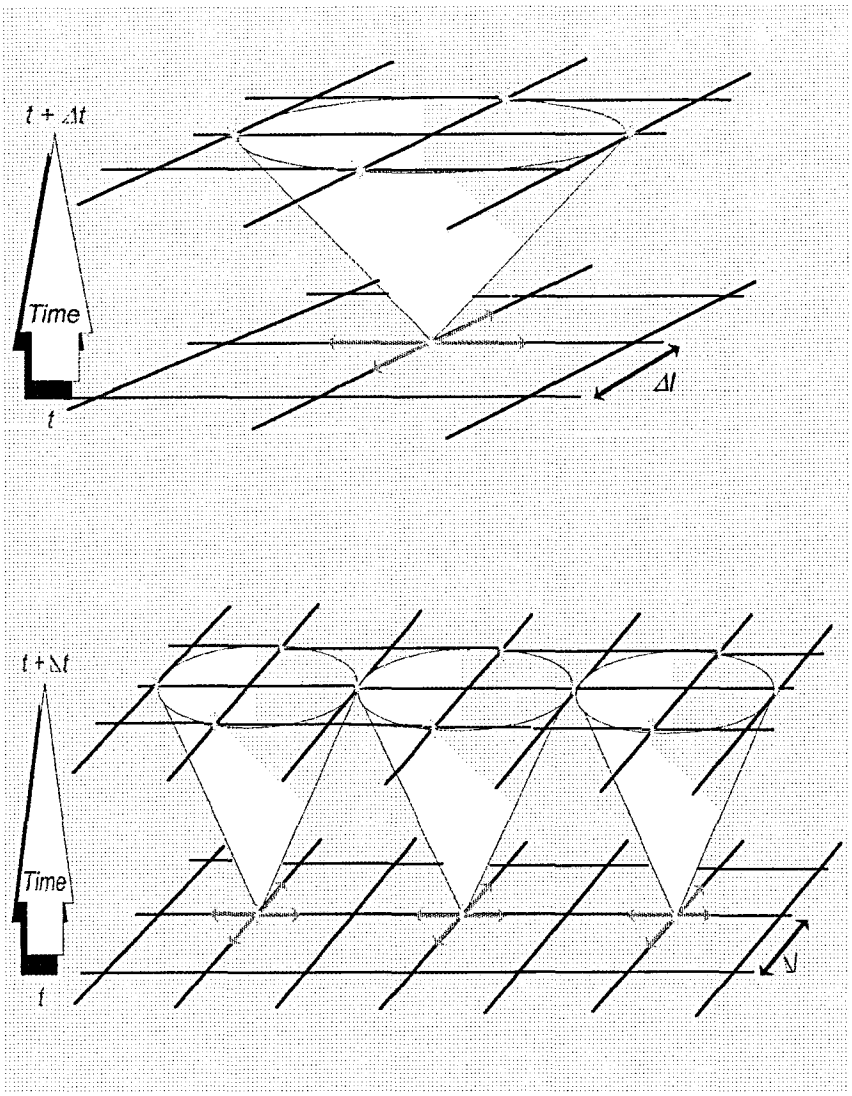


Figure 5.1 - Space-Time Diagram for a 2D TLM Scattering Event (a) After a time Δt a single scattering event may affect only adjacent nodes. (b) Scattering from several nodes in the same row will affect only three rows of the model.

Consider a serial TLM implementation in software where scattered data is written to a three row swap array. The connect process is performed by writing the scattered data to the neighbouring node locations in the swap array, thus removing the need for a separate connect routine. At the end of each row of the mesh, data in the last row of the swap array is transferred to the incident data array in preparation for the next iteration. Such a solution would make efficient use of the available memory and would require a reduced number of instructions due to the integrated scatter/connect processes. The serial execution of the software would allow the use of an arbitrary mesh geometry. It has been demonstrated above that the principle of a three row swap array extends to parallel

scattering events as long as they all occur in the same row of the model. Therefore this solution could be extended to processing partitions of the mesh up to one row long in parallel. The partially serial execution of such a system would provide an increase in throughput without placing a limit on the mesh size. This fulfils the requirements of the connect process defined in ***section 5.2***. Consequently the objectives laid out in ***chapter 1*** that the processor should not limit the mesh size or configuration are also met.

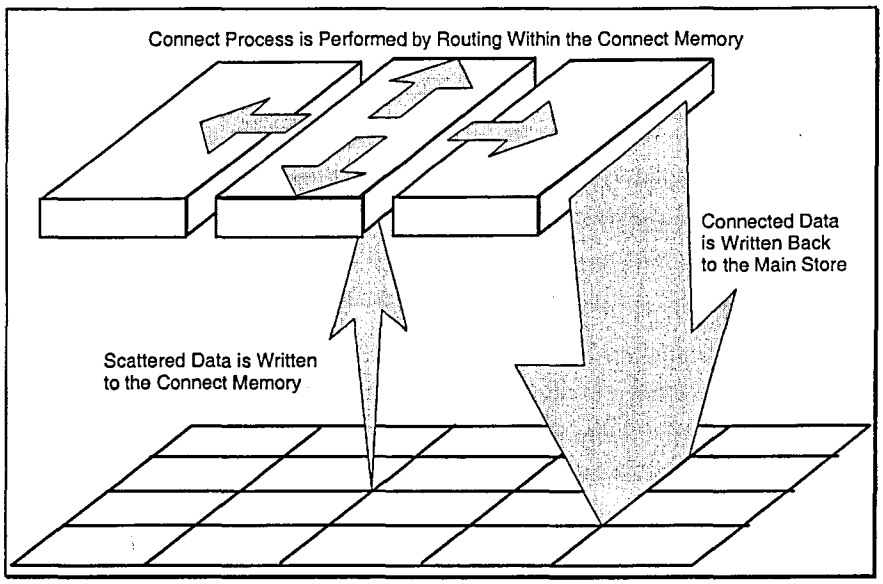


Figure 5.2a - Flow of Data Through the Connect Procedure

The above software based solution may be mapped into hardware. The incident data and swap arrays of the software are replaced by semiconductor memories respectively called the main store and connect memory. The scatter routine becomes an array of bit serial scattering processors and the connect process is mapped to routing logic. This results in the architecture of ***figure 5.2(a-b)***.

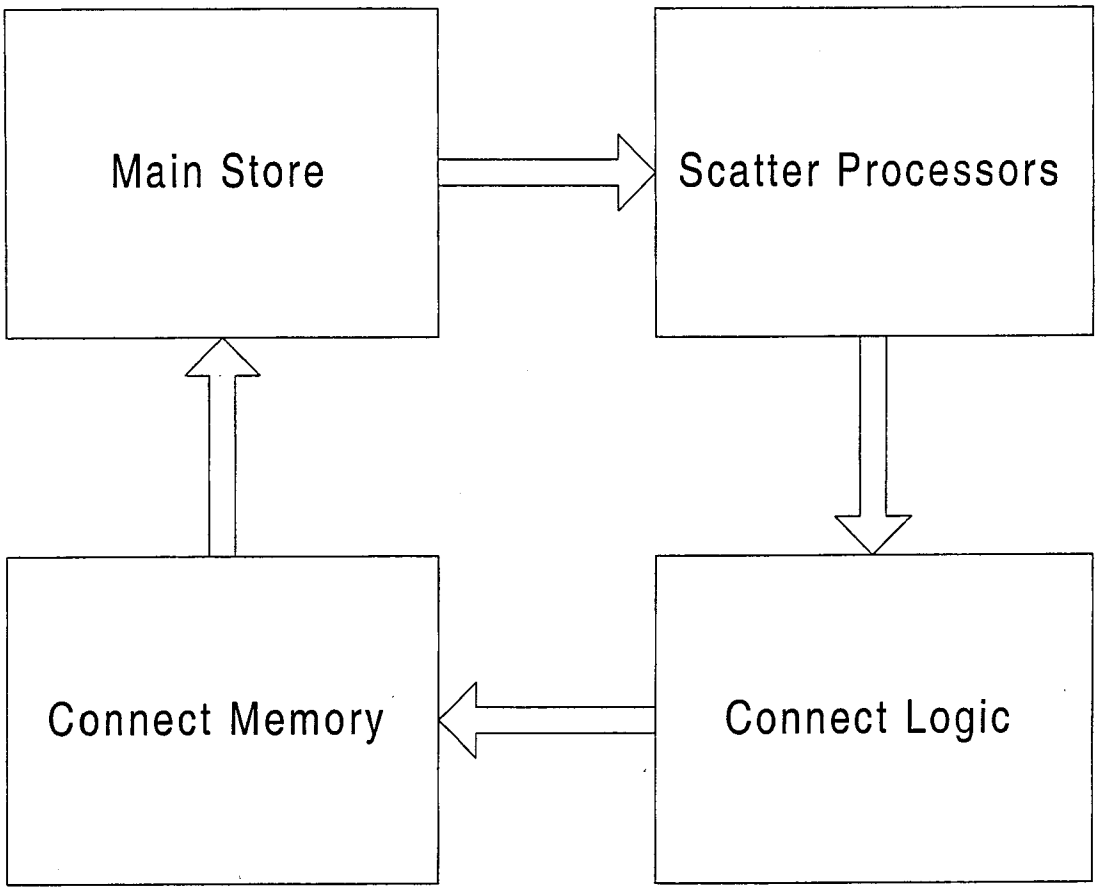


Figure 5.2b - Block Diagram of the Complete TLM System Architecture

5.3.1 Connect Memory Architecture

Consider an implementation of the above system in which an array of scatter processors is used to process a TLM mesh one row at a time. The main store memory holds the incident data array. ***Figure 5.3*** shows a memory map of the main store. The memory holds four words of data for each scatter processor. It therefore has a width of $4N^P$ bits for a system with N^P scatter processors. Each address holds a bit slice containing one bit from each of the $4N^P$ data words. Reading from consecutive addresses thus produces the input data for each processor in a bit serial format. Each group of M bit words requires M+2 addresses to hold the boundary code and the data. Consecutive blocks of M+2 addresses hold data for consecutive groups of N^P nodes within the mesh. A simple counter may be used to address the main store. This provides the data to the scatter processors in the correct order to process the entire mesh. Data scattered from a given node will be written to non-adjacent locations in the main store. The purpose of the

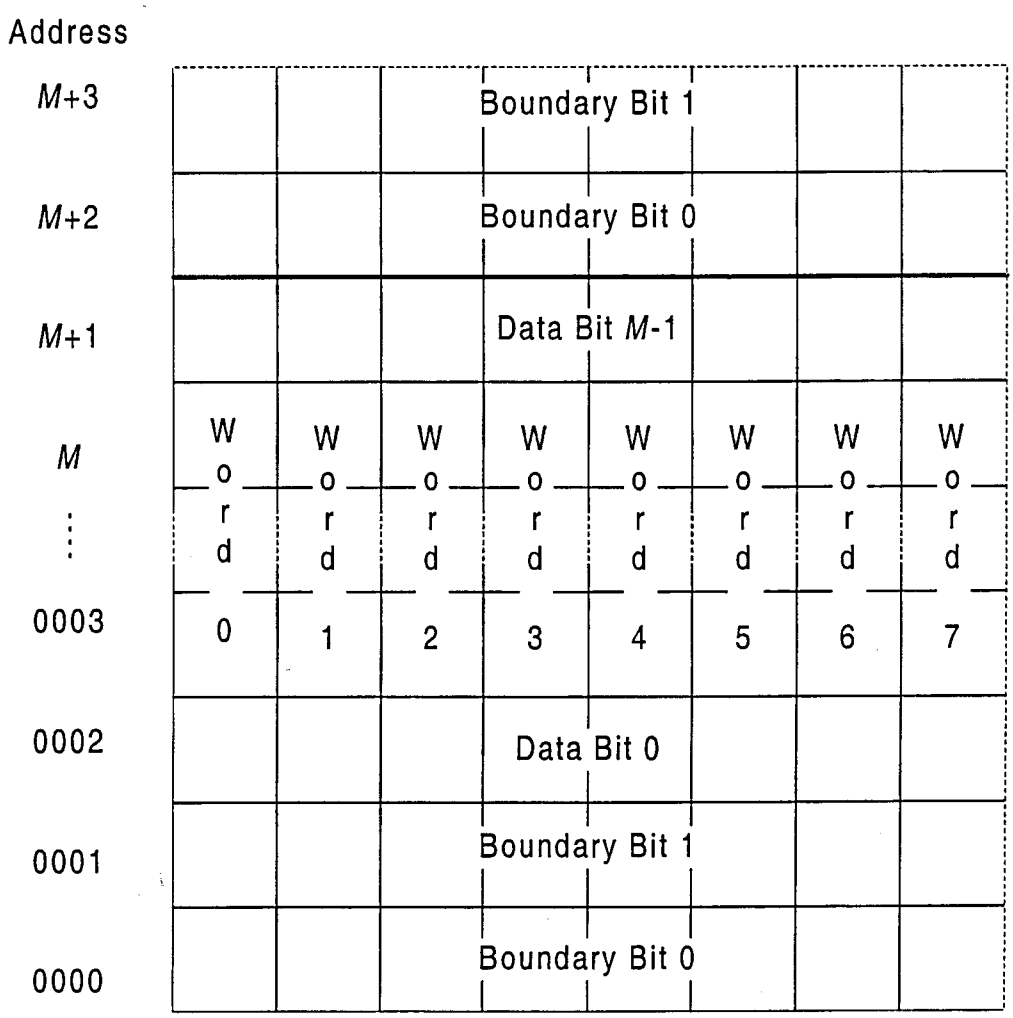


Figure 5-3 Memory Map of the Main Store Memory

connect logic is to re-order the scattered data such that it may be written back to the main store in the same ordered, bit slice format. This removes the need for a sophisticated write addressing scheme, thus minimising the processing costs of implementing the connect function.

The output from the main store is routed to an array of scatter processors. The scattered data output from the processors is routed via the connect logic to the connect memory.

5.3.2 Connect Logic Architecture

The purpose of the connect logic is to route data from the scatter processors to the correct location in the connect memory, dependent upon the boundary conditions at the scattering nodes. Each branch of each node in the array receives scattered data from one of two locations. These are the correct branch of an adjacent node (no boundary present) or the same branch of the node at which the scattering took place (boundary present). The choice of location is dependent upon the boundary data tagged to the front of each data word (***chapter 4***). The value 00 represents no boundary, therefore any non-zero code represents the presence of a boundary. A suitable circuit for the connect logic may be constructed from the basic components shown in ***figure 5.6***. The boundary data for

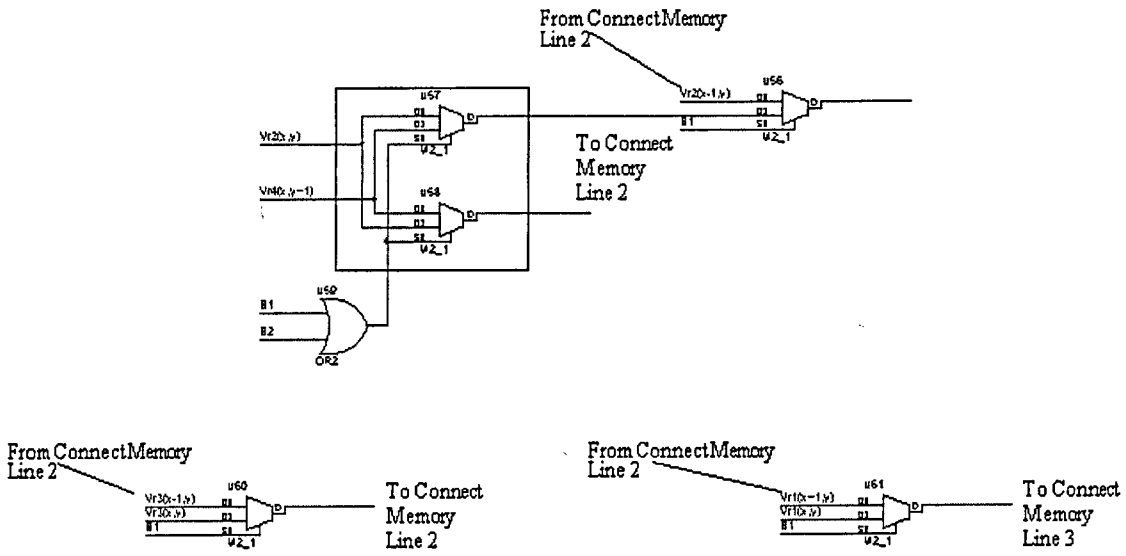


Figure 5-6 - Basic Connect Logic Components

each branch is stored in two flip-flops as it is read from the main store. An OR gate is used to generate the boundary flag for that branch if a '1' is present in the boundary data. The actual value of any non-zero boundary code is irrelevant to the connect processor as all computation is performed by the scatter processor. The boundary flag is used as the control input to a multiplexer, the output of which is routed to the connect memory. The boundary flag selects whether the memory receives data from the scatter processor or from the previous line in the connect memory (the adjacent node). In the case of data passed between nodes within the scattering row, row 2, a pair of multiplexers are used to

either swap impulses between adjacent nodes or route data back to the scattering node. A delay, generated by a short chain of flip-flops, is introduced between the OR gate and the multiplexers. This is to compensate for the time required for the incident data associated with the boundary data to pass through the scatter processor pipeline.

5.4 A Complete Application Specific System for TLM

The scatter processor of ***chapter 4*** and the connect logic and associated memory described above may be combined to produce a complete application specific processor for TLM using the architecture of ***figure 5.2b***. The incident data array is mapped to a $4N^P$ wide dual port memory of sufficient depth to store the boundary data and incident impulses for all nodes in the mesh. The nodes within the mesh are numbered starting in one corner at 0 and proceeding along the row to M^W-1 for a mesh of width M^W, the node below 0 becomes M^W, progressing along the second row to $2M^W$-1 and so on. The choice of starting corner and row direction is arbitrary. The scatter block contains some number of scatter processors, N^P, such that $N^P \leq M^W$. The main store memory is organised using the bit slice architecture of ***figure 5.3***. Processing follows a cycle of events.

- Data for the first N^P nodes is read bit serially from the incident data memory, the main store, on each rising clock edge and received by the scatter processors. The boundary code within the data is also received by the connect logic.
- After 5 clock cycles, during which the scatter processor pipeline fills and outputs the boundary data and discard bit, the first bit of scattered data is output from the scatter processors and the connect logic applies the new boundary conditions. Valid data is subsequently output from the scatter processors in each clock cycle.
- The connect logic routes the scattered data to the correct locations within the connect memory dependent upon the boundary conditions at each node.
- These steps are repeated for subsequent blocks of N^P nodes until the end of the row is reached.
- Data from memory row 3 is written back to the main store where it forms the incident data for the next iteration. Data from memory rows 1 and 2 are transferred to memory rows 2 and 3 in accordance with ***figure 5.4b***.
- The above cycle is repeated for subsequent rows of the mesh.
- When the final node is reached processing begins again at node 0, stopping when the required number of iterations have been performed.

In practice the use of dual port memory or a write through architecture allows the transfer of data between the rows of the swap array while the new scattered data is being written. The connect process then runs in parallel with the scatter process and consumes no further clock cycles. The communications overhead, shown in ***chapter 1*** to be a significant parameter in determining the efficiency of a system, is therefore zero in this case.

5.5 Discussion

The system described by the architecture of ***figure 5.2b*** forms a complete, self-contained processor for TLM. Unlike previous TLM processors both scatter and connect have been mapped to optimised hardware. The system meets the requirements for the connect process defined in ***section 5.2***. To summarise:

- The two dimensional connect process is correctly performed by the connect logic and associated memory.
- The bit serial organisation of data within the memory and the fact that consecutive groups of N^P nodes occupy consecutive blocks of memory mean that data within the main store is accessed sequentially. This allows the use of arbitrary data lengths by removing the dependence of both the address generator and memory width on the data length. The timing of the control signals by which the connect logic sets the boundary flag are data length dependent. However, as it is logical to use the same data length throughout the model, a simple counter with a reset register pre-loaded with the data length would suffice. The control logic is then activated at specific counter values.
- In order for the connect logic to work each boundary must be specified twice. Consider the mesh of ***figure 5.7***, the connect logic requires that the boundary be defined on both branch 2 of node (x+1,y) and branch 4 of node (x,y). However the reflection coefficients of the two boundaries may be different, allowing the inclusion of anisotropic boundaries within the mesh that behave differently depending upon the direction of propagation.

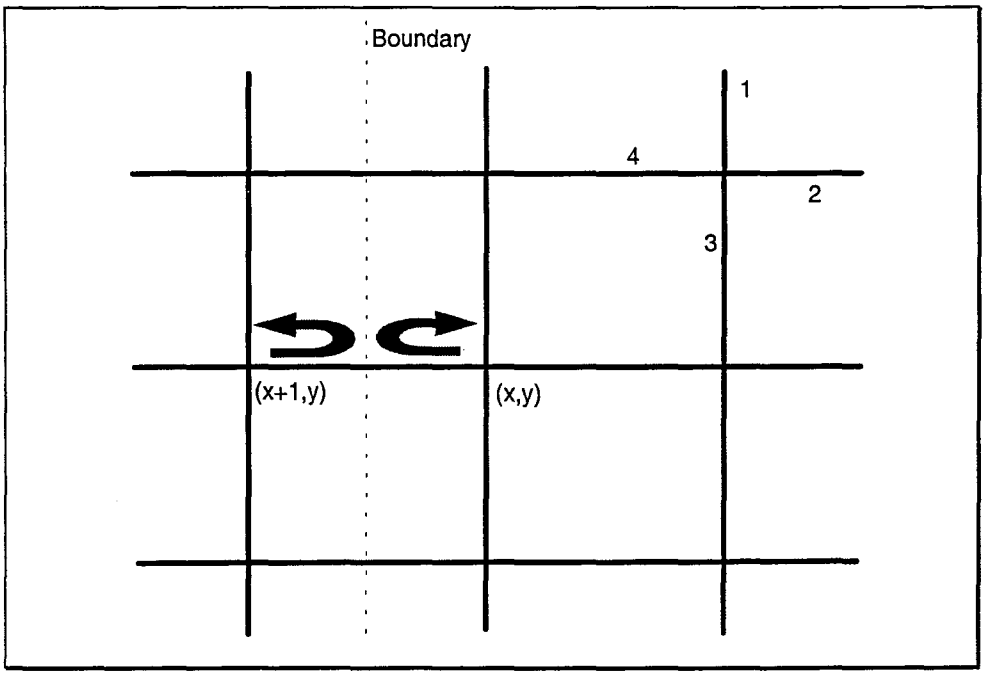


Figure 5-7 - Boundaries Must be Specified From Both Directions However Each Direction May Have Independent Properties

- The connect logic allows the connect process to be applied to a discrete region of the mesh without affecting calculations performed on the rest of the mesh. Thus scatter and connect may be performed concurrently in this region. By removing the direct interconnect schemes of traditional SIMD arrays the mesh size becomes independent of the number of scatter processors available. The system therefore has the flexibility of a serial implementation with the performance advantages of partial parallel processing of the mesh.
- Each scatter processor acts independently. The architecture of the connect logic is therefore scalable to any given number of scatter processors without penalty. There is no communication between the scatter processors. Each processor receives its input operands from memory and writes its outputs to memory, via the connect logic. Because of this the formation of an array of scatter processors does not impose any extra communication overhead to the processing time. Each processor in the array operates at the same rate as it would were it a single processor system. The efficiency of the processor array is therefore theoretically 100%.
- A block floating point scheme may be implemented by the connect processor. Normalised fixed point inputs are used, such that the maximum value on any branch of a node is 1. The maximum value of the scattered output on any branch, from ***equation 1.4***, is then 1. Overflow at the outputs of the processor will not occur with this

scheme. The discard bit formed by the divide by two may be held in the main store as the least significant bit of data. This requires 1 extra bit of storage per word. This bit is not read on a normal iteration. However if the system detects that none of the branches has significant data in its most significant bit the array may be left shifted and the block exponent decremented. The left shift is performed by offsetting the main store read address generator by one to include the discard bit. As long as the same number of bits are read the word length will be maintained and the redundant most significant bit will be dropped to facilitate the left shift. Preserving the discard bit prevents normalisation errors, which would otherwise propagate rapidly through the data. Thus the word length is maintained without decreasing the signal to noise ratio. The logic required to detect the need to shift the array may be built in to the connect logic or the scatter processor. A flag is set at the start of each iteration. The processor monitors the two most significant bits of each scattered word. These are the most significant data bit and the sign bit. If both of these bits are the same then a left shift can be performed without changing the sign or corrupting the data. If any word has different values for these two bits then the flag is cleared. If the flag is still set at the end of the iteration then the whole array can be shifted and the exponent decremented. The sensor logic required consists of exclusive OR gates to test each word, OR gates to combine the tests and a latch to store the result. Shifting the array and storing the exponent is the job of the host system.

There are however still some limitations imposed by the architecture.

- The maximum number of scatter processors is limited to M^W, the number of nodes in one row of the mesh. This limits the number of nodes that may be executed in parallel, and consequently limits ultimate performance.
- The connect process requires a regular, rectangular mesh. For optimum performance the width of the mesh, M^W, should be some integer multiple of N^P.
- The bit serial design produces a very efficient scatter processor capable of millions of operations per second. These processors generate new input and output data in each clock cycle, therefore relatively expensive, low access time SRAM is required for the main store and the connect memory to provide data at the rate required.
- The array size is limited by the capacity of the main store memory and the word length.

5.6 A High Access Time Memory Architecture

In order to process large TLM problems the system requires a large main store to hold the array. The low access time memory required for the main store in this architecture is expensive. Increasing model sizes lead to rapidly escalating system costs. Replacing the main store in the architecture of ***figure 5.2b*** with slower SRAM or DRAM memory would reduce the costs but would lead to a reduction in throughput due to the lower data transfer rates available. The design of ***section 5.4*** uses a bit serial memory organisation. An alternative is to use a data parallel memory organisation. Slower SRAM or DRAM memories may be used for the main store in this design. Impulse data is fed in to shift registers that perform a parallel to serial conversion and are capable of providing the high input data rates required by the scatter processors. Similar shift registers perform a serial to parallel conversion on the output data before it is written back to the main store. By double buffering the shift registers the scatter processors may be supplied while the data for the next N^P nodes is read from the main store. Only the main store is affected by these changes. The rest of the system remains unchanged.

In reducing the cost of the system the new architecture loses some of the flexibility inherent in the original design.

- By introducing a data parallel organisation to the main store the boundary data becomes an integral part of the data. It is no longer possible to skip this part of the data when writing back to the main store as it forms part of the current parallel word. This problem may be overcome either through the use of a read-modify-write architecture or by placing the boundary data in a separate memory.
- The size of the input and output shift registers and the width of the main store memory restrict the data length. The most efficient use of resources occurs when the memory width matches the word length. The availability of suitable components then dictates that the word length should be some multiple of 4 or 8 bits.

Several video RAM (VRAM) chips are commercially available which incorporate a wide SRAM with a fast parallel in-serial out shift register in a single package. These may be suitable for an implementation of a data parallel main store, however the problems surrounding the inclusion of boundary data into the serial data stream are exacerbated by the lack of access to taps within the shift register.

5.7 Conclusions

Two architectures have been presented for a complete application specific system for processing a two dimensional TLM mesh. The first approach utilises high speed, high cost components. The latter sacrifices some flexibility in order to reduce the cost of the system while maintaining performance levels. This is achieved by using a data parallel main store. The key to both systems is a unique mapping of the TLM connect process in to hardware. This mapping allows a fixed number of processors to operate on an array containing an arbitrarily large number of nodes. Processing sections of the array in parallel increases throughput and efficiency. However the design maintains the flexible geometry available in serial implementations. By generating an adaptable mapping of the array of scatter processors on to the mesh a small number of processors may be used to process models of arbitrary size. This is a considerable step forward from the limitations of existing large scale parallel computers. However by restricting parallel processing to partitions of the mesh throughput is reduced compared to a completely parallel architecture. There is a tradeoff between partition size, memory bandwidth and throughput. As partition size and therefore throughput increases, so the number of scatter processors must increase. The number of processors that may be accommodated within a given system is limited mainly by the databus width of the device in which the scatter processors are implemented.

Figure 5.3 shows how the required bandwidth of the main store increases with the number of scatter processors, N^P. A similar relationship exists between N and the width of the connect memories. There exists an optimum point beyond which the cost of developing wider memories outweighs the increase in throughput produced. However given the current low cost of SRAM the development of wide, shallow memories is a feasible proposal. As most systems which would be used to extract data from the TLM processor will operate on a fixed bus width, usually 8, 16 or 32 bits, there may be a need for extra logic to match a wide main store to a narrower output channel.

The system presented above performs both scatter and connect operations in parallel. As the connect process adds no overhead to the calculations the throughput of each processor in the system is equal to that of a stand alone bit serial node. This was calculated in the previous chapter to be 2.28×10^6 scattering events per second per processor, assuming 32 bit data. The efficiency of the system is therefore close to 100%. There is a slight reduction in throughput due to increased resource usage within the FPGA, so that the

efficiency never quite reaches 100%. The number of processors implemented concurrently determines the achievable speed-up. Speed-up thus becomes a factor determined by a specific implementation of the system. It is directly proportional to the number of processors implemented in the scatter array.

The complete system for the 2D shunt node represents the achievement of milestone 4. It also represents a completely new class of self-contained, application specific processors for TLM.

References

[1] Flynn, M.J 'Very High Speed Computing Systems', Proc. IEEE Vol.54(12), pp.1901-1909, 1966

[2] Saleh, A.H 'A Dedicated Processor For Solving TLM Field Problems', PhD Thesis, University of Nottingham, 1982

[3] Hawking, S 'A Brief History of Time', Transworld Publishers, London, 1988

6 A Stub Loaded Shunt Node Scatter Processor

6.1 Introduction

The preceding chapters have introduced a new class of application specific TLM processor. This processor addresses many of the shortcomings of previous implementations of TLM on parallel and application specific platforms which were described in ***chapter 1***. However in order to maximise the throughput of the processor the TLM algorithm is mapped directly to the hardware. This direct mapping restricts the new processor to a single form of TLM, the 2D shunt node.

There are four main types of TLM node. These are the 2D shunt node, the stub loaded shunt node, the symmetrical condensed node (SCN) and the symmetrical super condensed node (SSCN). The physical relationship between the shunt node and the stub loaded shunt node is clear. It was stated in ***chapter 1*** that the SCN is the equivalent of the distributed 3D node, made from a combination of shunt and series nodes. Similarly the SSCN must, under certain conditions, be identical to the SCN. The hierarchy of the nodes thus traces back to the shunt node. It is reasonable therefore to assume that the developments made in the preceding chapters regarding the shunt node may be extended to cover each of the other 3 types of node.

Some redundancy must be introduced to create a general purpose TLM processor. In order to minimise the redundancy, optimised forms must be found for implementing each of the node schemes. This has already been done for the shunt node. This chapter considers the concept of an optimised processor for the stub loaded shunt node. Subsequent chapters will study the implementation of the three dimensional nodes, the SCN and the SSCN. The aim of this study is not to build and test such a processor but rather to understand the requirements of each processor within the context of the existing system. This understanding will allow a general purpose design to be implemented more efficiently. The advantages inherent in the use of a bit serial architecture for both the scatter and connect logic have been demonstrated. The conceptual designs for the new TLM nodes must therefore draw on the shunt node design in order to preserve these advantages.

Chapter 1 introduced the concept of modelling lossy and inhomogenous media in two dimensions by adding stubs to the basic two dimensional shunt node. This chapter investigates the development of a new application specific processor based around this node configuration. The requirements for mapping the theory in to hardware are considered. An application specific processor for the stub loaded shunt node is developed from these requirements.

6.2 Requirements of an Application Specific Processor for the Stub Loaded Shunt Node

Section 3.2 highlighted the key considerations in the development of an application specific processor for TLM. These were algorithm development and hardware development. This section applies these considerations to the concept of an application specific processor based around the stub loaded 2D shunt node.

6.2.1 Algorithm Development

Scattering at the stub loaded shunt node may be expressed in the form

$$_{k+1}V_n^r = \left(\frac{2}{y}\left[\sum_{m=1}^{4} {}_k V_m^i + y_{0k} V_S^i\right]\right) - {}_k V_n^i \qquad (6.1)$$

The stub loaded shunt node requires more complex hardware than the basic shunt node due to the extended form of the scattering equation. Significantly the multiplication factor of ½ in ***equation 1.4*** is replaced by a factor of $2 / (4 + y_0 + g_0)$. Except in a few specific cases where $2 / (4 + y_0 + g_0) = e^{-n}$, $n = \{1,2,3...m\}$ this multiplication can no longer be performed by a shift operation. Instead a hardware multiplier is required. A number of multiplier algorithms were described in ***chapter 2***. Due to the complexity of the multiply operation *c.f.* an addition the multiplier becomes the main hardware component of the node. The optimum form of the scattering equation for hardware implementation must therefore minimise the number of multiplications performed.

The scatter equation as given in ***equation 6.1*** requires only two multiplication operations, the multiplication of the stub voltage by y_0 and the multiplication of the

total incident energy term by 2/y. The equation has a form similar to that of ***equation 1.5***, which has been shown in previous chapters to map efficiently onto hardware. No form of ***equation 6.1*** with less than two multiplications exists thus this form minimises the number of components required. However, while the number of multipliers used is an important consideration, the form of the algorithm used can impact upon the design in other ways.

With the inclusion of multipliers to the stub loaded shunt node processor the potential for the creation and propagation of truncation errors increases. This may be reduced through careful choice of numerical representation.

In order to select a suitable representation scheme for data within the processor the range of values to be represented must be known. Consider a closed system in which the total energy input to the system is less than unity. For simplicity the system is a single node with $\rho = 1$ boundaries on all four branches and a single stub of normalised admittance y_0. Consider an initial impulse applied to each of the four main branches of the node.

$$V_1 = V_2 = V_3 = V_4 = V_X$$
$$V_S = 0$$

After a single iteration this yields

$$V_{TOT} = V_1 + V_2 + V_3 + V_4 + V_S = 4 \cdot V_X$$

$$V_1^r = V_2^r = V_3^r = V_4^r = (4y - 1)V_X$$
$$V_S^r = 4y\,V_X$$

Where $y = \dfrac{2}{4 + y_0}$

This may be expanded to yield

$$V_{1-4}^r = \left[\left(\frac{8}{4 + y_0}\right) - 1\right]V_X$$
$$V_S^r = \left(\frac{8}{4 + y_0}\right)V_X$$

Since

$$\varepsilon_r \equiv 1 + \frac{y_0}{4}$$

We find that

$$\left.\begin{aligned} 0 \le V^r_{1-4} \le V_X \\ -V_X \le V^r_{1-4} < 0 \end{aligned}\right\} \begin{aligned} \varepsilon_r \le 2 \\ \varepsilon_r > 2 \end{aligned}$$

$$0 < V_S < 2V_X$$

In the following iteration the total energy incident upon the node becomes

$$\begin{aligned} V_{TOT} &= 4\left[\left(\frac{8}{4 + y_0}\right) - 1\right]V_X + \left(\frac{8\,y_0}{4 + y_0}\right)V_X \\ &= \left(4 + y_o\right)\left(\frac{8}{4 + y_0}\right)V_X - 4V_X \\ &= 4V_X \end{aligned}$$

Thus the total energy within the system is conserved. With the introduction of loss stubs the total energy will decrease but under no circumstances may it increase. If we impose the condition $V_{TOT} < 1$, the maximum value of the impulse scattered in to any given branch is also less than 1 and, from above, the maximum impulse scattered in to the stub is ½. A fixed point data scheme is therefore suitable for storing the total energy and the data for each branch of the node, including the stub. The concept of an implied fractional point in binary data was introduced in ***chapter 2***. By storing fixed length data such that the implied point is at the extreme left a wholly fractional number is formed. Impulses within the mesh may be stored in this way by normalising initial inputs with respect to a maximum total energy value of 1.

It is clear from the above that if $y_0 > 2$ intermediate values in the calculation of ***equation 6.1*** will be greater than one. These must be stored using a different representation, potentially one in which the fractional point lies at some point within the number. The range of numbers required will determine the placement of the fractional point. From above the maximum value scattered in to the stub is ½, therefore the maximum value generated by the stub multiplier is $y_0/2$.

The need for a separate number representation for intermediate values may be removed by rewriting ***equation 6.1*** as

$$ {}_{k+1}V_n^r = \left(\frac{2}{y} \sum_{m=1}^{4} {}_k V_m^i + \frac{2\,y_0}{y} V_S^i \right) - {}_k V_n^i \qquad (6.2) $$

The term $2y_0/y$ is 0 when $y_0 = 0$ i.e. no stubs are present, is asymptotic to 2 as y_0 increases (See ***figure 6.1***). It was shown above that the maximum value that may be scattered in to the stub is ½, so the maximum product of the right hand side of the total energy term in ***equation 6.2*** is 1. This result

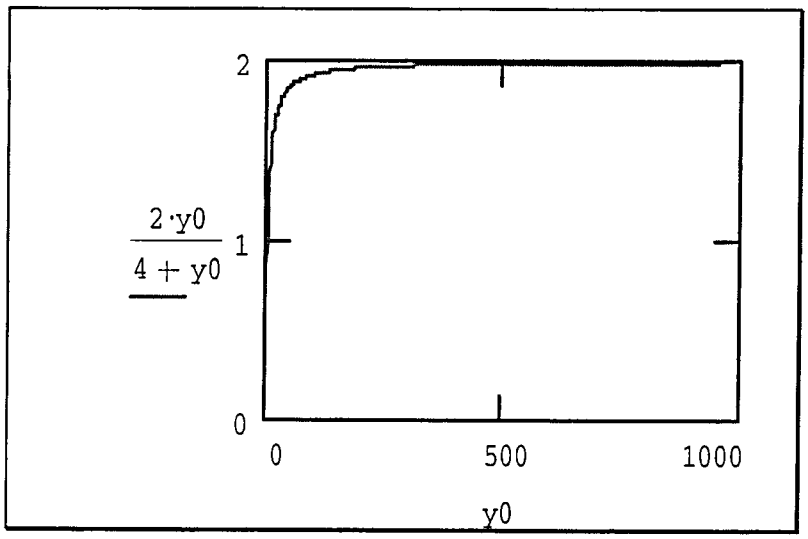


Figure 6-1 Plot of Stub Parameter 2y0/y

is consistent with the principle of conservation of energy which must be obeyed by the node. All intermediate values within the calculation may therefore be represented using a fixed point notation. The parameter $2y_0/y$ may be stored to any given accuracy by increasing the word length, W^S. No overflow errors are generated in the multiplication on the left hand side of the total energy term as the product of two numbers less than one will itself always be less than one. The fixed point multiplier contributes the same mean value to the noise level as each of the adder stages. This is the equivalent of a truncation of the output to a length of W.

6.2.2 Hardware Development

The hardware development path described in ***chapter 3*** was followed in the development of the stub loaded shunt node processor. Experience gained on the basic

shunt node processor demonstrated that in order to maximise the performance of a circuit synthesised from VHDL a structural description is preferable to a behavioural description. The structural description offers more control over the architecture of the synthesised circuit, as it may implicitly state the components to be used in performance-critical sections of the design. However if a particular form of multiplier is required a schematic based design may prove more efficient. This would allow the exact structure of the multiplier to be designed by hand instead of leaving it to the synthesis software.

Given the increased complexity of the stub loaded processor it would be counterproductive to place and route the design manually. The Xilinx Alliance software allows the user to place and route a circuit automatically and analyse the propagation delays of critical paths. These paths may then be edited manually to achieve the desired level of performance.

The advantages introduced through the use of a bit serial architecture for the TLM processor have been proven. These include a significant reduction in pin count and resource requirements and the ability to perform calculations to arbitrary precision. The stub loaded shunt node architecture needs to preserve the bit serial aspect of previous designs in order to gain these advantages. However any gains must be weighed against the increased processing time of a bit serial multiplier.

As energy scattered in to a stub is returned directly to the node from which it was scattered[Þ] there is no change in the connect routine. Preserving the use of a bit serial architecture therefore also allows the stub loaded scatter processor to interface with the connect hardware developed in ***chapter 5***. As there are significant advantages in relation to mesh geometry offered by this system, this is an important argument in favour of retaining a bit serial architecture.

6.3 System Design

A block schematic of an application specific TLM processor for modelling 2D inhomogenous media using the stub loaded shunt node is presented in ***figure 6.2***. The only noticeable change from the non-stub loaded circuit of ***figure 5.2b*** is the addition of the stub memory. Each node has one permittivity stub for which the

[Þ] Loss stubs are an exception to this rule. Energy scattered in to a loss stub is absorbed at the stub termination and is not returned to the node.

values $2y_0/y$, $2/y$ and V_S must be stored. Therefore the stub memory is $3N^P$ bits wide for a system with N^P scatter processors. Using a separate memory for the stub data leaves the architecture of the main store unchanged, providing backward compatibility with the non-stub loaded system. Data scattered in to the loss stub, if present, is absorbed and therefore does not require storage. Scattered stub data is routed directly back to the stub memory and does not pass through the connect logic. These features allow full backward compatibility with the previous system. The connect logic is unchanged by the introduction of stubs, therefore the only processing changes occur in the scatter logic.

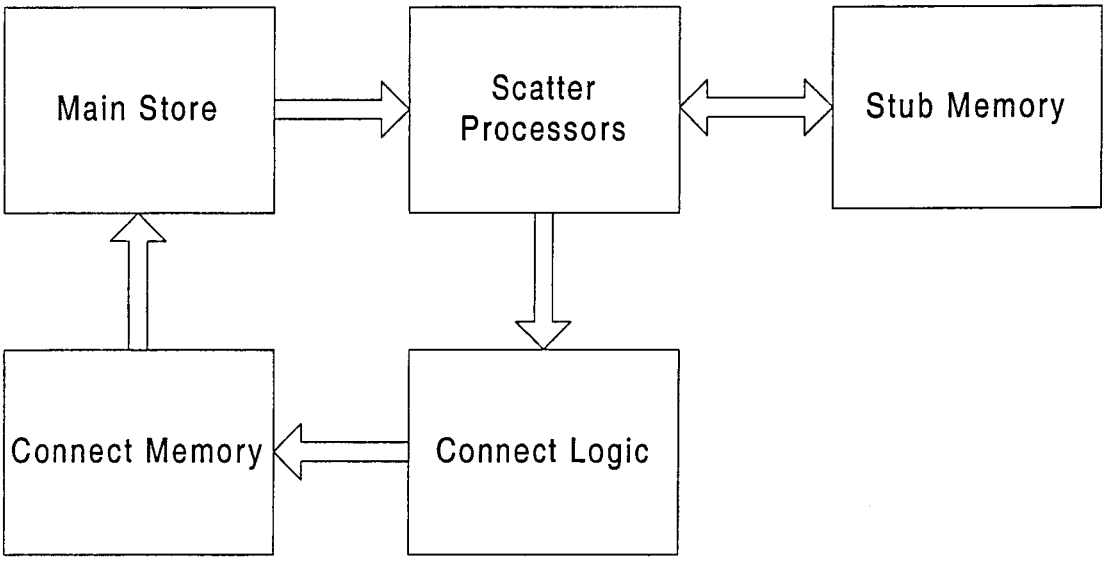


Figure 6.2 - Architecture of a Stub Loaded Shunt Node TLM Processor

Multiplication in the scatter processors is performed using the fast serial-parallel multiplier described in ***chapter 2***. By fixing the length of the stub parameters the FSP multiplier architecture remains independent of the data length used. Most genuine bit serial multipliers operate on a fixed word length, requiring all data to be sign extended to $2W$ bits long. A fast serial-parallel multiplier is therefore preferred to a genuine bit serial multiplier in order to maintain capability to arbitrarily choose the word length and to reduce the amount of data storage required. A parallel adder is used for the final summation of the bits in the carry chain to prevent the need for sign extension of the data. This also reduces processing time. The FSP multiplier is composed of full adders and delay elements. It is therefore well suited to the granularity of the Xilinx target device.

The architecture of the stub loaded scatter processor is shown in ***figure 6.3***. The processor has seven inputs. These are the four incident branch impulses, the stub voltage V_S and the stub parameters $2/y$ and $2y_0/y$. A pipeline of full adders provides

the summation of the non-stub incident impulses. Two pipeline stages are required to obtain the total incident non-stub energy. This total is multiplied by the impedance parameter 2/y while the stub voltage is multiplied with the parameter $2y_0/y$. The latter product is then routed via three delay elements to a further full adder which sums the two products to produce the product of the total incident energy and 2/y, which may be accessed via the **TOTAL** output signal. This sum is routed to the five single bit wide subtractors that generate the reflected impulse values. These are output via the five output ports to the connect logic or stub memory as appropriate. For data and stub parameter word lengths of W and W^S respectively the product will be of length $W + W^S$ bits. As only the W most significant bits of the product up to the fractional point are required the first W^S bits produced must be discarded. This may be done after the

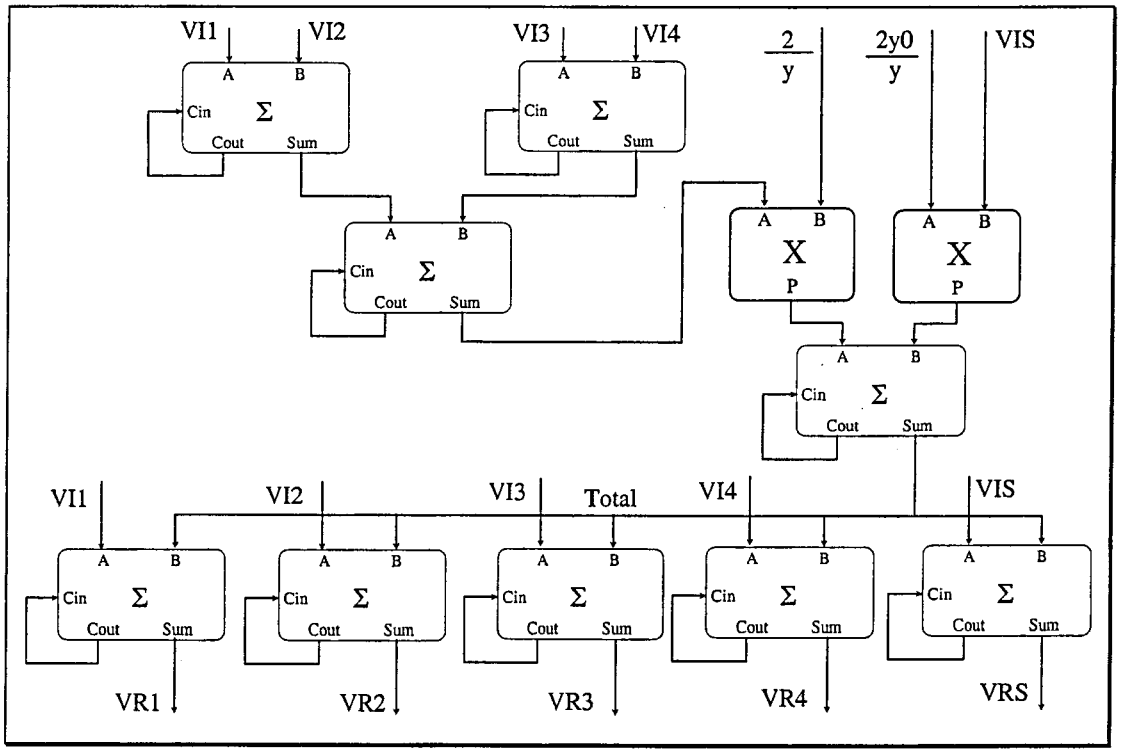


Figure 6.3 - Architecture of the Stub Loaded Shunt Node Scatter Processor

summation of the products, as the increased word length during the summation will minimise truncation errors. The processor contains 5 pipelined stages, each contributing the same amount to the mean noise level. The shunt node processor had three such pipeline stages. The mean noise at the output of the stub loaded shunt node processor is therefore approximately 5/3 that of the shunt node processor.

6.3.1 Stub Memory

The stub memory must store the values of the stub parameters $2y_0/y$ and $2/y$, and the stub voltage for all nodes in the mesh. As the stub parameters are independent of the scattered data and do not change they can be stored in a separate memory. This prevents the need for a sophisticated memory management system to ensure that constant parameter data is not overwritten as the scattered data is changed. This potential hazard arises from the bit serial data organisation. Each address in a combined parameter/data stub memory would hold both variable (stub voltage) and read only (parameter) data. The stub memory thus consists of two separate memories, an N^P bit wide SRAM which holds V_S for each node in the array and a $2N^P$ bit wide SRAM which holds the fixed stub parameters

6.4 Design Issues

- A maximum size must be set for the multiplier registers that hold the stub parameters. The size of these registers defines the precision of the stub data and the range of relative permittivity that may be modelled. ***Table 6.1*** gives values for some materials frequently encountered in electromagnetics. It is important to note that while a non-stub loaded section of the mesh represents a background medium, it need not necessarily represent a vacuum. The concept of relative permittivity must be discussed with reference to the permittivity of the background medium. The length of the multiplier register, W^S, therefore sets the range of the permittivity relative to the background medium. The register should be large enough so as not

MATERIAL	ε_r	$2y_0/y$	$2/y$
Air	1.0006	0.0012	$\approx$½
Polystyrene	2.56	1.218	0.1953
Water (Sea/Distilled)	70 / 81	1.97 / 1.9744	0.007142 / 0.00617
Silicon	11.8	1.83	0.04237
Germanium	16.0	1.875	0.03125

Table 6.1 - Relative Permittivity and Stub Parameters for Common EM Materials

to limit the accuracy of the mesh parameters, however as the product requires $W + W^S$ clock cycles to produce the register should not be made longer than necessary, thus incurring performance losses. The bit serial nature of the processor permits the use of arbitrary word lengths for the stub parameters provided the multiplier

register is sufficiently large. Therefore through reconfiguration of the registers any stub parameter precision may be accommodated.

- The performance of the stub loaded processor will be inferior to that of the simple shunt node processor due to the increased number of clock cycles required to perform the multiply operations. A total of $W + W^S$ clock cycles are required for each multiplication with an extra 4 clock cycles required for the addition/subtraction operations. The shunt node processor required only $W + 3$ clock cycles to produce an output.
- In order to simplify memory management and preserve the synchronisation of impulses within the mesh all stub parameters should be of the same word length. This may result in a loss of accuracy for smaller stub parameters. If the orders of magnitude between the largest and smallest values approaches the number of bits in the assigned word length then much of the smaller value consists of leading zeros. This may be overcome by increasing the word length, but at the cost of increased processing time.
- The inputs to the processor must be delayed so that they appear at the inverting inputs to the subtractors at the same time as the first bit of total energy summation. In the shunt node processor the inputs must be delayed for three clock cycles. The introduction of the multiplier in the stub loaded processor requires a delay of $W + W^S$ clock cycles. If W^S is large then long delay chains can be required. Additionally, the delay introduced will vary depending upon the mesh parameters; thus the delay chain must also be variable so as not to limit the range of these parameters. One way to introduce a variable delay is to use the Xilinx internal RAM capability to create a variable length FIFO.
- The multiplicand registers for the two multipliers may be double buffered so that they are pre-loaded while the current calculation is in progress. As the multiplier registers must be loaded bit serially this operation can be a considerable source of latency if pre-loading is not used.
- While the system retains the capability to vary stub parameters for each individual node, in practice the mesh parameters are usually constant over large regions. It is therefore often unnecessary to pre-load the multiplier registers for each node, rather only when local parameters change. However retaining the ability to alter the parameters for individual nodes increases the system's flexibility, as the processor is able to model gradually varying media. The arbitrary placement of boundaries between regions of differing parameters is also allowed.

6.5 A Boundary Equipped, Stub Loaded Scatter Processor

The introduction of simple boundary conditions to the scatter processor was discussed in ***chapter 4***. The use of a two bit code tagged to the front of each data word to indicate the local boundary conditions is equally applicable to the stub loaded processor. The preceding work has shown how simple boundaries, those with reflection coefficients of $\rho = 1, -1$ or 0, may be implemented through changes made to the subtractor stage of the processor pipeline. The operation of the subtractors is identical in both the stub loaded and non-stub loaded processor, so an identical system may be used. As stated above, the introduction of stubs does not affect the connect logic, and the implementation of boundaries within the connect logic is unchanged.

6.6 Performance Issues

The stub loaded shunt node scatter processor is applicable to the solution of a wide variety of problems in electromagnetics and many problems in acoustics. However in introducing the ability to model inhomogenous media the processor suffers a loss of performance. The increase in resources consumed by the node will lead to the use of less efficient routing within the FPGA, giving rise to a possible reduction in the maximum clock rate. For a word length of 32 bits plus a two bit boundary code the non-stub loaded processor produced a throughput of 2.3×10^6 scattering events per second. Assuming a stub parameter value of length 9 bits[ϖ], the stub loaded processor would be capable of 1.81×10^6 scattering events per second. This represents a reduction in throughput of 21% and assumes the maximum clock rate of the processor is not significantly reduced. A general relationship for the throughput is

$$t = \frac{1}{1.25 \times 10^{-8} \times (W + W^s + 3)}$$

As with the simple shunt node processor, the word length is increased by 3 due to the boundary data and the discard bit. This function is compared to the shunt node throughput in ***figure 6.4***.

The connect logic is unchanged from the non-stub loaded system and therefore adds no overhead to the operation of the scatter processors. The efficiency of an array of stub loaded scatter processors, as for the non-stub loaded processor, will approach 100%.

[ϖ] An implied inaccuracy in the stub parameter of ± 0.001

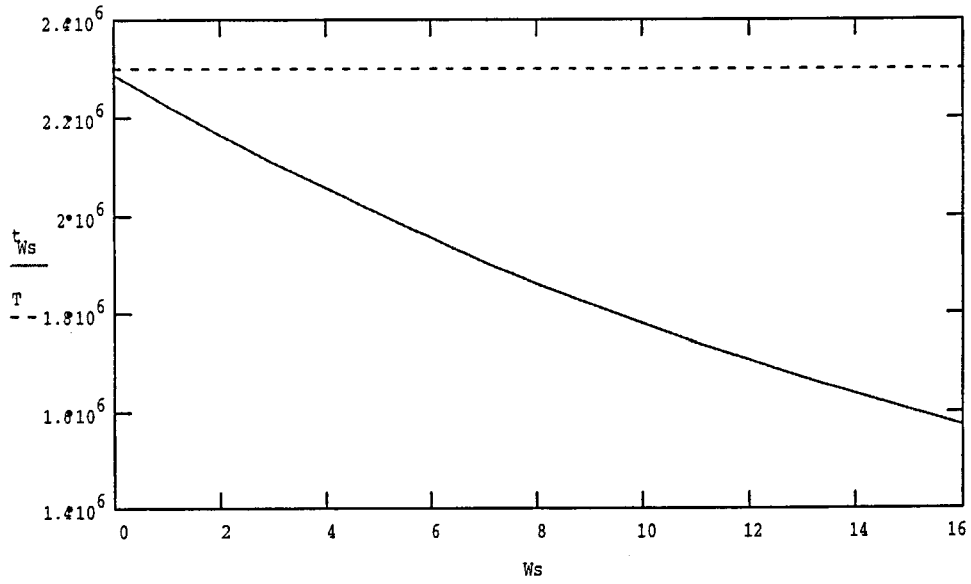


Figure 6-4 - Stub Loaded Shunt Node Throughput versus Stub Parameter Word Length. Dotted line shows shunt node throughput

6.7 Conclusions

An application specific TLM processor has been developed around the 2D stub loaded shunt node. The node is fully compatible with the connect logic developed in ***chapter 5***. Integration of the stub loaded scatter processor in to the TLM system of ***chapter 5*** requires the addition of an extra memory bank and a reconfiguration of the scatter logic. The new system is fully backward compatible with the original.

As with previous systems the stub loaded shunt node processor retains the use of arbitrary arithmetic precision and arbitrary mesh geometry. By utilising the same connect logic as the previous system the ability to process any size of rectangular mesh using a small, fixed number of processors is also retained. As with the shunt node processor, the efficiency of an array of stub loaded scatter processors integrated with the connect logic will be close to 100%.

The increase in flexibility provided by the addition of stubs to the TLM mesh is offset by a loss in performance due to the increased complexity of the arithmetic, most notably the inclusion of 2 fast serial-parallel multipliers. However this reduction in throughput is only of the order of ≈ 20-25% to the shunt node processor operating with the same precision. This is due to the increased number of clock cycles required for the stub loaded scatter operation and assumes a stub parameter word length of

between 10 and 16 bits. The word length used to store the stub parameters has a significant effect on the throughput. A balance must be struck between precision and performance.

References

7. Three Dimensional TLM Modelling using the Symmetrical Condensed Node (SCN)

7.1 Introduction

Most practical problems involving propagation phenomena exist in 3 spatial dimensions. The extension of TLM to modelling propagation in 3 dimensions has been well documented. A review of the theory has been presented in ***chapter 1***. The basic building block of a 3D electromagnetic TLM mesh is the symmetrical condensed node (SCN)[1], ***figure 7.1***. An application specific scatter processor for the SCN is developed using the basic principles of algorithm development and hardware development. The concept of a three row connect logic is extended to operation in three dimensions. These two sections are combined to produce a complete system for three dimensional TLM modelling and the system's operation is discussed.

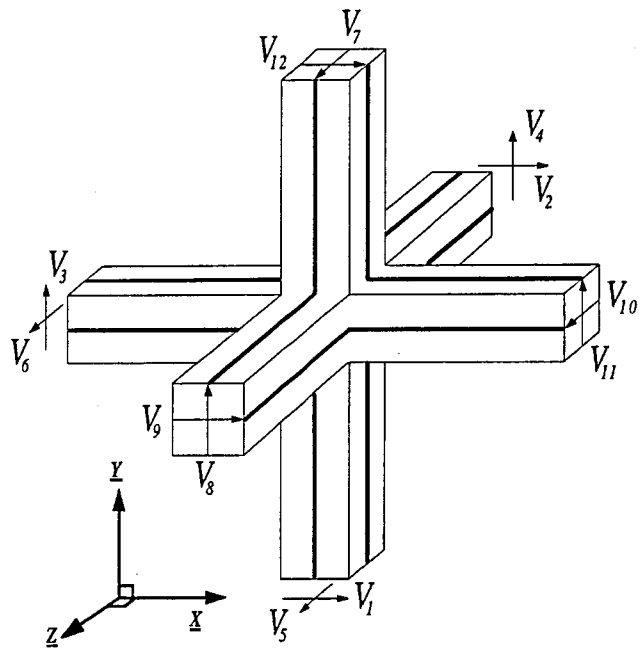


Figure 7.1 The Symmetrical Condensed Node

7.2 Development of an Application Specific 3D TLM Processor

The development of an application specific processor for the 3D TLM method requires the development of both an SCN scatter processor and a 3D connect processor. While the 2D shunt node processor of ***chapters 4*** and ***5*** presents some useful techniques which may in principle be applied to a 3D processor, significant algorithmic differences exist. Principally these lie in the following areas

- *The 3D Connect Process* - The 3D TLM method requires data scattered from a node to be transmitted to adjacent nodes in 6 directions. Data scattered within one plane of the mesh follows a pattern identical to that of 2D TLM. However 3D TLM also requires data to be transmitted to nodes in the planes above and below the scattering plane. The 2D pattern thus occurs in three planes of the model, ***figure 7.2.***

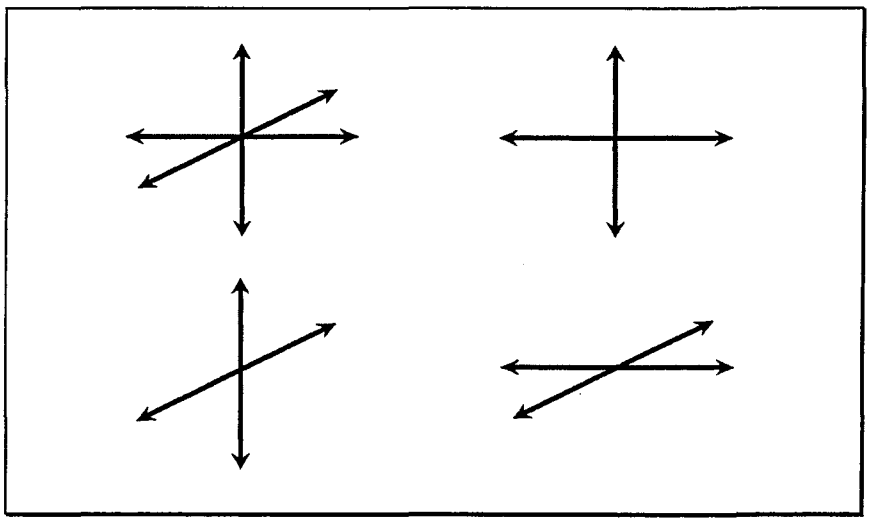


Figure 7.2 - Three Dimensional Scattering Comprises of 2D Scattering Events in Three Planes

- *2 Impulses per Link Line* - Each of the 6 link lines of the SCN is composed of two orthogonally polarised branches. The connect process must therefore transmit two words of data in each direction. Boundaries must cope with the possibility of each branch having independent boundary conditions.
- *6 Field Components Available* - In 2D TLM energy distribution within the mesh is commonly visualised by plotting the total incident voltage or current at each node. This value represents a single field component. In the SCN 6 field components may be formed from appropriate combinations of the input impulses. Any of these components may be required for visualisation and should be provided by the processor.

7.3 The SCN Scatter Processor

7.3.1 Algorithm Development

The SCN has a scattering matrix of the form

$$\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \\ V_9 \\ V_{10} \\ V_{11} \\ V_{12} \end{bmatrix}^r = \frac{1}{2} \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & -1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \\ V_9 \\ V_{10} \\ V_{11} \\ V_{12} \end{bmatrix}^i$$

Examination of the scattering matrix reveals explicit solutions of the form

$$V_1^r = \frac{1}{2}\left(V_2^i + V_3^i + V_9^i - V_{11}^i\right) \tag{7.1}$$

However there is no simple rule for determining which of the arbitrarily numbered incident port voltages combine to produce each output. A suitable scheme may be developed based upon the notation of Trenkic[2]. In this notation each port is labelled by three letters representing respectively the direction of the line, whether the direction is positive or negative with respect to the centre of the node and the port polarisation. Thus *XnY* is the Y polarised port in the negative X direction (this corresponds to port 3 in the notation of ***figure 7.1***). We can further exploit the concept of data pairing within the SCN. A cursory examination of the scattering matrix reveals that certain pairs of ports, e.g. 1 and 12, are linked. When ever one of the pair appears in any row of the scattering matrix its pair also occurs. In some cases one of the pair is negated, however in no cases are both members of the pair negated. Using the dummy indices *I, J, K* = {*x,y,z*}, it is noted that pairing occurs between ports *IpJ* and *InJ*. We can utilise these facts to produce two relationships

$$V_{IJ} = V_{IpJ} + V_{InJ}$$

$$V'_{IJ} = V_{IpJ} - V_{InJ}$$

where V_{IpJ} is the incident voltage at port *IpJ*.

The reflected port voltages are thus determined by

$$V^r_{I\{p\atop n\}J} = \frac{1}{2}(V_{KJ} \mp V'_{JI}) \tag{7.2}$$

It is noted that the expanded forms of ***equation 7.2*** correspond to the mathematically optimised SCN equations developed by Trenkic[3]. The 6 field components modelled by the node may be obtained from

$$\begin{aligned} V_I &= V_{JI} + V_{KI} \\ I_I &= V'_{JK} - V'_{KJ} \end{aligned} \tag{7.3}$$

where the electric and magnetic fields are proportional to the voltage and current respectively in the required direction. When calculating the current, *J* and *K* are chosen such that *JpK* progresses around *I* in a right handed sense.

7.3.2 Hardware Development

By expanding ***equation 7.2*** we obtain explicit solutions for each port of the SCN. These are of a similar form to the solutions of the shunt node. The principle operations performed are addition, subtraction and a division by two. Indeed each port of the SCN may be implemented using a shunt node processor given the correct input values and the selective use of a single output. This may be expected given that the SCN must be the physical equivalent of the distributed and asynchronous nodes described in ***chapter 1***. These nodes are formed from networks of shunt and series nodes.

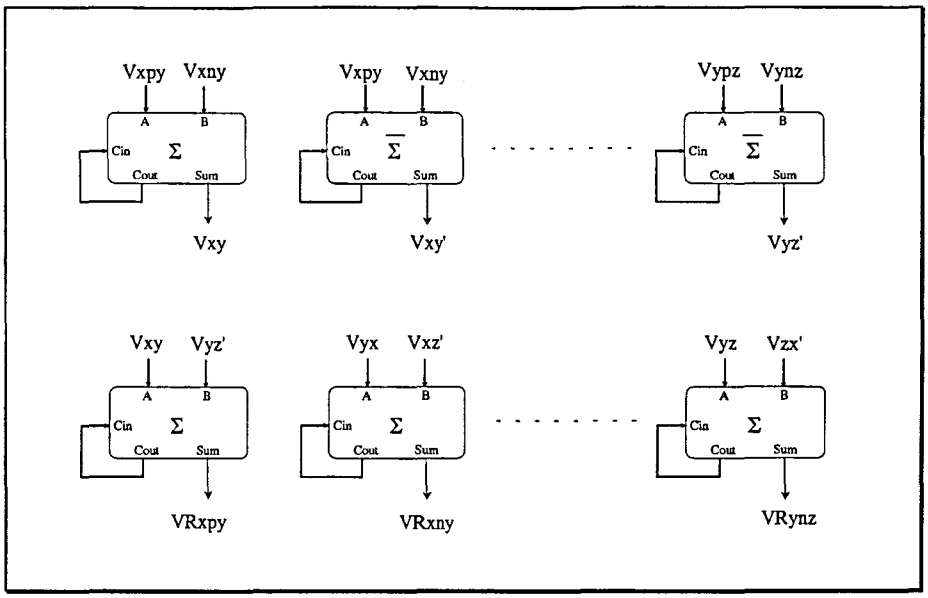


Figure 7.3 - Architecture of the SCN Scatter Processor

However the sequence of equations shown in ***equation 7.2*** suggest a much more efficient structure. The architecture of ***figure 7.3*** implements the SCN algorithm in this form. The adders are developed using a bit serial architecture as used by the shunt node processors developed previously. These components were shown in ***chapter 4*** to map efficiently to the hardware of the Xilinx FPGA. The processor retains the properties inherent in the bit serial architecture and discussed at length in previous chapters. These include the use of arbitrary precision and a significant reduction in hardware costs per processor. Each of the 12 parameters V_{IJ} and V'_{IJ} is formed in a single pipeline stage using 12 bit serial full adders/subtractors. The output values may thus be formed using a second pipeline stage to produce the correct combinations of these parameters. Again only 12 bit serial full adders are required. Each scattered data sum is divided by two by discarding the first bit of the output. This architecture provides a considerable reduction in resource requirement compared to a processor developed from a combination of shunt nodes. This is evident from the reduction in the number of adders used by the processor, 24 as opposed to 84 required for the shunt node configuration. Further reductions arise from the order in which operations are performed. In the SCN processor all subtractions are performed in the first stage of the pipeline, therefore there is no need for extra logic to produce delayed copies of the input streams. Writing the algorithm in this way also reduced the number of stages of logic in the pipeline. Each stage contributes to the total noise at the output. This architecture therefore minimises the total noise.

As the SCN processor uses identical components to those of the shunt node processor it is also well matched to the granularity of the Xilinx FPGA. A single node requires

24 CLBs and 25 pins including a clock input to control the flow of data through the pipeline.

7.3.3 Implementation of Boundaries in the SCN

Boundaries within a 3D SCN mesh are placed a distance $\Delta l/2$ from nodes to preserve the synchronisation of data within the mesh. Boundaries with reflection coefficient ρ are traditionally implemented by modifying the connection equations in the following manner.

$$ {}_{k+1}V_n^i(x, y, z) = \rho_k \, V_n^r(x, y, z) $$

It was demonstrated in the development of the shunt node processor that the computational elements of the above may be moved to the scatter processor, thus reducing the connect process to the routing of data. This eliminates any computational latency from the connect process and allows the processor array to operate at maximum throughput. However in that case the boundary calculation is performed by modifying the subtract stage in the pipeline. The architecture of the SCN does not provide a single stage at which this modification may take place, therefore a new technique is required. The use of integer data restricts the boundary conditions which may be readily implemented to those of reflection coefficients $\rho = 0$, 1, -1. A two bit code is added to the front of each data word to specify the presence and type of any local boundaries. The code specified in ***table 4.1*** may be used. This code is interpreted to set two flags, the zero flag (ZF) and the inversion flag(IF). The nature of these two flags was defined in ***chapter 4***. The output at each port is ANDed with the active low ZF and is hence forced to zero when ZF is asserted. The inversion flag must produce the 2s complement of the data output from the second stage of the pipeline. This may be generated by inverting all bits in the output and adding the result to 0001_H. This technique would obviously require an extra stage in the pipeline as an extra adder must be inserted in to the output stream. This stage would effectively subtract the output from zero. A simpler method of generating the 2s complement is presented in ***figure 7.4***. The effect of converting a

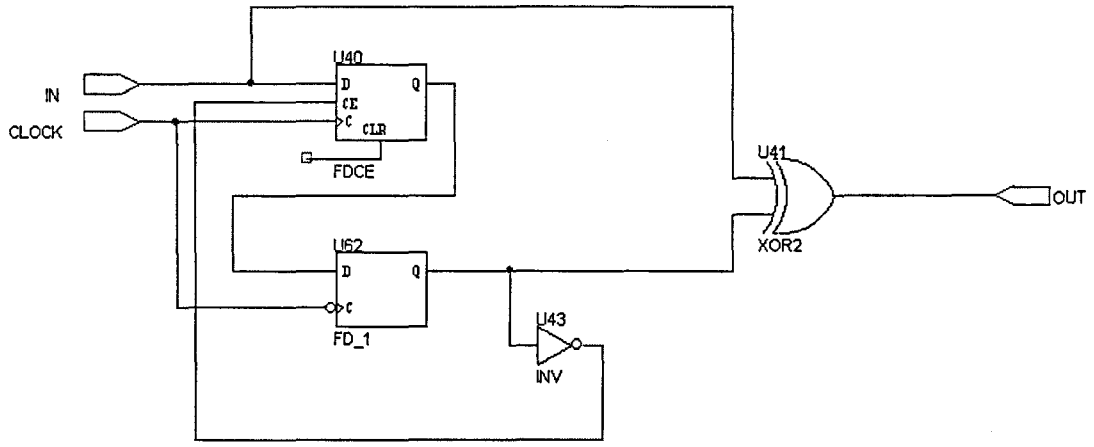


Figure 7.4 - 2s Complement Logic

number to its 2s complement is that all bits up to and including the first '1' remain unchanged while all subsequent bits are inverted e.g.

13_{10}		-13_{10}
0110(1)		1001(1)
	$\leftrightarrow$	
8_{10}		-8_{10}
0(1000)		1(1000)

The circuit operates on a similar principle to that of the subtractor modifier used in the shunt node. That is when a bit is XORed with '0' its value is unchanged where as XORing with '1' inverts the bit. The circuit uses a transparent latch, the output of which is fed back via a flip flop to form the gate signal. At the start of each data word the reset signal sets the gate to open. Any preceding zeros pass through and, as the gate signal is active low, do not change the gate conditions. The first '1' in the data word is passed through and is clocked on the next rising clock edge, latching the gate shut to prevent the data from changing again until the reset signal is applied. The output of the flip flop is ANDed with IF and XORed with the output data. Thus if IF is negated the output of the AND gate is '0' regardless of the state of the latch and the output data is unchanged. If IF is asserted the output of the AND gate becomes '1' on the clock cycle after the first '1' is presented, thus all data after the first '1' is inverted. The advantage of this circuit is that it may be compressed in to two LUTs, i.e. a single CLB, with one LUT providing the complete latch, AND gate and XOR gate and the second LUT and associated flip flop producing the gate signal. The increase in propagation delay of the circuit is therefore only ~ 5ns. A small transient signal change may occur in the output following the first active clock edge after the first '1'

is output. This is due to the propagation delay in the AND gate LUT. Careful selection of the time at which the output is sampled prevents this transient from being propagated as data.

As the boundary flags are set for each port individually it is possible to specify different boundary conditions for each of the 2 cross-polarised ports within a single branch of the node. This may be useful for modelling anisotropic or polarising materials. Indeed it is possible to specify a boundary upon one polarisation while transmitting data of the opposite polarisation unchanged.

The six field parameters modelled at the node may be produced by combining two of the 12 parameters from the first pipeline stage, ***equation 7.3***. Generating each field parameter requires a single CLB. Six output pins are required for the field parameter data, however a single pin may be used with a mutliplexer to select a single parameter for output. This method reduces the pin count and the routing resource requirement of the node. In many practical cases only a single parameter is considered, therefore the latter scheme offers an acceptable compromise between flexibility and cost reduction.

7.4 The 3D Connect Processor

With the implementation of boundary calculations in the scatter processor the design of the connect processor becomes that of a 3D data router. In an SCN mesh data is scattered in 6 directions. As with 2D TLM, impulses on the mesh travel a distance Δl in an iteration period Δt. Thus scattered impulses become incident upon neighbouring nodes in the next iteration. Consider data scattered from a row of SCNs within a single plane of a 3D model. Data is scattered to neighbouring nodes in that plane in a pattern identical to the 2D connect process. However two impulses are transferred between nodes in each direction. Along with this 2D-type scattering within the plane, impulses are scattered to neighbouring nodes perpendicular to the plane. Scattered data may be routed to a connect memory as in ***figure 5.2a***. If scattering progresses along each plane of the model one row at a time as in the 2D processor, data scattered within the plane may be transferred the 3 row connect memory developed in ***section 5.3***. This must be modified to allow for the scattering of 8 impulses within the plane as opposed to 4. Each impulse must be routed individually based upon its own local boundary conditions. Impulses scattered perpendicular to the plane must be stored until they are written back to the main store. This requires extra storage for 2 planes

in the model. A complete TLM processor architecture incorporating a 3D connect architecture is shown in ***figure 7.5***.

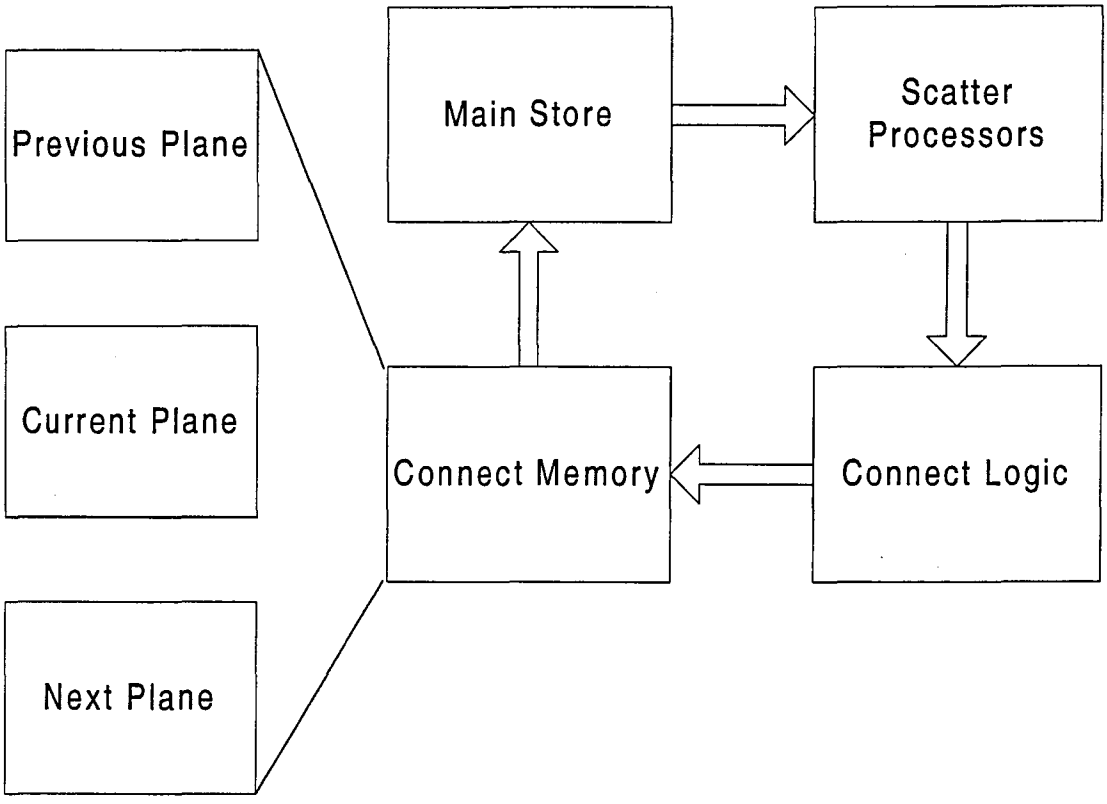


Figure 7.5 - Block Schematic of the SCN Processor

Data is passed from the main store to an array of SCN scatter processors. The scattered output data is routed via the control logic to one of three locations within the plane memory.

- *The 2D-Type Connect Memory* - This memory consists of three blocks of SRAM, each sufficient to hold 8 data words for each node in one row of one plane of the mesh.
- *The Next Plane Memory* - This memory consists of a single SRAM sufficient to hold two data words for each node in one plane of the mesh.
- *The Current Plane Memory* - This memory consists of a single block of SRAM sufficient to hold all data (12 data words) for all nodes in a single plane of the mesh.

The operation of the plane memories is analogous to the operation of the three rows in the 2D connect memory. As data is passed between the rows each location is filled such that when the data is written back the main store all locations hold pertinent data for the next iteration. Data scattered within the plane of the scatter processors is routed to locations within the 2D-type connect memory. The next plane and current

plane memories hold respectively data scattered in to the next plane to be scattered and the current plane in which scattering is taking place. Data output from row three of the 2D-type connect memory can not be written back to the main store as the scattering window, the causal region in which scattering events may affect neighbouring nodes (***section 5.3***), becomes a sphere of radius Δl in 3D. Nodes in the current plane will therefore still fall within the scattering window of nodes in the next plane. The scattered data is instead written to the current plane memory. Data previously held in this memory is written back to the main store along with any impulses passed back to the previous scatter plane. The boundary data of the two ports linking to the previous scatter plane must be held until the data is written back to the main store as it is required to multiplex the data direct from the scatter processors and data from the current plane. Data scattered to the next plane is held in the next plane memory and must also be stored until the scattering window has passed that plane for the current iteration. Data currently held in this memory is passed to the current plane memory as scattering progresses on to the next plane in the mesh. This is illustrated in ***figure 7.6***, which shows how the scattered data is accumulated for each node as the scattering event moves through the array.

The 3D connect logic minimises the amount of memory required by the processor by only storing the data that falls within the causal region of the scattering event. As with the 2D connect logic it eliminates the traditional 1 node per processor mapping. This allows a small array of scatter processors to operate on a cuboid mesh of arbitrary size. Data scattered within the current plane is processed in a manner similar to the 2D connect logic. The maximum number of scatter processors is therefore limited to the number of nodes in one row of the mesh. This ensures that the connect logic does not overwrite incident data for the current iteration.

The connect logic and main store use a bit slice architecture as introduced for the shunt node processor in ***chapter 5***. It was shown in ***chapter 5*** that such an architecture may be easily adapted to implement a block floating point scheme. The amount of storage required is increased by one bit per word to hold the discard bit. This is necessary to prevent normalisation errors when the array of data is shifted. The use of a block floating point scheme is particularly important in 3D TLM where larger arrays often require considerably more iterations to yield a solution than 2D models. The block floating point scheme effectively increases the word length to allow the storage of extra significant bits as the array is shifted. This reduces the noise added in each computation, hence the total noise added during a large number of iterations is reduced.

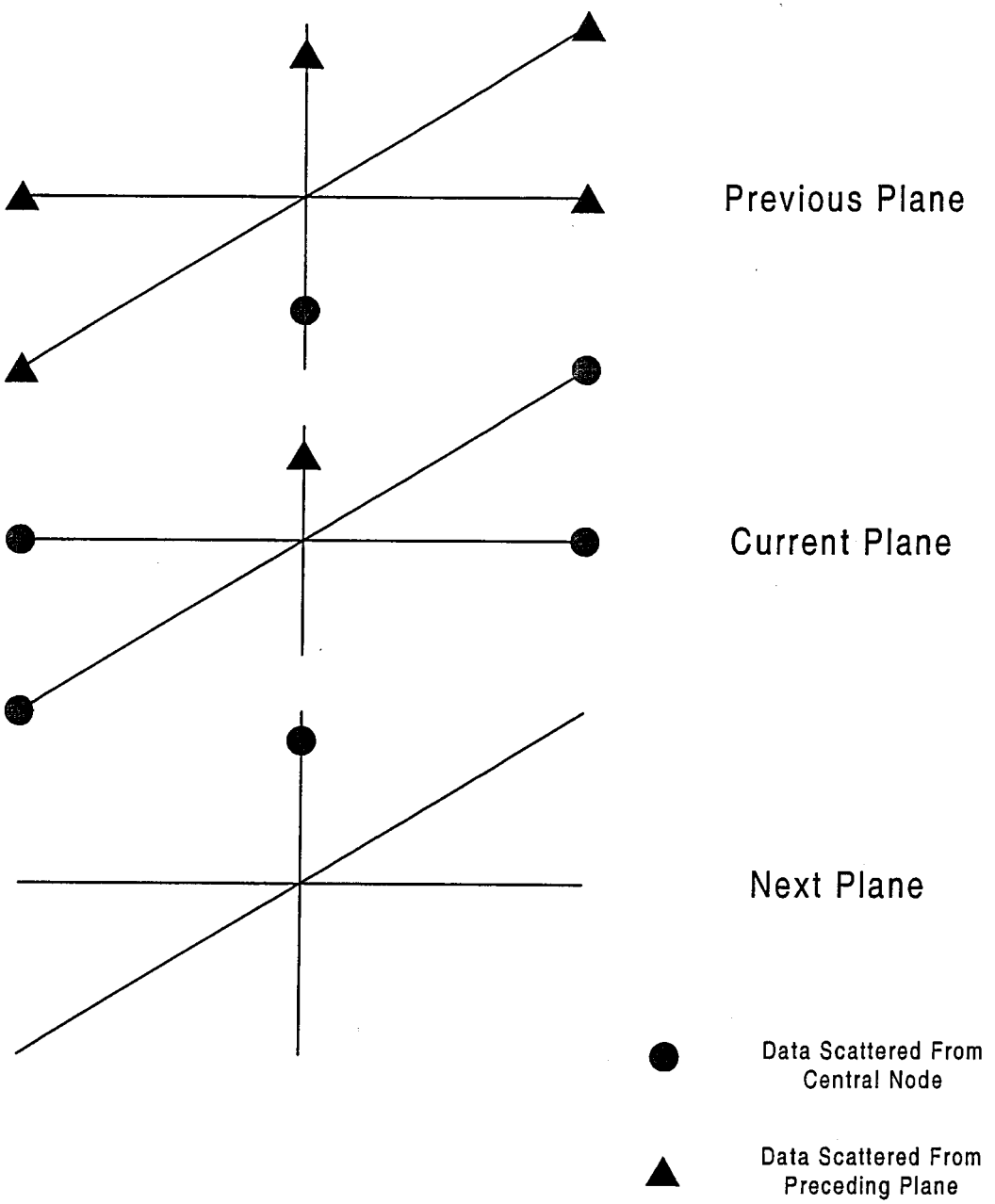


Figure 7.6 - Routing of Scattered Data Within the 3D Connect Memory

7.5 Conclusions

In this chapter the extension of the TLM processor to three dimensions using the symmetrical condensed node has been reviewed. A compact notation for scattering at the node has been developed. An application specific scatter processor has been developed to implement the SCN. By maintaining the bit serial architecture and encoded boundary representation developed in ***chapter 4*** for the 2D shunt node the

system is able to use arbitrary word lengths and arbitrary boundary placement. The 2D connect processor has been adapted to operate in three dimensions. This allows a small number of scatter processors to operate on a cuboid mesh of arbitrary size up to that permitted by the available memory of the system.

The scatter logic for the SCN processor is very similar to that of the 2D shunt node, hence throughput for a single node may be expected to be of a similar rate. Each word of length W bits requires $W+3$ clock cycles to produce an output. The SCN processor requires only two pipelined stages compared to three in the shunt node. The latency of the processor, i.e. the time between the first bit entering the pipeline and the first bit being output, is therefore reduced. Each pipeline stage contributes to the overall noise at the output, so the SCN processor can be expected to have a lower total noise than the shunt node processor.

As with the 2D processors developed in ***chapters 4*** and ***5***, no computational latency is introduced by the connect logic. The efficiency of an array of SCN scatter processors will therefore be close to 100%. Each scatter processor will operate at the rate of a stand alone processor. The connect logic uses the bit slice memory architecture introduced in ***chapter 5***. This allows a block floating point scheme to be implemented in the 3D main store in the same way as for the shunt node. The block floating point scheme, coupled with the use of arbitrary word lengths, permits the trade off between precision, range and throughput.

The extra IO requirements of the SCN processor limit the number of processors that may be formed on a single FPGA. This limits the size of the parallel processing partition and thus limits the achievable speed up. However the use of a bit serial architecture minimises the IO requirements of each processor.

References

[1] Johns, P.B 'New Symmetrical Condensed Node for the Three Dimensional Solution of Electromagnetic Wave Problems by TLM', Electronics Lett. Vol.22, pp.162-64, 1986

[2] Trenkic, V; Benson, T.m and Christopoulos, C 'Dispersion Analysis of a TLM Mesh Using a New Scattering Matrix Formulation', IEEE Microwave and Guided Wave Letters, Vol.5(2), 1995

[3] Trenkic, V; Christopoulos, C and Benson, T.M 'Efficient Computational Algorithms for TLM', 1st International Workshop on Transmission Line Matrix (TLM) Modelling : Theory and Applications, pp.77-80, University of Victoria, 1-3 Aug., 1995

8 An Application Specific Processor for Modelling Inhomogeneous Three Dimensional Media

8.1 Introduction

As with the shunt node in two dimensions, the SCN, introduced in ***chapter 7***, is capable only of modelling homogeneous media. This chapter presents an application specific processor for the modelling of inhomogeneous media in three dimensions using the stub loaded SCN. This node configuration is shown to be inefficient when implemented in hardware. An application specific processor to implement the SSCN scattering algorithm for isotropic, non-graded media is developed.

8.2 Development of a 3D TLM Processor for Generalised Media

Two different approaches to modelling inhomogeneous media in three dimensions were presented in ***chapter 1***. The first is the stub loaded SCN. As with the shunt node stubs may be added to the SCN to increase the capacitance or inductance at the node. Separate stubs are required for each dimension. Loss stubs may also be added. A total of 9 stubs may therefore be added to each node. The second approach uses a new node called the symmetrical super condensed node (SSCN). The SSCN varies the parameters of the region modelled by the node by varying the impedances of the link lines forming each branch.

8.2.1 Algorithm Development

8.2.2 Stub Loaded SCN

Stubs may be added to the SCN to model increased permittivity and permeability in the same way they are added to the shunt node. However due to the increased number of stubs the parameters y and y_0 are replaced by a series of 10 parameters. These are denoted by the lower case letters a-j. The stub loaded SCN has a densely packed 18 x 18 scattering matrix[1]. 10 parameters are stored for each homogeneous region of the mesh. It is possible to generate twelve secondary values from the incident data as done previously for the SCN

$$V_{ij} = V_{ipj} + V_{inj}$$
$$V'_{ij} = V_{ipj} - V_{inj}$$

As with the SCN inspection of the scattering matrix reveals patterns that may be used to develop compact equations for the scattering at each port. These take the form

$$\begin{aligned} V^r_{ipj} &= \hat{\delta}\, V_{ij} + b\, V_{kj} - d\, V'_{ji} + g\, V_{oi} + i\, V_{sk} \\ V^r_{inj} &= \hat{\sigma}\, V_{ij} + b\, V_{kj} + d\, V'_{ji} + g\, V_{oi} - i\, V_{sk} \end{aligned} \tag{8.1}$$

The subscripts *oi* and *sk* denote the voltages of the *i* and *k* polarised open and short circuited stubs respectively. The operators $\hat{\delta}$ and $\hat{\sigma}$ are defined as

$$\hat{\delta}V = \frac{1}{2}\left[(a + c)V + (a - c)V'\right]$$
$$\hat{\sigma}V = \frac{1}{2}\left[(a + c)V - (a - c)V'\right]$$

Reflected capacitive and inductive stub voltages may be calculated from

$$\begin{aligned} V^r_{oi} &= e\, V_i + h\, V^i_{oi} \\ V^r_{si} &= f\, I_i + j\, V^i_{si} \end{aligned} \tag{8.2}$$

Loss stub voltages may be similarly determined. V_i and I_i were defined in ***equation 7.4***.

8.2.3 Symmetrical Super Condensed Node

The SSCN[2] offers many degrees of freedom. By restricting the node to lossless, uniformly graded meshes the design may be simplified considerably. However both restricted and generalised cases will be considered.

It was shown in ***chapter 1*** that the SSCN may be solved using a series of equations of the form

$$\begin{aligned} V^r_{inj} &= V_j + I_j\, Z_{ij} - V^i_{ipj} \\ V^r_{ipj} &= V_j - I_k\, Z_{ij} - V^i_{inj} \end{aligned} \tag{8.3}$$

In order to simplify the SSCN scattering equations (***equation 8.3***) two new parameters are introduced[3].

$$C'_{ij} = \frac{Y_{ij}}{\left(Y_{ij} + Y_{kj}\right)}$$
$$L'_{ij} = \frac{Z_{ij}}{\left(Z_{ij} + Z_{ji}\right)} \qquad (8.4)$$

Following the data pairing notation introduced previously for the secondary values generated from the incident data at a node, the definitions for the voltages and currents in ***equation 8.1*** may be written as

$$V_j = C'_{ij} V_{ij} + \left(1 - C'_{ij}\right) V_{kj}$$
$$= C'_{ij} \left(V_{ij} - V_{kj}\right) + V_{kj}$$

$$I_k = \frac{L'_{ij} \left(V'_{ij} - V'_{ji}\right)}{Z_{ij}}$$

The general scattering equations may thus be written

$$V^r_{ipj} = C'_{ij} \left(V_{ij} - V_{kj}\right) - L'_{ij} \left(V'_{ij} - V'_{ji}\right) + V_{kj} - V^i_{inj} \qquad (8.5)$$
$$V^r_{inj} = C'_{ij} \left(V_{ij} - V_{kj}\right) + L'_{ij} \left(V'_{ij} - V'_{ji}\right) + V_{kj} - V^i_{ipj}$$

It is noted that the current term I_k may take one of two values dependent upon the ordering of the indices *i* and *j*. It is further noted that for isotropic media

$$L'_{ij} = C'_{ji} = \left(1 - C'_{ij}\right)$$

In this case only three parameters are required to define the node, C'_{xy}, C'_{yz} and C'_{zx}. This reduces both storage requirements and the number of multiplication operations required. If the node is further restricted to modelling isotropic, uniformly graded media ***equation 8.4*** yields

$$C'_{xy} = C'_{yz} = C'_{zx} = C'$$

Thus as $L' = (1 - C')$ only one parameter must be stored to define the whole node.

Trenkic further notes that the number of multiplications performed may be reduced using the relationships

$$I_k^{(ij)} = -I_k^{(ji)}$$
$$L'_{ji} = \left(1 - L'_{ij}\right)$$

where $I_k^{(ij)}$ denotes the positive or negative k directed loop current. These may be substituted in to the scattering equations to yield

$$I_k^{(ji)} Z_{ji} = I_k^{(ij)} Z_{ij} - \left(V'_{ij} - V'_{ji}\right)$$

As the bracketed term has already been produced during the earlier stages of the calculation this substitution reduces the number of multiplications by three at the cost of only three extra additions. However the form of ***equation 8.5*** is preferred for hardware implementation as it produces an equal delay in each branch of the processor.

8.2.4 Comparison of the stub loaded SCN and the SSCN

The series of equations derived for the stub loaded SCN offer a significant advantage over large matrix multiplication for hardware development. However they still require many pipelined levels of arithmetic including many multiplication operations. As demonstrated previously these are complex operations in digital arithmetic and can incur high processing time and resource costs. The large number of levels of arithmetic required will give rise to a large noise value at the output.

It has been shown[3] that the SSCN offers a considerable reduction in software processing requirements over the stub loaded SCN. The preceding sections show that this conclusion also holds true for hardware implementation. The SSCN requires fewer levels of arithmetic and considerably fewer multiplications that the stub loaded SCN. As such it is reasonable to expect an SSCN based processor to be more compact than a stub loaded SCN based processor. It would also be expected to yield a higher throughput due to the reduced number of multiply operations. The reduction in logic levels also yields a lower additive noise at the output.

8.2.5 Hardware Development

From ***section 8.2.4*** it is clear that the SSCN offers significant advantages over the stub loaded SCN. It is therefore the preferred scheme for the development of a 3D TLM processor for non-uniform media. Unlike the extensions to the TLM method detailed in previous chapters, the SSCN is not developed from the basic shunt node configuration, nor does it present equations of a similar form to the shunt node. However there are certain similarities between the SSCN and previous systems.

- Intermediate values are formed from combinations of the incident impulses at each port.
- Combinations of these values are used to produce the output values for each port via some intermediate stages.
- All ports in the node may be described by equations of the same form.
- The basic operations performed are addition, subtraction and multiplication.

These similarities are sufficient to suggest that a bit serial, pipelined structure similar to that used in previous processors would be suitable for an SSCN processor. The connect operation in the SSCN is identical to that of the SCN with the exception that internodal reflections may occur between regions of differing material parameters (i.e. between regions of varying link line impedance). If the calculation stages required for the internodal reflection are treated separately to the actual transfer of data then the three dimensional connect processor developed in the previous chapter for the SCN may be utilised here for the SSCN. This requires that the scatter processor accept 12 bit serial inputs and produce 12 bit serial outputs to provide compatibility with the existing system.

All forms of the scatter equations presented above use the 12 intermediate values developed in ***section 7.4.1***. These may be formed using 12 full adders/subtractors. A further 9 adders/subtractors are required to produce the multiplicands in the voltage and current terms of ***equation 8.5***. Trenkic has shown that the 3 voltage and 6 current terms may be produced using only 6 multiplications, however in the development of a hardware based solution for the SSCN the form of ***equation 8.5***, which requires 9 multiplications, is preferable. Using this form the number of computational steps required to produce each product is equal, hence synchronisation is preserved. In the reduced scheme, three of the current terms require an extra subtraction stage thus synchronisation is lost. It has been shown in previous chapters that a fast serial parallel multiplier with one operand of length M and the other presented in a bit serial format requires an M bit wide parallel adder, and an M bit wide accumulator register. It is true that the resources required to produce the necessary delays in the reduced

scheme are less than those required for a further three multipliers. However the introduction of the extra multipliers opens up the system to the implementation of anisotropic media where the reduced scheme simplifications are not valid.

As with previous processor designs the granularity of the SSCN processor is well suited to implementation using a Xilinx FPGA. The use of 9 multipliers leads to high resource requirements for the SSCN processor. However this requirement is mitigated through the use of a bit serial architecture.

8.3 Design of an Application Specific System for the SSCN

A block diagram of a single SSCN processing element is shown in ***figure 8.1***.

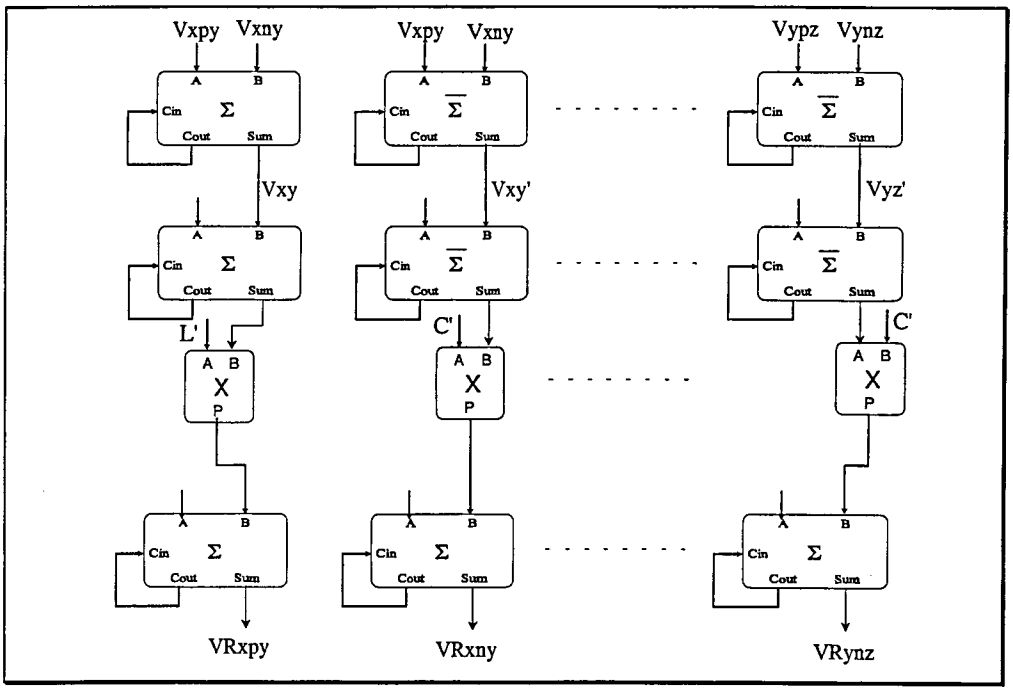


Figure 8.1 - Block Schematic of an SSCN Scatter Processor

This basic processor is suitable for isotropic, uniformly graded media. The bit serial, pipelined architecture is composed of several distinct computational stages. Stage one produces the twelve intermediate values V_{ij} and V'_{ij}. These are combined to form the nine multiplicands for the voltage and current terms in stage two. The multiplicands are passed bit serially to the nine multipliers as they are produced. The double buffered multiplier registers are pre-loaded to reduce latency. In order to reduce

storage requirements only the value of C′ is stored. The value of L′ is found using the relationship

$$L' = (1 - C') = -C'+1$$

The 2s complement of C′ is formed using the circuit developed in ***section 7.4.3***. One is then added to the result bit serially using a half adder with an initial carry input forced high. This computation occurs during the pre-loading stage. The multipliers produce the three voltage and six current terms which are summed in stage four to produce the six combinations required by the next stage of the calculation. A second operation in stage four produces the sum and difference terms which complete ***equation 8.5***. These are formed from delayed copies of the inputs and the necessary secondary values formed in stage one. Finally stage five combines the two sets of outputs from stage four to produce the scattered data at each port. This architecture may be expanded to incorporate anisotropic media by allowing each multiplier register to be loaded individually, thus allowing the material properties to be varied in each dimension independently. Extension of the processor to generally graded media would require the addition of three further multipliers and subsequent adjustment to the logic of stages four and five. This would allow the properties of all link lines to be varied independently, giving rise to a very powerful modelling tool.

8.4 Discussion

As with the stub loaded shunt node a considered choice of numeric representation is required in order to prevent the introduction of errors, particularly during the multiplication operations. As the principle of energy conservation is upheld by the SSCN a fixed point notation may be used to store the incident and scattered data as wholly fractional values. A second format may be required to hold the multiplicands and any intermediate values formed during the calculations. Careful choice of values for the parameters L and C will prevent the need for an intermediate notation.

The significant increase in resources required for the SSCN processor in comparison with the previously developed TLM processors leads to a decrease in the number of processors that may be implemented on a single FPGA. It would be expected that the throughput of an SSCN based system would be less than that of the other systems. This is due mainly to the increased consumption of resources within the FPGA.

It is clear from above that by using bit serial IO the SSCN processor may be used in conjunction with the 3D scatter logic defined in the previous chapter. An additional link line data memory would be required, analogous to the stub memory of the stub loaded shunt node system, to hold the link line parameters. This gives rise to the architecture of ***figure 8.2***. The processor retains the other benefits of a bit serial architecture, e.g. the use of arbitrary word lengths and arbitrary mesh size.

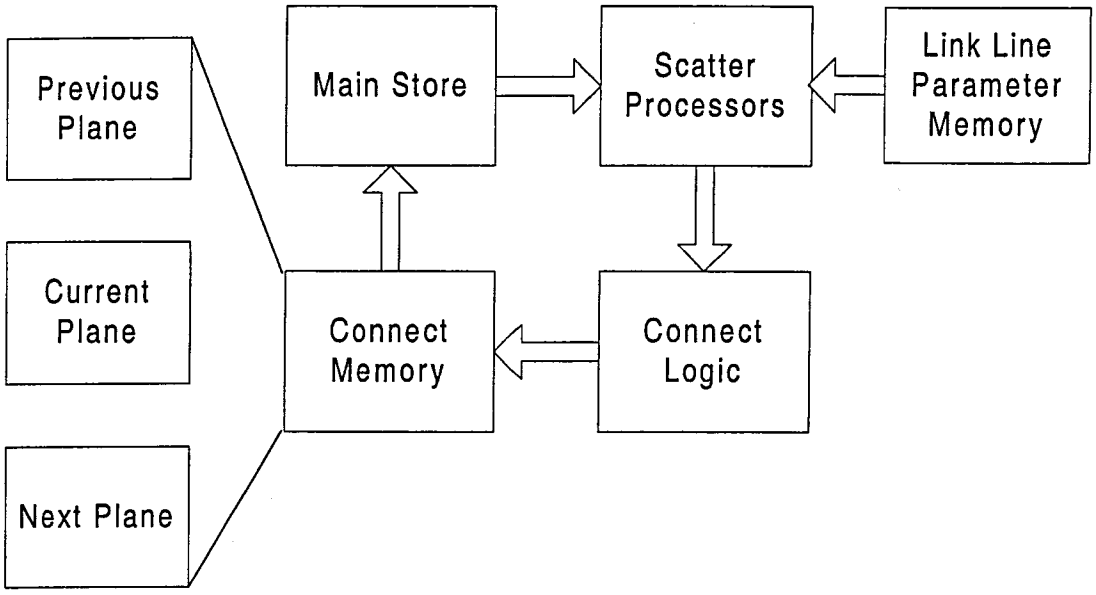


Figure 8.2 - Block Schematic of a Complete SSCN Processor

The introduction of boundaries within the SSCN mesh may be achieved in a manner similar to that used for the SCN. A two bit code is added to the front of each incident data word. This code is used to set the zero and inversion flags which act on the output of the node to set the correct values for the next iteration. As with the SCN additional logic is required to produce an inversion of the output when $\rho = -1$.

8.5 Conclusions

The path between input and output of the SSCN processor includes three full adders and one multiplier. This is the same as for the stub loaded shunt node. Throughput in the SSCN processor has the same dependency upon parameter word length as the stub loaded shunt node. The plot will therefore follow the same trend as ***figure 6.4***. However due to the increased resource requirements it is reasonable to expect that the clock rate would be slightly reduced. This is because the routing within the FPGA becomes congested, forcing the use of less efficient paths. Actual throughput would therefore be slightly lower than for the stub loaded shunt node.

As with previous systems, the efficiency of an array of SSCN processors will be close to 100%. The SSCN processor forms a highly adaptable platform for the study of three dimensional problems in electromagnetic. The processor preserves the full functionality of the SSCN, allowing the modelling of anisotropic, irregularly graded meshes. The combination of the SSCN scatter processor and the 3D connect logic developed in ***chapter 7*** provides a very powerful modelling tool for TLM.

References

[1] Tong, C.E and Fujino, Y 'An Efficient Algorithm for Transmission Line Matrix Analysis of Electromagnetic Problems Using the Symmetrical Condensed Node', IEEE Trans. On Microwave Theory and Techniques, Vol.39(8), pp.1420-1424, 1991

[2] Trenkic, V; Christopoulos, C and Benson, T.M 'New Symmetrical Super-Condensed Node for the TLM Method', Electronics Letters, Vol.30(4), pp.329-30, 1994

[3] Trenkic, V; Christopoulos, C and Benson, T.M 'New Developments in the Numerical Simulation of RF and Microwave Circuits Using the TLM Method', Facta Universatis (Nis), Series Electronics and Energetics, Vol.1, pp.87-95, 1995

9. A Reconfigurable, General Purpose TLM Processor

9.1 Introduction

Chapter 4 of this thesis documented the development of an application specific processor for the two dimensional shunt node. ***Chapters 6, 7 and 8*** subsequently demonstrated the development of scatter processors for three other TLM nodes. These were the stub loaded shunt node, the symmetrical condensed node (SCN) and the symmetrical super condensed node (SSCN). The four scatter processors are based around a common set of features. These are:

- Each processor maps a particular form of the TLM scattering equation to hardware in a manner ensuring that there are no redundant elements.
- The granularity of the proposed implementation technology, the Xilinx XC4000 FPGA, is closely matched to the low level mathematical constructs within the TLM equations.
- The processors all use bit serial arithmetic. There are a number of advantages resulting from the use of a bit serial architecture. They are:
  - Reduced data bandwidth compared to a data parallel processor.
  - Reduced resources required for each processor compared to a data parallel architecture.
  - The word length is independent of the width of the internal logic, thus arbitrary word lengths may be used. This allows the trade off between arithmetic precision and throughput to be evaluated on a per simulation basis.

The TLM connect process has also been mapped to hardware. Connect logic for 2D TLM (***chapter 5***) and 3D TLM (***chapter 7***) has been described. The connect processors also exhibit a set of common features. These are:

- A small number of scatter processors are mapped on to a TLM mesh of arbitrary size.
- The bit serial memory organisation supports the use of arbitrary word lengths.
- The architecture is infinitely scalable for word length, number of scatter processors and mesh size.
- Memory requirements are minimised for a given scheme.
- The connect process adds zero overhead to the computation in the scatter processors.

- Each scatter processor works independently. The throughput of each processor is independent of the size of the scatter array. The efficiency of the array is therefore close to 100%.

Each scatter processor implements only one form of TLM. The distinct, individual processor designs do not achieve the aim of removing limitations on the type of node they can implement. In order to produce a combined scatter processor some redundancy must be introduced. This redundancy must be minimised without affecting the performance of the individual scatter processor implementations. A system capable of implementing the four TLM schemes, preserving the capabilities inherent in each of the existing application specific architectures, would be a very powerful modelling tool. This chapter shows how such a system may be developed. The concept and implications of a general purpose TLM processor are documented and its implementation is discussed.

9.2 Architecture of the General Purpose Processor

A general purpose TLM processor may be realised through one of two techniques, reprogrammability and reconfigurability.

Reprogrammable systems were studied in the literature review of ***chapter 1***. These systems use an instruction stream to control the operations performed on a data stream by the processor. The processor used must be capable of performing a range of tasks. Certain instructions will utilise only certain parts of the processor. The processor must therefore contain some redundant components. The minimum amount of logic the processor can contain is that required to perform the most complex task. Throughput is reduced through the need for instruction fetch and decode cycles. The main advantage of such systems is the ease with which their operation may be changed. To implement a new function the user simply loads a new set of instructions. The level of redundancy required is balanced against the complexity of the instruction set and the range of operations which may be performed. RISC (Reduced Instruction Set Computer) processors use this principle to produce fast, small processors with a limited range of operations, often tailored to a specific task.

Application specific processors represent the logical extreme of the RISC principle. By restricting the processor to a single operation the need for an instruction stream is removed. The processors developed in ***chapters 4, 6, 7*** and ***8*** increase throughput by mapping a single nodal scheme in to hardware. The logic is minimised to perform only

the required scatter operation. Each processor maintains an inherent generality in terms of the use of arbitrary mesh geometries and word lengths. Some basic level of programmability is required to support this flexibility. This takes the form of instructions for loading configuration registers and memory access operations. The logic required to implement this can be kept separate from the scatter and connect logic to prevent it from impacting upon their operation. The general purpose system must incorporate all four types of scatter processor without affecting their individual performance. This is achieved by implementing each processor in its current form using the reconfigurable properties of the FPGA.

Consider ***figure 9.1***. This is the architecture for the SSCN processor developed in ***chapter 8***. Consider also the architectures of ***figure 9.2(a-c)***. These are the architectures for the shunt node, the stub loaded shunt node and the SCN processors respectively. A comparison of these with ***figure 9.1*** indicates that each of these three architectures exists as a subset of the SSCN architecture.

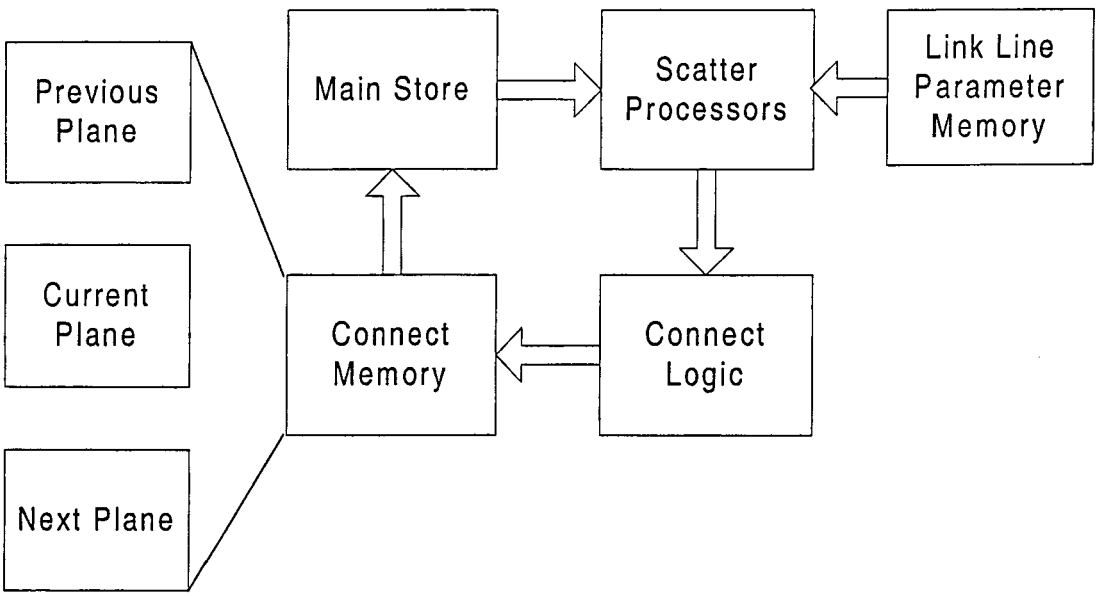


Figure 9.1 - SSCN System Architecture

9.2.1 Implementation of the Reconfigurable TLM Processor

It has been shown that the Xilinx XC4000 family of gate arrays has an internal architecture well matched to the components required to perform TLM calculations. The reconfigurable nature of the FPGA has been used up to this point to allow the development of the shunt node processor. Reconfiguration of the gate array may be utilised to implement a TLM general purpose processor (TLM-GPP). From ***figures 9.1*** and ***9.2*** it is clear that the scatter and connect components are common to all four TLM

schemes. However the contents of these components vary in each case. By implementing the scatter and connect processors using FPGAs the configuration of each may be changed as required. Thus both scatter and connect may be implemented using the optimised logic developed for each scheme. Changing the function of the system becomes a case of loading a new set of bitstreams. This is analogous to loading a new program in to a reprogrammable system and maintains the same inherent simplicity. The system still creates some redundancy. This occurs in the memory requirements, particularly the stub/link line memory. This redundancy does not impact upon the operation of the scatter and connect logic.

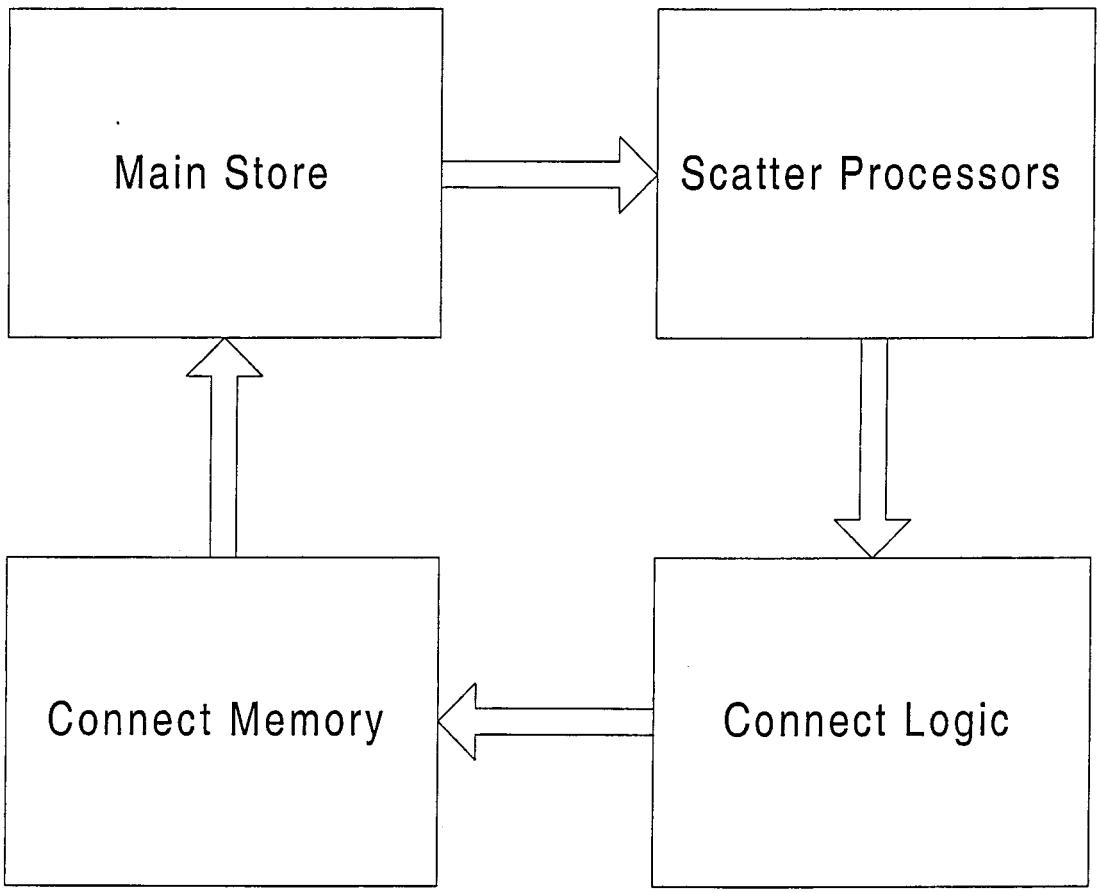


Figure 9.2a - System Architecture for the Shunt Node

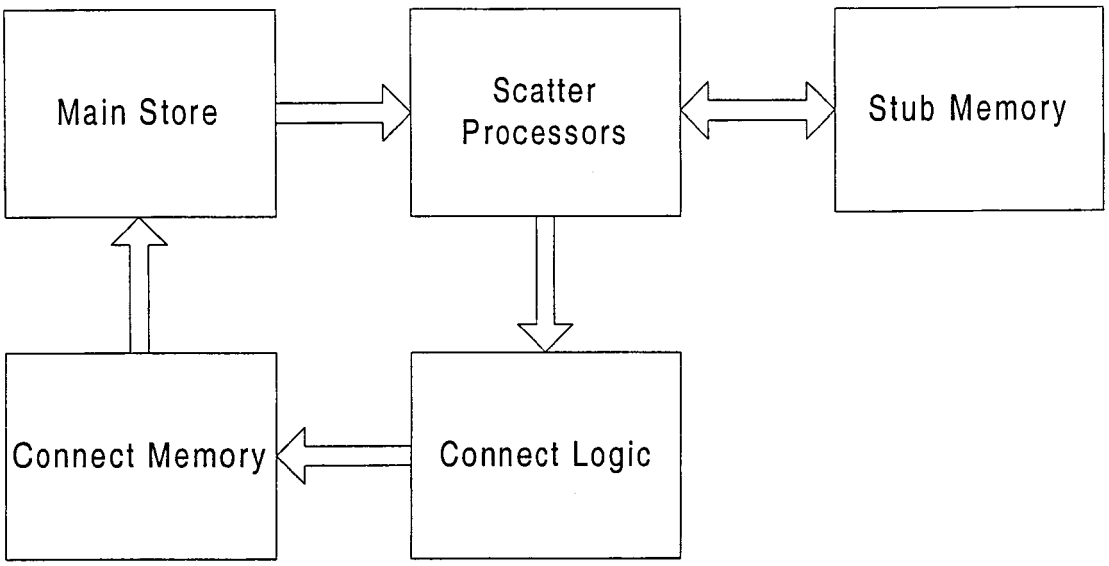


Figure 9.2b System Architecture for the Stub Loaded Shunt Node

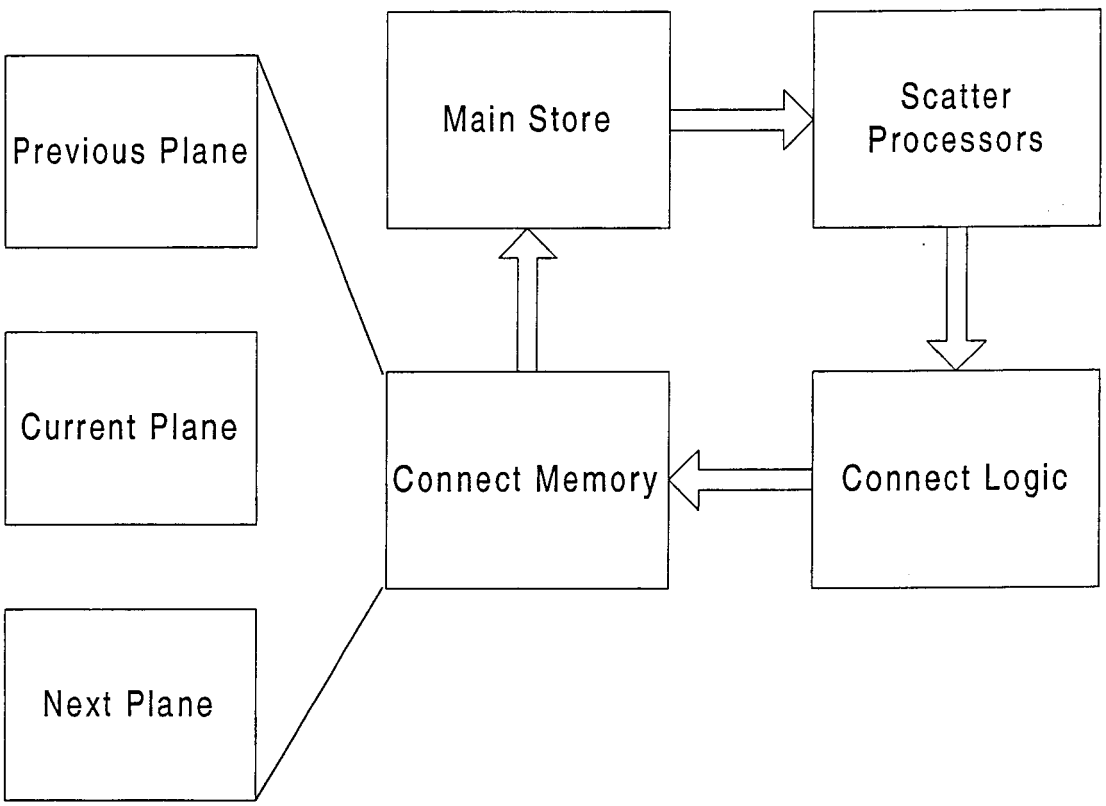


Figure 9.2c System Architecture for the SCN

Each system configuration requires a different combination of control signal timings and configuration options. These may be provided by programmable micro-controller. Another option is to use a control FPGA to provide a reconfigurable control structure. The advantage of this over a micro-controller is that an FPGA contains additional logic resources which may be used for subsidiary functions such as main store read and write

address generation. An FPGA may also be configured to perform multiple control tasks in parallel, an option open to few micro-controllers. The control logic may also store the exponent of a block floating point scheme. This permits the shifting of the main store to improve numeric precision without the intervention of the host, thus reducing data transfer latency.

9.3 Discussion

The reconfigurable TLM-GPP system may perform multiple TLM schemes without the introduction of redundant components or instructions. This is a major advantage over reprogrammable, software based systems. The optimised, hardware mapped scatter processor for a given TLM scheme is implemented unchanged in the TLM-GPP. The considerable flexibility of the new system is gained with no loss of performance. The use of a single architecture for all four schemes removes the need for separate software routines. The processor must be initialised with the data length, the mesh dimensions and the location and type of any boundaries. The timing and control of the processor is then governed by a control FPGA. The correct control bitstream is downloaded for the required node scheme. The operation of each of the four TLM schemes therefore appears identical to the user. Some form of front-end software is required to generate the initial data for the main store. Through careful development of this software the configuration and operation of the TLM-GPP could become totally transparent to the user.

The bit serial architecture common to all four processors permits the use of arbitrary precision arithmetic, independent of the choice of fixed point or integer data. The unique connect hardware developed in ***chapter 5*** for two dimensions and ***chapter 7*** for three dimensions allows a small, fixed number of scatter processors to operate on a rectangular or cuboid mesh of arbitrary[ϖ] size. The connect logic allows the implementation of a block floating point scheme. Numeric representation, word length and arithmetic accuracy can be balanced to achieve the desired performance for a given problem.

The use of FPGAs for the scatter and connect logic provides sufficient resources for expansion of the system. This includes the implementation of new TLM schemes as they arise. Each new scheme can be developed using the procedures followed above and implemented by loading the appropriate bitstreams in to the scatter, connect and control

[ϖ] With the condition that the mesh size in one dimension must be an integer multiple of the number of scatter processors available.

FPGAs. In this way the system is 'future-proof', providing its own minimum effort upgrade path.

The memory requirements of the processor are increased by the extension to three dimensional meshes. The use of a reconfigurable control processor allows extension of the address bus for the main store to effectively allow the use of an arbitrarily large memory. The decoding of the upper address bits to form the memory chip selects is incorporated in to the control device. The bit serial organisation of data within the main store simplifies data addressing.

The stub/link line memory constitutes a redundant component in those schemes that do not require its use. However it is possible to route the address and data lines from these memories to multiple sources, e.g. both the scatter and connect FPGAs, thus when the memory is not required the pins it consumes may be used as general purpose IO.

In developing a hardware implementation of the TLM-GPP certain compromises must be reached.

- The width of the main store will determine the number of scatter processors that may be operated concurrently. The bus widths will in turn be limited by the number of pins available on the FPGAs and the internal resources required for a given number of nodes. Performance must be balanced against resource availability and cost.
- Future expansion of the system will be limited by the configuration of the circuit board on which the FPGAs are placed. While it is possible to reconfigure the computational elements of the design the routing is fixed. Careful choices must be made as to the inclusion of spare IO for future expansion, both how much to provide and where to route between. The provision of spare IO will also reduce the number of pins available for the implementation of existing schemes and will impact upon the performance of the device by limiting the number of scatter processors which may be operated concurrently. Configurable routing using crosspoint switches would increase the flexibility of the board.
- The host interface must be chosen to allow for possible future changes to both the TLM-GPP and the systems to which it will communicate. Thorough research is required to find an interface which provides the required data transfer rates, is widely accessible and will not become obsolete within the predicted lifetime of the TLM-GPP. In recent years many data transfer standards in computing have appeared and not been accepted by the community in general, in particular many high performance standards are not widely used as they offer too great an overhead and are beyond the

account future developments in TLM. Through a careful design of front end software the configuration and operation of the processor become totally transparent to the user, thus making an SSCN mesh as simple to implement as a simple shunt node mesh.

The TLM-GPP represents a new concept in parallel architectures for TLM due to its unique mapping of the TLM connect process in to hardware. This overcomes much of the data transfer latency inherent in other parallel systems due to a mismatch between the requirements of TLM and the provisions of the architecture. As a result of this mapping the connect routine adds zero overhead to the computation. This is a considerable improvement over existing parallel implementations where data transfer is the overriding source of latency within the system. The efficiency of an array of scatter processors for any of the four node TLM schemes considered approaches 100%. The literature review of ***chapter 1*** revealed that most existing large scale parallel computer implementations of TLM exhibited an efficiency of less than 1%. The scalability of the connect logic removes any restrictions upon the size of the TLM mesh. Processing time is a linear function of mesh size and the number of scatter processors. The throughput of individual scatter processors is a function of word length and, where applicable, stub data word length. The overall performance of the system is therefore highly predictable once these parameters are known.

The hardware required by the system is minimal. A typical implementation is composed of 3 FPGAs (scatter, connect and control logic), memory and an interface. Again this represents a significant improvement over existing parallel implementations of TLM where resource requirements range from large scale supercomputers to networks of workstations.

10 Realisation of the TLM Processor as a PCI Card

10.1 Introduction

The development of the TLM processor has concentrated on the elementary design of the TLM processor, focusing on the mapping to hardware of the scatter and connect routines, i.e. the TLM algorithm itself. The feasibility of producing a general-purpose processor for TLM, the TLM-GPP, has been demonstrated. Many of the issues raised in ***chapter 1*** regarding processor design and efficiency have been addressed by the TLM-GPP. Issues of accessibility have not been addressed up to this point. Little consideration has been given to the interface between the TLM processor and the user. Such an interface is necessary to allow the input of model geometry, mesh parameters and excitation and the visualisation of output data. The input and output data is provided in a bit serial format, each byte containing a bit slice from each branch of two nodes. Most host systems store data in a more conventional, word parallel format. At the very least some processing by a host system is required to provide inputs in the correct format to allow the processor to interface to visualisation software or other analysis tools.

10.2 The Host System

The role of the host system is to act as both a controller and a processing platform. The host itself must not violate any of the aims of the TLM processor, it must therefore be cheap, simple to use and conform to a common standard for portability. The host system has two main functions. The first of these is initialisation. This involves programming the FPGAs, defining the mesh and setting the initial state of the main store memory. The FPGAs may be programmed in serial or parallel from the host. The second function is to post process the output data from the mesh. The interface must therefore be bi-directional. Post processing functions will include reformatting data and possibly filtering or Fourier transformation. The host must be capable of performing these functions. The host may also be responsible for controlling the main store as a block floating point array and for injecting impulses in to the mesh at run time.

A number of host configurations are available.

- Stand alone device
- Networked device
- Expansion card

A genuine stand alone system would require all the necessary hardware for pre- and post processing of the input and output data. Some method of visualisation of the output of the models is also required. To include all of this within the TLM-GPP would make the processor very costly, thus limiting acceptability on the grounds of price. It is unlikely that code developed for such a device would be portable to other formats. A second machine acting as a code compiler would be necessary.

By attaching the TLM processor array to a network a host PC or Unix workstation could be used to provide external processing and visualisation. This option would allow multiple users access to a single device. This would minimise cost and increase accessibility. Networks such as Ethernet are supported by a wide range of standard platforms. However the high bandwidth requirements of the TLM processor places restrictions upon the choice of network. Many packet switched networks, e.g. Ethernet, do no provide a guaranteed quality of service (QoS) and can not therefore guarantee the correct or timely delivery of packets of data. To ensure no data was lost would require that considerable control overhead be built in to the TLM processor. The processor would have to be stalled should a data packet be delayed or lost. This would reduce the throughtput of the system. Guaranteed delivery of data under heavy traffic loads, e.g. when the whole mesh is output for visualisation, would require a high speed, dedicated network segment. Networks that do offer a fixed QoS, e.g. ATM, do so at a price. System complexity and component costs tend to be high. These networks often require non-standard interface cards further increasing cost and reducing accessibility.

In fact the TLM processor is small enough to be realised as an expansion card for a host system. The increase in processing power and affordability of the personal computer (PC) in recent years makes it the most widely available platform to host the processor. The main advantage of the PC is that systems developed by many manufacturers share a common architecture and therefore offer a common interface. This standardisation was introduced to simplify the design of both hardware and software for PCs by removing the need to produce separate host adapters for each manufacturer or system. The use of a common interface also simplifies the design of software for the system. Widely available routines and compilers may be used,

allowing designers to develop software for the system without requiring an intimate knowledge of its operation. This maximises code portability. Software written to access the processor from one platform will also successfully access the processor when run from a different platform.

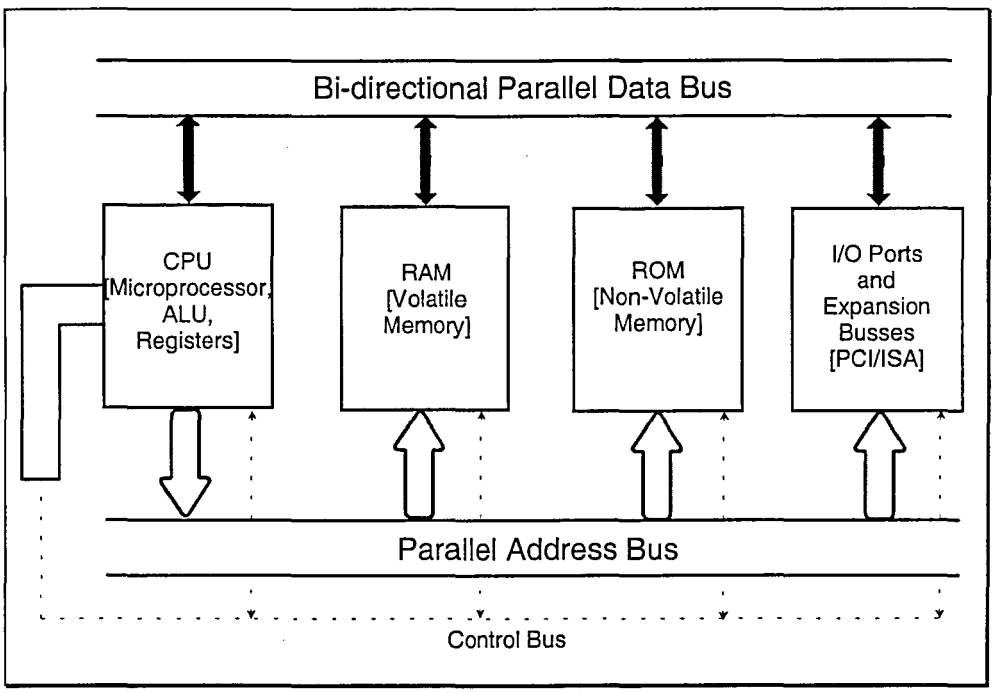


Figure 10.1 - Three Bus Architecture of a Generic PC

The PC has a bus driven architecture, ***figure 10.1***. Adapters and plug-in cards such as graphics and sound cards are connected to the CPU via the expansion buses. There are two standard expansion buses, the older Industry Standard Architecture (ISA) bus and the more recently developed Peripheral Component Interconnect (PCI) bus[1]. Other data transfer protocols may be used to connect an expansion card to a PC, e.g. SCSI or USB. While these may offer higher data transfer rates than either the PCI or ISA expansion buses they are not a standard part of most PC architectures and require an intervening protocol adapter to connect to the PC. The PCI and ISA buses both offer relatively simple interfaces to the PC minimising costs and reducing the scope for design failure. Although the operation of the two systems is similar in principle the protocols used are very different. The high data bandwidth required by the TLM processor is best provided by the PCI bus as it offers a wider data bus and a higher clock rate than the ISA bus.

The development of the TLM-GPP as an add in card for a personal computer would appear to offer the best solution in terms of accessibility. It is a relatively low cost solution offering portability between a number of host platforms. The PC has more than adequate computing power to perform the necessary post processing and

visualisation tasks. The PCI bus would appear to be the most appropriate of the internal PC buses for hosting the TLM-GPP. It offers a wider bandwidth than the ISA bus and represents an emerging standard, thus ensuring survivability. The development of a TLM-GPP architecture for use as a PCI card is detailed below.

10.2.1 The PCI Bus

The PCI local bus was designed to provide a low latency transfer path between high bandwidth peripheral functions and a host processor. The processor is connected to the bus via a *PCI bridge*, through which it may directly access PCI devices mapped anywhere within the processor's memory or I/O address space. The PCI Special Interest Group (PCISIG[Þ]) has been set up by members of the microcomputer industry to control the evolution of the open standard that defines the bus. Devices connected to the PCI bus may take one of two forms, Master or Target (slave). A bus master may take control of the bus and communicate with or control other bus devices where as a target may never read data from or write data to other bus devices, it may only be read from or written to by those devices.

One advantage of the PCI bus is its low pin count, ***figure 10.2***. This is achieved by multiplexing the 32 bit address and data lines on the same pins, **AD[31:00]**. In total only 47 pins are required by a PCI target device and 49 by a master device. The clock signal, **CLK**, runs at 33 MHz, although a compatible 66 MHz specification is under development. A similar expansion to 64 address and data lines is also under development. The current specification offers a data transfer rate of up to 132 Megabytes per second using burst mode transfers.

Any device connected to the PCI bus must adhere to its protocols. To simplify the design of peripheral systems there are a number of PCI interface chips that provide a standardised interface between a local system bus and the PCI bus. Such chips are readily available as interface cards that plug directly in to a free PCI slot on the motherboard of the host PC. The boards generally offer a PCI bridge, a limited amount of on board memory space and either stripboard (or similar) or connectors for a daughter board on which the user's circuit may be developed. These cards form useful prototyping tools as they allow the user to concentrate design effort upon the target system as opposed to the interface. PLX Technologies Ltd has developed one such family of chips and development boards[2]. The PCI9050, ***figure 10.3***, is a target only device.

Þ *PCI Special Interest Group, P.O.Box 14070, Portland, OR 97214*

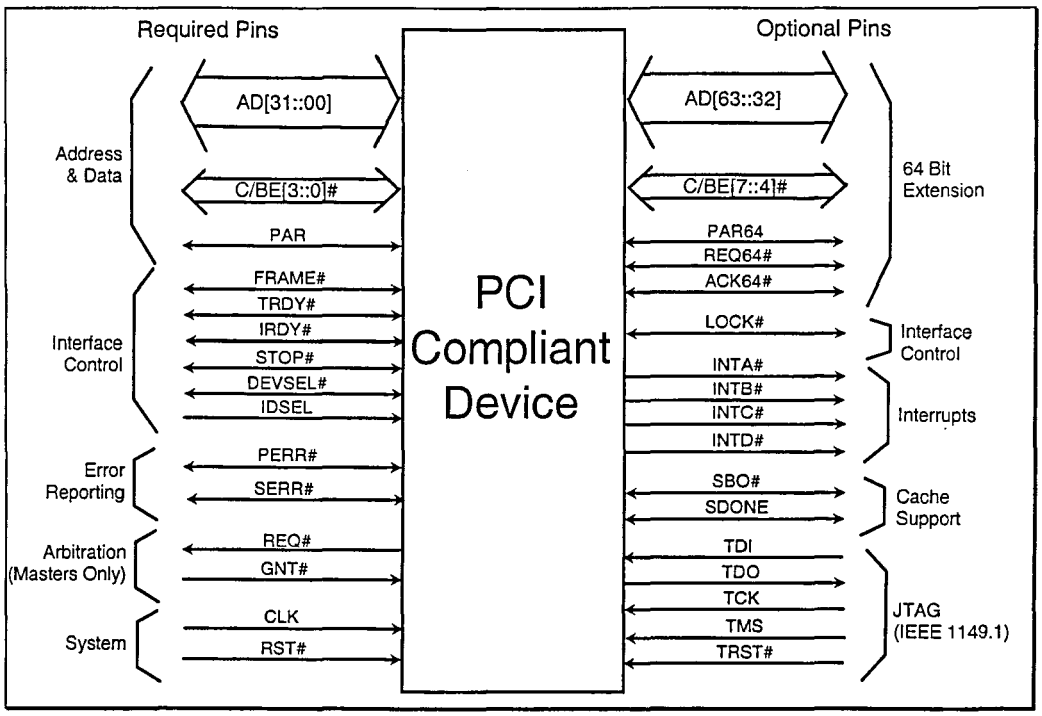


Figure 10.2 - Pin Requirements of the PCI Bus

10.3 A PCI Compliant TLM Processor

The TLM-GPP processor developed in ***chapter 9*** requires some modification to make it compliant with the PCI protocols; a revised architecture is presented in ***figure 10.4***. For clarity the processor is shown in a basic shunt node configuration. Removing the stub and 3D connect memories simplifies the architecture without losing the structure of the system. A PCI9050 interface device provides a bridge between the processor and the PCI bus. The local bus of the PCI9050 development card becomes the main data path of a bus driven architecture for the TLM processor. The main components of the system are the scatter and connect logic and the control and arbitration blocks. Each component occupies a dedicated FPGA. Each 2D scatter processor requires four bit serial data inputs therefore the 32 bit wide local bus is capable of supporting an array of 8 scatter processors i.e. $N^P = 8$. For 3D processors, 12 inputs are required. The use of the 32 bit local bus would permit only 2 processors to operate in parallel. The local bus must therefore be made wider, with the lower 32 bits selected to form the local data bus from the PCI9050. The width of the local bus is dependent upon the width of the main store. A reasonable limit of 128 bits may be set, allowing the parallel operation of 10 3D nodes or 32 2D nodes. Wider buses may be developed at a cost. The physical development of fast synchronous buses becomes more difficult

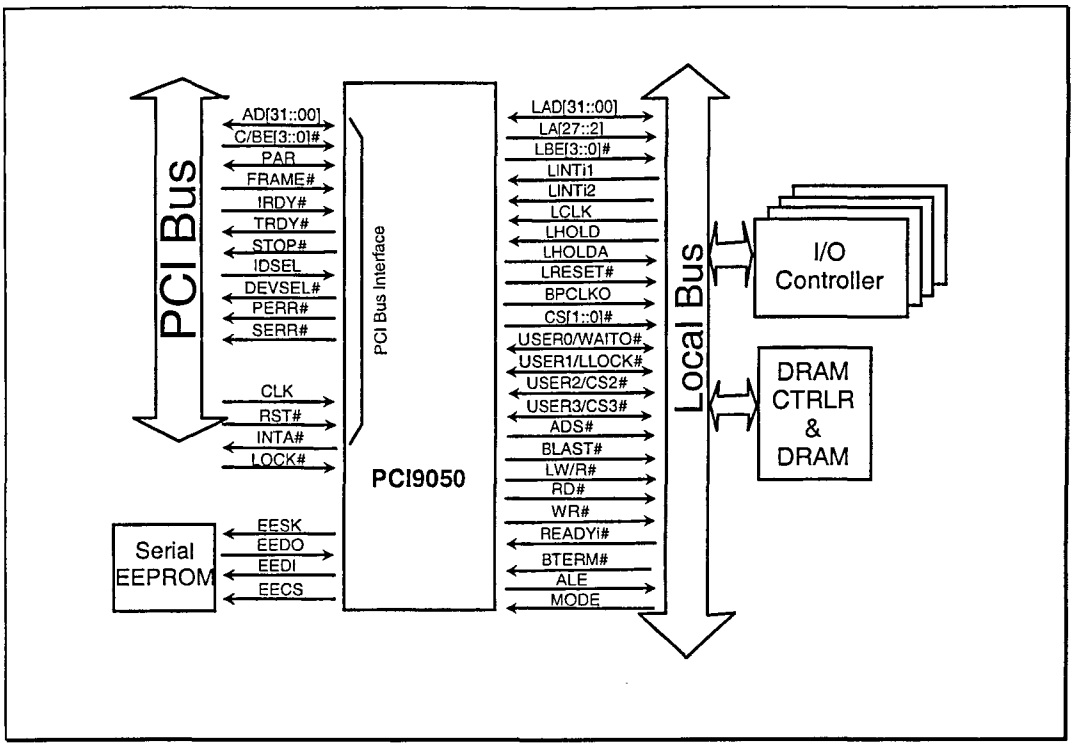


Figure 10.3 - Interface Between the PCI Bus and Local Bus Provided by the PCI9050

as the bus width increases. This is due to the difficulty equalising path lengths and minimising crosstalk effects[3].

The main store is mapped on to a block of SRAM on the board. This is configured to appear as part of the PCI address space of the host, allowing direct access to the main store over the PCI bus using standard memory read and write operations. The PCI memory space is divided in to three sections. The first contains the configuration registers on the connect logic. These are used to set the data word length and store the mesh parameters. The second section is mapped to the main store memory. The final section maps to a small block of SRAM, which may form part of the main store SRAM, used to store the total incident energy at each node. This final section is called the visualisation memory as it holds the data used to visualise the propagation within the mesh.

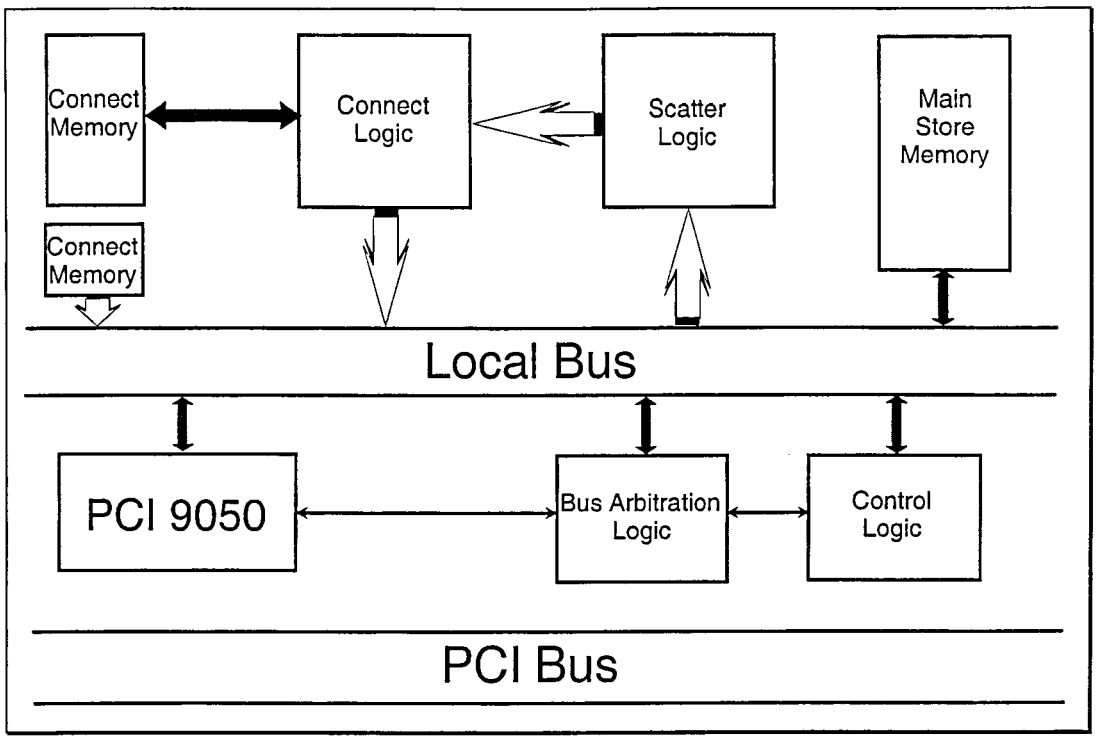


Figure 10.4 - PCI Compliance Requires a Bus Driven Architecture for the TLM Processor

10.3.1 Scatter Logic

The scatter logic is composed of an array of bit serial scatter processors. The number of processors in the array is dependent upon the type of processor. Incident data is input to the scatter processors via the local bus and the scattered data output is routed to the connect logic on a dedicated bus to minimise latency. Total incident energy data is output to the local bus from where it written to the visualisation memory. Only two control lines are required, a clock input to control the flow of data through the processor and a flag to indicate the start of each new data word.

10.3.2 Connect Logic

The connect logic block comprises the logic necessary to route the scattered data to the correct locations within the connect memory. It also co-ordinates the flow of data between the rows of the connect memory and between row 3 and the main store. Boundary data is read from the local bus when the **B_Set** signal is active. Data is received from the scatter processor on each rising edge of the scatter clock and is routed out to the connect memory via three bi-directional buses. Row 3 of the connect memory may be accessed via the local bus. This allows data transfer between the connect logic and the main store. One bit of data is transferred on each rising edge of

the control clock. Data passed to the right of the scatter window is stored internally in a memory generated using the Xilinx internal RAM capability[4]. The memory may be made sufficiently deep to accommodate any given word length. A simple counter suffices as an address generator due to the sequential data storage highlighted in ***section 5.5***. The **CE** signal indicates the start of a new scattered data word. This signal both acts as a reset for the internal memory address generator and gates through the relevant boundary flags for the new scattered data.

10.3.3 Control Logic

The control logic is responsible for the generation of the control and timing signals required by the scatter and connect logic. It also generates the address and control signals for the main store, the connect memory and the visualisation memory. The sequential organisation of data within the memories permits the use of counters as address generators. The counters are returned to zero when they reach a value pre-loaded in to their corresponding control register during initialisation. Each counter has its own register, this is necessary to provide independence between the number of nodes in the array and the aspect ratio of the array. The active low signal **FORCE~** acts as an enable signal for the TLM processor. When **FORCE~** is negated the processor enters configuration mode allowing reading from and writing to the memory mapped configuration registers by the host via the PCI bus.

The amount of data transmitted between the host and the main store may be reduced significantly by using the control logic to provide driving waveforms to the mesh. This is best suited to problems requiring the input of a few repeated or short duration driving signals. The discrete samples for each signal are stored in ROM based look up tables on the control logic FPGA. At the end of each cycle the control logic accesses the main store and adds the stored sample value to the source node. This requires extra, redundant logic in the control processor but is significantly faster than implementing the source on the host system.

10.3.4 Bus Arbitration Logic

The arbitration logic provides all the signals necessary for communication between the TLM processor and the PCI9050 interface. When the processor enters configuration mode the arbitration block takes control of the address and control

signals for the on board memory. This allows direct, random access to the memories from the host instead of via the on board control logic.

10.4 Output Data Post Processing

The TLM processor stores the total incident energy data at each node for each iteration. By mapping the total incident energy memory to the PCI9050 local address space, random access is provided to individual nodes within the array. This allows for the arbitrary placement of output points within the array. Data within the total incident energy memory is organised bit serially. Each output byte placed on **LAD[7:0]** contains one bit from each of the N^P nodes in a given scattering partition. W read operations are therefore required to obtain the data from a single output point for a word length of W. These may be performed as a continuous burst transfer via the PCI9050 read FIFO to minimise performance loss. Careful pre-processing of the array can ensure that local output points exist where possible in the same or adjacent scattering partitions to maximise the use of burst mode transfers and sequential addressing.

Some post-processing of the output data is required to perform a serial to parallel conversion and strip away data from any unwanted nodes. The latter stage is necessary as each read from the total incident energy memory retrieves one bit of data from each of N^P adjacent nodes. If the word length used is not a multiple of 8 then null data must be added to the end of the word to prevent the parallel data lying across byte boundaries. The post processing software may also convert the output data in to a format suitable for a given visualisation programme or prepare data for a Fourier transform for visualisation of results in the frequency domain. Filtering may also be required to reduce the noise present in the output signal. Most modern PCs are capable of performing these functions in real time. It may be preferable to store the output data on disk for processing at a later time.

As the TLM processor requires no external input during the calculation phase the PCI interface is free to transfer output data to the host. Post processing of the output results from one iteration may therefore take place while the next iteration is in progress. This further helps to reduce the latency introduced through post processing. Other tasks requiring the PCI bus may also be performed by the host during this down time.

It is worth noting that the amount of post processing required by the relatively slow access time memory architecture of ***section 5.6*** is considerably reduced due to the parallel organisation of words within the main store. Only a single write instruction is required per output word. This is at the cost of flexibility in defining model parameters.

10.5 Predicted Performance

The performance of the above system is dependent upon the clock rate of the local bus. As the local bus is required for more than one transfer of data within each computational cycle, multiple clock cycles are required to produce each bit of data. In order to synchronise the transfer of data between the host and the TLM processor the PCI 33MHz clock is used as a source for the system clock. It is possible to run the TLM processor at a higher clock rate. However synchronisation with the 33MHz clock must be maintained at the PCI bridge. The use of the 66MHz extended PCI protocol would double the throughput of the system. The scatter processors would still be operating below their predicted maximum clock rate. Using the extended PCI protocol the processor would be capable of approximately 30 million scattering events per second. This is based on boundary equipped shunt nodes using 32 bit data and assumes a local bus width of 64 bits, i.e. $N^P = 16$. Visualisation data is therefore created at a rate of $114MBs^{-1}$. Naturally the throughput for other data lengths or node configurations would differ as discussed in the relevant chapters of this thesis.

As demonstrated in the literature review, latency in data transfer is the governing factor in system performance[5]. Limited transfer rates on the board due to the restrictions on clock rate are further exacerbated by bottlenecks imposed in reading total incident energy data from the board. While the PCI protocol states that data transfer rates of $132MBs^{-1}$ are possible using burst mode these transfer rates are rarely sustainable for any significant period. In a modern PC the PCI bus supports the sound card, network adapter and a number of other expansion cards. Each card competes for use of the bus. Naturally adapters such as the graphics card occupy a considerable portion of the operating time of the bus, thus wait states must be introduced in the transfer of data from the PCI card. As the PCI9050 is a slave device it can not request ownership of the bus and write data to the host system. Rather it must wait until the host initiates a data transfer. The hardware is thus limited by the operating rate of the underlying software and the requirements of the other add in cards.

10.6 Conclusions

A PCI compliant TLM processor has been presented. This encompasses the ideas developed in the preceding chapters within a practical, working system. The TLM mesh is divided in to partitions and all nodes in each partition are processed in parallel. The unique mapping of the TLM connect routine in to hardware developed in ***chapters 5 & 7*** remove the limitation on mesh size imposed by traditional architectures. The large scale parallel computers introduced in ***chapter 1*** are replaced by a single plug in card for a personal computer. The processor achieves many of the aims for this research presented in ***chapter 1***.

- All redundant hardware has been removed from each processor configuration.
- The bit serial architecture and the use of partitioning reduce the required bandwidth to manageable proportions.
- Through reconfiguration any of the four main TLM schemes may be implemented.
- The use of arbitrary word lengths is permitted, allowing dynamic range, accuracy and processing rate to be balanced as required.
- The hardware mapping of the connect process removes limitations on mesh size.
- Boundaries may be distributed arbitrarily within the mesh, allowing the implementation of arbitrary mesh geometry.

Although the processor offers a significant step forward from previous application specific TLM processors it has limitations, notably

- Boundaries are limited to one of three basic types. This is largely a consequence of the use of fixed point data. The use of multipliers to provide a wide range of boundary reflection coefficients must be undertaken only after a careful study of the effect on errors within the mesh.
- Pre-/post-processing of the mesh data is required before it can be used.
- The mesh parameters are limited by the available memory. Memory forms a significant part of the overall cost of the system. The width of the main store determines the maximum number of scatter processors which may operate concurrently. The total size of the main store limits the overall size of the mesh. Suitable provision must be made when routing between memory and the connect logic to allow for extra address bits, paging signals or the implementation of other, more sophisticated techniques to access an extended main store.

- The width of the data buses on the board limit the number of scatter processors that may be supported. A single FPGA may contain many more processors than allowed by the bus width. The number of scatter processors defines the size of each parallel partition of the mesh, therefore limiting the number of processors limits the performance of the system. Careful floorplanning of the board is required to maximise the available bus width without limiting the achievable clock rate.

It is clear that the development of the system as a PCI slave device places constraints upon performance. Future generations of the processor may be developed around other data transfer protocols. However, the concept of developing the processor as a PC expansion card offers the advantages of standardisation and accessibility. While the use of the PCI card does not allow the full potential of the design to be realised it has been clearly demonstrated that an application specific TLM processor is a viable idea and may be realised using relatively inexpensive, accessible components. This is a major advantage over the custom ICs and large scale parallel computers of previous TLM systems. More work is necessary to establish a system architecture and host interface which allow larger partitions and sustained data transfer before the full potential of the processor will be realised.

References

[1] PCI Specification, PCI Special Interest Group

[2] 'PCI Interface and clock Distribution Chips', Product Catalogue, PLX Technology Inc. 1996

[3] Johnson, H and Graham,M 'High Speed Digital Design', Prentice Hall, New Jersey, 1993

[4] 'XC4000 Family Data Book', Xilinx Inc. 1996

[5] Flynn, M.J 'Very High Speed Computing Systems', Proc. IEEE Vol.54(12), pp.1901-1909, 1966

Conclusions

In ***chapter 1*** of this thesis a number of aims were defined for the new class of TLM processor. An implementation path with distinct milestones was also defined. It was determined that the success of this work would be determined by three factors. The first of these was the successful implementation of all the aims, achieved through reaching each of the milestones in the implementation plan. The other factors for success were creating a processor with a high efficiency percentage and producing a significant increase in processing throughput over existing TLM implementations.

It has been demonstrated in the preceding chapters that each of the milestones laid out in the implementation plan has been successfully achieved. This in itself indicates that a working, general purpose TLM processor has been developed. It remains to summarise how the development of the processor addresses each of the original aims.

"The granularity of the TLM algorithm must be successfully mapped to hardware. This means both removing redundant elements from the computational hardware and providing sufficient bandwidth for the connect process."

It was shown in ***chapter 3*** that what appears to be a minimised form of the shunt node scatter equation in a high level representation is actually very inefficient when mapped to hardware. The granularity of the scatter equations was reduced in two ways. Firstly the principle of circuit folding was introduced in ***chapter 4*** to show how a data parallel structure can be efficiently mapped to a bit serial, pipelined structure. This produced a significant reduction in circuit complexity and in bandwidth requirement per clock cycle. Secondly the concept of data pairing was used to highlight repeated structures within the SCN scatter equations. These lower level structures allow the scatter equations to be re-written in a form more acceptable to hardware implementation.

"The chosen architecture should not limit the processor to a single form of the TLM algorithm or a single mesh configuration."

The mesh configuration is varied by the introduction of boundaries and regions of differing material parameters. Both of these may be altered on a node by node basis. The creation of a scatter processor capable of performing a range of simple boundaries generates an homogeneous mesh which may model any arbitrary geometry. By

passing the boundary data to the scatter processors with the input data the geometry may be altered on the fly on a per iteration basis. In the case of the stub loaded shunt node and the SSCN, the material parameters may be altered at each node individually and on a per iteration basis. Mesh configuration therefore has many degrees of freedom.

It has been shown that through reconfigurable computing the TLM-GPP architecture can be used to process a mesh using any one of the four main TLM schemes. This flexibility is provided without the introduction of redundant computational elements and without attenuating the performance of the individual types of processor.

"The chosen architecture should be scalable to allow any mesh size to be implemented."

The mapping of the connect process in to hardware allows a fixed number of scatter processors to process a rectangular/cuboid mesh of arbitrary size. The processing time increases linearly with the number of nodes in the mesh. Each scatter processor in the array operates independently and has no communication with the other scatter processors. The size of the scatter processor array may thus be varied without affecting the throughput of the individual processors. As the connect process is simply a data router and adds no overhead to the computation time the whole processor array is scalable to any given number of scatter processors. This allows throughput to be balanced against resource availability. The only limitation on mesh size is the depth of the main store memory. By using SIMMs, the TLM-GPP allows the size of the main store to be adjusted to meet individual requirements.

"The processor must be accessible. That is its use should not be prohibited through

- Portability
- Cost
- Programming requirements"

By implementing the processor as a PCI card it becomes available to the mass market of personal computer users. This is a significant advance over the large scale parallel computers used in previous applications, where access is generally limited to the research sector. The PCI card is small, light and physically portable. It is also portable in the sense that it may be transferred between computers. The PCI bus is a widely used standard, thus the potential hosts for the card are not limited to a particular manufacturer or class of processor.

The main components of the system are the PCI bridge chip, four Xilinx XC4000 FPGAs and the SIMMs used for the main store. At current prices the board could be produced for considerably less than £1000, even for small volume production. This is well within the range of most potential users, both in industry and research. Using off the shelf components as opposed to a full custom IC development reduces the cost of the system.

The operation of the processor is governed almost entirely by the control FPGA. The host system is required to transfer data to and from the processor and to post process result data. For each of the four classes of TLM node implemented the operation is the same. The host transfers information about the size, shape and material parameters of the mesh to the TLM-GPP. The processor passes scattered data back to the host which has to realign the bit serial data on to a byte organised data bus for visualisation. A simple interface could be developed to handle all of these features and communicate between the host and the TLM-GPP. As reconfiguration is controlled by downloading an appropriate bitstream there would be no requirement for any programming by the user. The same, simple interface could be used to access all four types of node.

The development of the system has highlighted the importance of the connect process to the efficiency of TLM. The TLM-GPP implements the connect routine as a series of multiplexers. These add no communications overhead to the scatter processing time. Because of this each scatter processor in the system may be treated as an independent unit. Each scatter processor operates at its stand alone processing rate. The speed up gained by adding an extra processor is approximately linear with a 1:1 relationship between speed up and number of processors. The efficiency of the system is therefore very close to 100%. The efficiency is slightly reduced by the slight reduction in clock rate inherent when extra resources are consumed within the FPGA. Through careful floorplanning this may be minimised and the efficiency score maintained close to 100%. Given that many large scale parallel computer implementations of TLM demonstrated an efficiency score of less than 1% the new processor represents a significant breakthrough. This is primarily due to the reduction in bandwidth requirements and the unique mapping of the scatter processor array onto the mesh.

The throughput available to a basic 32 bit PCI implementation of the TLM-GPP has been estimated at 30 x 10^6 node iterations per second using a shunt node configuration. This demonstrates a significant throughput increase over software based implementations of TLM at the time of writing. The use of larger FPGAs, allowing large scatter processor arrays, or a faster scatter processor clock would produce a further increase in throughput.

The development of the scatter processors has focused on applications in electromagnetics. This is the dominant area for TLM applications. The 2D scatter processors may be applied to problems in acoustics or other fields. The 3D processors are specific to electromagnetics. Scalar 3D TLM, in which only one link line is required in each direction, has not been studied. The reconfigurable nature of the TLM-GPP allows schemes such as the scalar 3D method to be implemented as suitable scatter and connect processors are developed. The main field of application for the TLM-GPP in its current form remains the modelling of electromagnetic phenomena. The motor industry and the electronics industry remain potential applications of the system. The low cost of the TLM-GPP makes it viable as a research tool or a teaching aid.

One of the initial reasons for developing the TLM-GPP was to overcome the lengthy run times associated with processing many iterations of large models. The TLM-GPP provides the required increase in throughput. The use of integer data and the proliferation of truncation errors and quantisation noise may be expected to make the processor unsuitable for models requiring many iterations. However it was shown in ***chapter 3*** that given a sufficient word length, integer arithmetic results are indistinguishable from those achieved using floating point arithmetic. In certain cases where the word length decreases rapidly, e.g. a mesh excited by a single impulse, a block floating point scheme has been demonstrated in software to provide an accuracy comparable to that of floating point over models in excess of one million node iterations. The TLM-GPP is able to implement a block floating point scheme with very little computational overhead. The use of arbitrary word lengths allows the numeric precision to be balanced against throughput.

From the above it is clear that the TLM-GPP has attained all the goals laid out at the start of the project. The system represents a considerable step forward from existing attempts to accelerate the processing rate of TLM. The power of a large scale parallel

computer has been harnessed within a small, low cost platform. The system is reconfigurable with many degrees of freedom in defining the mesh for implementation. While its main applications lie in the field of electromagnetics, the reconfigurable nature of the TLM-GPP provides an upgrade path for the introduction of many other TLM schemes. The TLM-GPP demonstrates the significant potential of an application specific processor for TLM and provides a platform from which further research may be developed.

Recommendations for Further Study

This thesis has presented the reader with an introduction to the transmission line matrix method for the modelling of electromagnetic wave propagation. A review of techniques for the development of digital arithmetic systems has been presented. The literature review of ***chapter 1*** provides evidence of the need for a more efficient accelerator for TLM. In ***chapter 3*** the efficiency of application specific computing for this purpose was demonstrated. This work forms the basis for the application specific processor designs presented in ***chapters 4-9***. The product of this thesis is an application specific, general purpose TLM processor design. The processor is able to individually implement the four main electromagnetic TLM schemes. These are the shunt node and stub loaded shunt node in two dimensional modelling and the SCN and SSCN for three dimensional problems. In ***chapter 9*** it was shown how a single processor architecture, the TLM-GPP, may be used to implement each of the above schemes without the introduction of any redundant computational logic elements. An example implementation of the processor as a PCI card for use with a personal computer has been described. This example demonstrates the requirements of the interface to the processor.

The use of a bit serial architecture and a unique mapping of the TLM connect process in to hardware produce a system which is capable of operating on a rectangular or cuboid mesh of arbitrary row length and aspect ratio using a small, fixed number of scatter processors. The numerical precision of the calculations is arbitrary and may be selected to fit the requirements of each individual application. Implementing the system as a PCI card provides accessibility, which is significantly absent from traditional high performance computing applications of TLM. These advantages coupled with the high performance of the system fully justify its development. Although the TLM-GPP presents a significant step forward in application specific computing for numerical modelling there are a number of features which must be addressed before the system becomes a truly powerful modelling tool.

Point 1 focuses on the obvious need to build a physical system that may be used to test the TLM-GPP principle. A physical platform would allow the individual node designs to be better evaluated and fully optimised.

Future development of the TLM-GPP may be focussed on two key areas. The high degree of scalability inherent in the architecture of the system is constrained by the physical interface to the host computer. Unless a more suitable interface can be found which maintains the accessibility benefits offered by PCI, other methods of improving the scalability of the system must be found. These issues are addressed in points 2 and 3.

The second area to be addressed is improving boundary and mesh representation. These areas are addressed in points 4 and 5. These points relate to improving both the quality and efficiency of the models produced by the processor.

1 Realisation of a Physical Device

One of the most obvious requirements of any further work is to realise the concepts presented within this thesis as a complete physical system. Using such a system the performance of the processors could be accurately tested against the predictions made in the relevant chapters above. A detailed study of the effect of word length on the accuracy of the processor could also be undertaken. While this is possible through simulation, the large numbers of calculations required to obtain statistically meaningful data are much better performed in hardware.

A physical processor would also provide an ideal platform for prototyping. This would allow the testing of host interfaces, such that alternatives to the PCI bus might be evaluated. Options worthy of study include the Advanced Graphics Port (AGP) and the use of a dedicated Gigabit Ethernet link. It would also allow the rapid testing of new node designs. Possible candidates include diffusion modelling or the development of a floating point processor.

2 Overcoming Memory Limitations

At present the design of the system allows the modelling of meshes containing an arbitrarily large number of nodes. However in practice the number of nodes in the mesh is limited by the word length required and the memory available on the PCI card. A one Mbyte deep main store is sufficient to hold only 8192 nodes using 32 bit data. This represents a mesh of only 90 x 90 nodes. In order to reduce bandwidth, the memory used for the main store must be located on the board with the scatter and

connect processors. One option is to provide a number of free SIMM sockets. The user is then free to customise the system according to requirements and budget by adding or removing memory cards. The control logic must be adaptable to allow sufficient addressing for deeper memories. For very large meshes the main store must be partially stored on the host system and data transferred between the host and the main store as required. Due to the very high bandwidth this would require such a system would suffer from a significant loss of performance. An improved interface to the host would alleviate this problem. The creation of a memory expansion card is another interesting way of extending the applicability of the processor to much larger meshes.

3 Increasing the Level of Parallelisation

The throughput of the system is enhanced in two ways with respect to software implementations of TLM. Firstly the use of application specific, optimised hardware reduces the time required to perform each scattering event and secondly multiple processors are operated concurrently. The throughput of the system is directly proportional to the number of scatter processors operating concurrently. By increasing this number the throughput is raised. The width of the local bus defines how many scatter processors may operate. The bus width is constrained by physical problems e.g. crosstalk. There are several ways of increasing the effective bus width. Firstly the mesh may be partitioned between several boards which are operated concurrently. Data is transferred only during the connect process. As only a single bit serial connection exists in each direction from a scattering node only a two bit wide connection is required between the two boards. The PCI bus protocol allows for proprietary signals between boards. Extra signals would be required for preserving synchronism between the two boards. By generating all control signals on a single board this condition is met. More than two boards may be connected in a daisy chain however the PCI bus will only support a maximum of 4 add in boards in a typical application. Some of these slots may be taken up by graphics cards, sound cards or other peripheral controllers.

The second expansion method is to work on multiple iterations of the mesh in parallel on a single board. When data is written back to the main store from the third row of the connect memory it is no longer influenced by scattering in the current iteration. Instead of writing this data back to the main store it may be passed to a second array of scatter processors that operate on it as the incident data in a second iteration. Any

number of iterations may be processed concurrently; each array of scatter processors requires its own connect memory. Each subsequent iteration lags three rows behind the previous one. The number of iterations must be such that data from the last iteration is written back to the main store behind the current scattering event in the first iteration i.e. it is not possible to operate on the whole mesh concurrently in this way.

4 Improving Boundary Representation

The TLM processor uses a two bit code tagged to the front of each data word to specify the location and type of any local boundaries within the mesh. Due to the use of integer data in the basic shunt node the boundaries which may be represented were chosen as those with reflection coefficients of $\Gamma = 0$, 1 and -1. These boundaries are sufficient for representing simple electric and magnetic walls. However in many practical models, particularly in acoustics, boundaries must represent partially absorptive media and have non-integer reflection coefficients. The processing of such boundary conditions would require the introduction of a multiplier in to the output stream of each branch of the scatter processor. The two bit code would no longer be sufficient to specify the boundary conditions at each branch. Extra storage would be required to hold the reflection coefficient at each branch as this could no longer be specified as part of the data. However if the number of reflection coefficients within the mesh is limited it may be possible to store all values on the scatter processor and specify using a longer code which boundary condition is required at each branch. The length of the boundary code is added to the length of the data when calculating processing time therefore the number of boundaries which could be represented using this system must be kept small to avoid unnecessarily reducing throughput.

It was stated in ***chapter 2*** that there are a number of problems associated with performing multiplication using integer data of a fixed word length. In particular overflow can lead to significant errors. The fixed point scheme used in the stub loaded shunt node and the SSCN may be equally used for the shunt node and the SCN. Assuming the incident data is normalised to ensure that the total energy within the mesh is less than 1 the use of fixed point data would ensure that overflow does not occur. This assumes that only passive boundaries, i.e. those with $|\rho| \leq 1$, are required.

5 Arbitrary Mesh Geometries

At present the system is limited to processing rectangular or cuboid meshes. In order to model a non-rectangular geometry the problem must first be mapped to a rectangular mesh and nodes outside of the problem area processed. This is naturally inefficient. Consider the narrow cross of ***figure 1***.

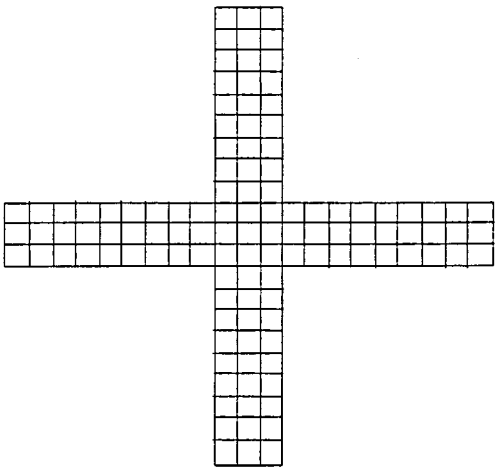


Figure 1 - TLM Mesh Representation of a Narrow Cross

In order to model this geometry a square mesh must be processed. A very high percentage of the nodes in the mesh are outside the problem area but must be processed with null data to preserve the flow of data through the system. This is due to the way in which the mesh is mapped to the system. The connect processor maps to the mesh starting in one corner and traverses each row in turn. It is assumed that each node has a neighbour in all four directions. This is clear if no boundaries are specified at the edges of the model, when the connect processor gives rise to a toroidal mapping of the mesh. In the case of a non-rectangular mesh this is not necessarily the case. The main store may hold data for any mesh geometry, however it is assumed that adjacent nodes in each row occupy adjacent memory locations. Only the connect logic requires any knowledge of the physical geometry of the mesh in order to produce the correct data routing. The smallest geometric feature that may be isolated by the mesh is one row deep and N^P nodes wide for a system of N^P scatter processors. By defining where active nodes exist in the mesh the system may be adapted to operate on arbitrary geometries. This may be performed by setting up a bit map image of the mesh, with one bit representing each 1 x N^P node block. A '1' would represent a block with active nodes where as a '0' would represent an empty cell that may be

passed over. ***Figure 2*** shows the bit map representation of the cross geometry of ***figure 1***.

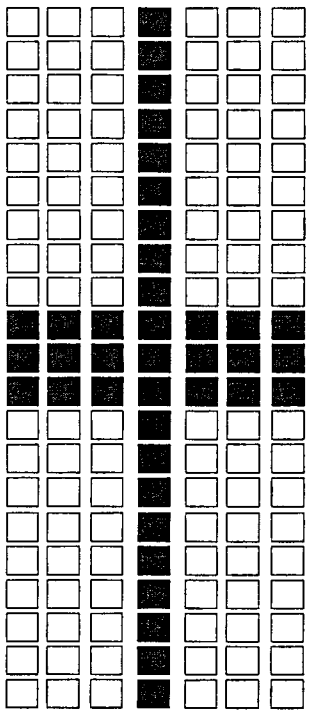


Figure 2 - Bitmap for the Narrow Cross Geometry with $N^P = 3$

The three connect memories must be sufficiently large to store all data for the longest row in the model. Each time the scatter processors are moved the bits corresponding to the current location and the nodes directly in front of it in the bit map image are read. If either is a '1' processing continues as normal, otherwise the address generators are incremented by the word length and processing continues with the next partition. The nodes in front of the scatterers must be checked as these correspond to row 3 of the connect memory. If these nodes are active then they contain data that must be written back to the main store.

In TLM many nodes are initially set with zero data. In a large mesh it may be many iterations before data from the initial impulses reaches these nodes. However in every iteration they are processed with null inputs. This is obviously inefficient. By assigning the mesh bit map values dynamically at run time it is possible to reduce this inefficiency. Initially only those nodes with t=0 inputs are set to '1'. Each time data is scattered in to a new partition its bit map location is also set to '1'. In this way the mesh is dynamically defined at run time such that only those partitions containing real data are active.

6 Conclusions

In conclusion the processors developed in this thesis demonstrate the performance of a large-scale parallel computer in a small, accessible unit. While the above recommendations highlight some of the limitations of the current design the concepts it has introduced lay the foundation for the realisation of a very powerful, generalised class of TLM processor.

The specific issues of low efficiency and poor scalability identified from the literature as affecting previous parallel implementations of TLM have been overcome through a direct mapping of the connect process in to hardware. An efficiency of close to 100% ensures speed up is predictable and the scatter array is scalable without penalty. Scalability in the areas of word length, mesh size and the number of scatter processors ensure that performance can be balanced against throughput requirements to suit individual situations.

Publications

Much of the work presented in this thesis has been publish by the author. Publication details are presented here for further reference.

Stothard,D; Pomeroy,S.C; Sillitoe,I.P.W "An Application Specific Processor for TLM". 1st International Workshop on Transmission line Matrix (TLM) Modelling, 1995

Stothard,D; Pomeroy,S.C "An Application Specific TLM System". TLM : The Wider Applications, 26th June 1996, U.E.A, Norwich

Stothard,D; Pomeroy,S.C "A Parallel Processor for TLM". TLM : The Wider Applications, 24th June 1997, University of Hull, Hull

Stothard,D; Pomeroy,S.C "An Application Specific System for TLM", 2nd International Workshop on Transmission line Matrix (TLM) Modelling, Munich, 28th-31st November 1997

Stothard,D; Melton,M.D; Goodson,D; Pomeroy,S.C; Jaycocks,R "Modelling Near Field Sound Pressure Level Variations in a Shallow Water Marine Mammal Enclosure" Underwater Bio-Sonar Systems and Bioacoustics Symposium, Loughborough University, December 1997

Stothard,D; Pomeroy,S.C "A Dedicated TLM Array Processor", Journal of the Applied Computational Electromagnetics Society, Special Issue on High Performance Computing, Vol.13(2), pp.188-196, 1998

Appendix

- **Derivations of Maxwell's Wave Equation and the Telegraphers' Equation**
- **VHDL Listings**
- **Shunt Node Processor Partitioning Test Results**
- **Schematics for the Shunt Node System**

Derivation of Maxwell's Wave Equation

Maxwell's equations state

$$-\frac{\partial H_x}{\partial x}-\frac{\partial H_z}{\partial z}=\varepsilon\frac{\partial E_y}{\partial t}$$

$$-\frac{\partial E_y}{\partial x}=\mu\frac{\partial H_x}{\partial t}$$

$$-\frac{\partial E_y}{\partial z}=\mu\frac{\partial H_z}{\partial t}$$

These may be combined to yield Maxwell's wave equation by taking the second derivative of the E field in the x and z directions and combining thus

$$-\frac{\partial^2 E_y}{\partial x^2}=\mu\frac{\partial H_x}{\partial t\partial x}$$

$$-\frac{\partial^2 E_y}{\partial z^2}=\mu\frac{\partial H_z}{\partial t\partial z}$$

$$-\mu\frac{\partial H_x}{\partial t\partial x}-\mu\frac{\partial H_z}{\partial t\partial z}=\varepsilon\mu\frac{\partial^2 E_y}{\partial t^2}$$

The last equality is found by comparison with the first of Maxwell's equations. Rearranging the above we arrive at the wave equation.

$$\frac{\partial^2 E_y}{\partial x^2}+\frac{\partial^2 E_y}{\partial z^2}=\varepsilon\mu\frac{\partial^2 E_y}{\partial t^2}$$

Derivation of the Telegraphers' Equation

Consider the transmission line circuit of ***figure 1***.

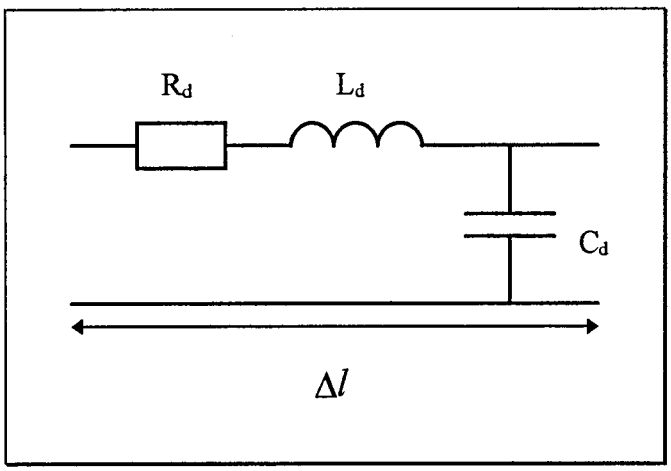


Figure 1 - Equivalent circuit of a length of transmission line

The voltage change across transmission line is the sum of individual voltages across the resistor and the inductor.

$$\frac{\partial v}{\partial x} = IR + L\frac{\partial I}{\partial t}$$

The current flowing through the circuit is given by

$$\frac{\partial I}{\partial x} = C\frac{\partial V}{\partial t}$$

Which may be written as

$$\partial I = C\frac{\partial V}{\partial t}\partial x$$

Substituting this in to our equation for the voltage across the circuit yields

$$\frac{\partial V}{\partial x} = RC\frac{\partial V}{\partial t}\partial x + LC\frac{\partial^2 V}{\partial t^2}\partial x$$

Which may be rearranged to yield the Telegraphers' equation

$$\frac{\partial^2 V}{\partial x^2} = RC\frac{\partial V}{\partial t} + LC\frac{\partial^2 V}{\partial t^2}$$

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY bit_node IS

  PORT (VI1, VI2, VI3, VI4, CLOCK, CS1, CS2, CS3, CE      : IN  std_logic;
        VR1, VR2, VR3, VR4, TOTAL                         : OUT std_logic);

END bit_node;


ARCHITECTURE behav_bit_node OF bit_node IS

COMPONENT a_block
  PORT (A, B, CIN, CS, CLOCK : IN  std_logic;
        COUT, SUM            : OUT std_logic);
END COMPONENT;

COMPONENT b_block
  PORT (A, B, CIN, CS, CLOCK, FLAG, ZERO_FLAG : IN  std_logic;
        COUT, SUM                             : OUT std_logic);
END COMPONENT;

COMPONENT delay
  PORT (I, CLOCK : IN std_logic;
        O        : OUT std_logic);
END COMPONENT;

COMPONENT store
  PORT (I, CLOCK, CE : IN  std_logic;
        O1, O2       : OUT std_logic);
END COMPONENT;

SIGNAL C1, C2, C3, C4, C5, C6, C7, S1, S2, S3, F1, F2, F3, F4,
    ZF1, ZF2, ZF3, ZF4, D1, D2, D3, D4 : std_logic;


BEGIN

U1 : a_block

  PORT MAP (VI1, VI2, C1, CS1, CLOCK, C1, S1);

U2 : a_block

  PORT MAP (VI3, VI4, C2, CS1, CLOCK, C2, S2);

U3 : a_block

  PORT MAP (S1, S2, C3, CS2, CLOCK, C3, S3);

U4 : b_block

  PORT MAP (D1, S3, C4, CS3, CLOCK, F1, ZF1, C4, VR1);

U5 : b_block

  PORT MAP (D2, S3, C5, CS3, CLOCK, F2, ZF2, C5, VR2);
```

```
U6 : b_block
   PORT MAP (D3, S3, C6, CS3, CLOCK, F3, ZF3, C6, VR3);
U7 : b_block
   PORT MAP (D4, S3, C7, CS3, CLOCK, F4, ZF4, C7, VR4);
U8 : delay
   PORT MAP (VI1, CLOCK, D1);
U9 : delay
   PORT MAP (VI2, CLOCK, D2);
U10 : delay
   PORT MAP (VI3, CLOCK, D3);
U11 : delay
   PORT MAP (VI4, CLOCK, D4);
U12 : store
   PORT MAP (VI1, CLOCK, CE, F1, ZF1);
U13 : store
   PORT MAP (VI2, CLOCK, CE, F2, ZF2);
U14 : store
   PORT MAP (VI3, CLOCK, CE, F3, ZF3);
U15 : store
   PORT MAP (VI4, CLOCK, CE, F4, ZF4);
TOTAL <= S3;
END behav_bit_node;
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY a_block IS

   PORT (A, B, CIN, CS, CLOCK : IN  std_logic;
      COUT, SUM          : OUT std_logic);

END a_block;

ARCHITECTURE behav_a_block OF a_block IS

BEGIN

PROCESS(CLOCK)

BEGIN

IF (CLOCK = '1') AND (CLOCK'EVENT) THEN

SUM  <= (A XOR B) XOR CIN;
COUT <= ((A AND B) OR (A AND CIN) OR (B AND CIN)) AND CS;

END IF;

END PROCESS;

END behav_a_block;
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY b_block IS

   PORT (A, B, CIN, CS, CLOCK, FLAG, ZERO_FLAG : IN  std_logic;
      COUT, SUM                        : OUT std_logic);

END b_block;


ARCHITECTURE behav_b_block OF b_block IS

BEGIN

PROCESS(CLOCK)

BEGIN

IF (CLOCK = '1') AND (CLOCK'EVENT) THEN

SUM  <= (((A XOR NOT(FLAG)) XOR (B XOR FLAG)) XOR CIN) AND NOT(ZERO_FLAG);
COUT <= (((A XOR NOT(FLAG)) AND (B XOR FLAG)) OR ((A XOR NOT(FLAG)) AND CIN)
OR ((B XOR FLAG) AND CIN)) OR NOT(CS);

END IF;

END PROCESS;

END behav_b_block;
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY delay IS

   PORT (I, CLOCK : IN  std_logic;
         O        : OUT std_logic);

END delay;


ARCHITECTURE behav_delay OF delay IS

BEGIN

PROCESS(CLOCK)

VARIABLE A, B, C, X : std_logic;

BEGIN

IF (CLOCK = '0') AND (CLOCK'EVENT) THEN

A := I;
B := A;
C := B;
O <= C;

END IF;

END PROCESS;

END behav_delay;
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY store IS

   PORT (I, CLOCK, CE : IN  std_logic;
      O1, O2      : OUT std_logic);

END store;


ARCHITECTURE behav_store OF store IS

BEGIN

PROCESS(CLOCK, CE)

VARIABLE A, B : std_logic;

BEGIN

IF (CLOCK = '1') AND (CLOCK'EVENT) THEN

   IF (CE = '0') THEN

   B  := '0';
   A  := '0';

   END IF;

END IF;

IF (CE = '1') AND (CE'EVENT) THEN

   O1 <= '1';
   O2 <= '0';

END IF;

END PROCESS;

END behav_store;
```

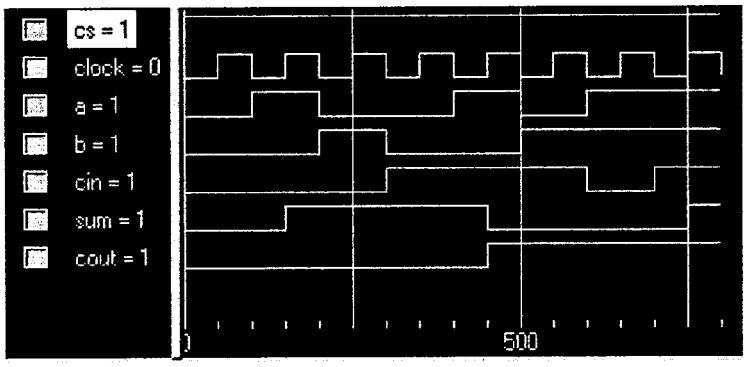


Figure 1 - Full Adder Test Waveforms

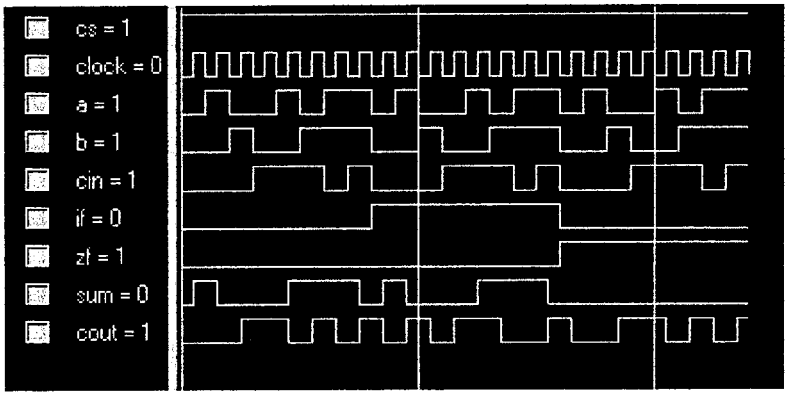


Figure 2 - Subtractor Test Waveforms showing the effect of the boundary flags

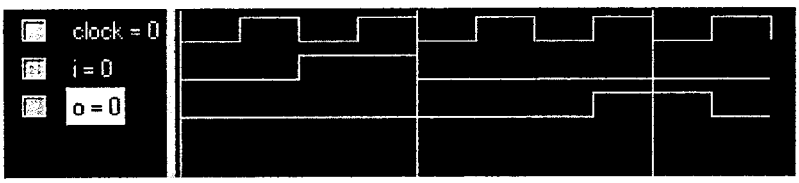


Figure 3 - Delay Chain Test Waveforms

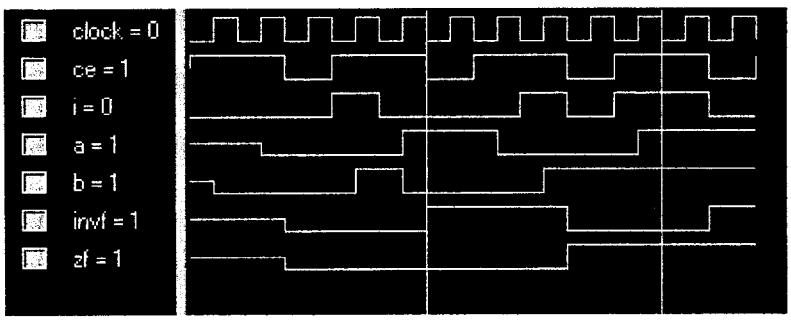
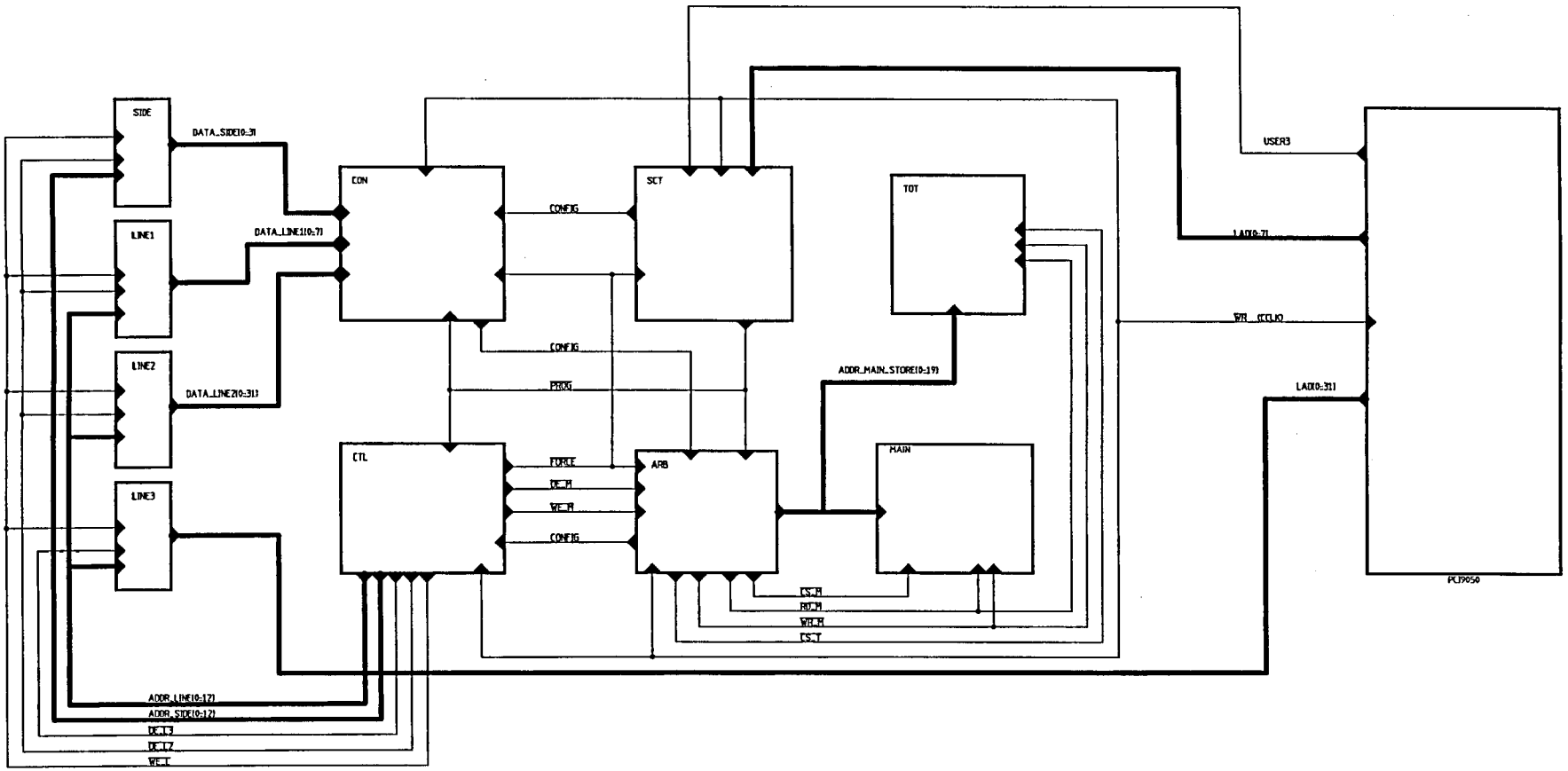
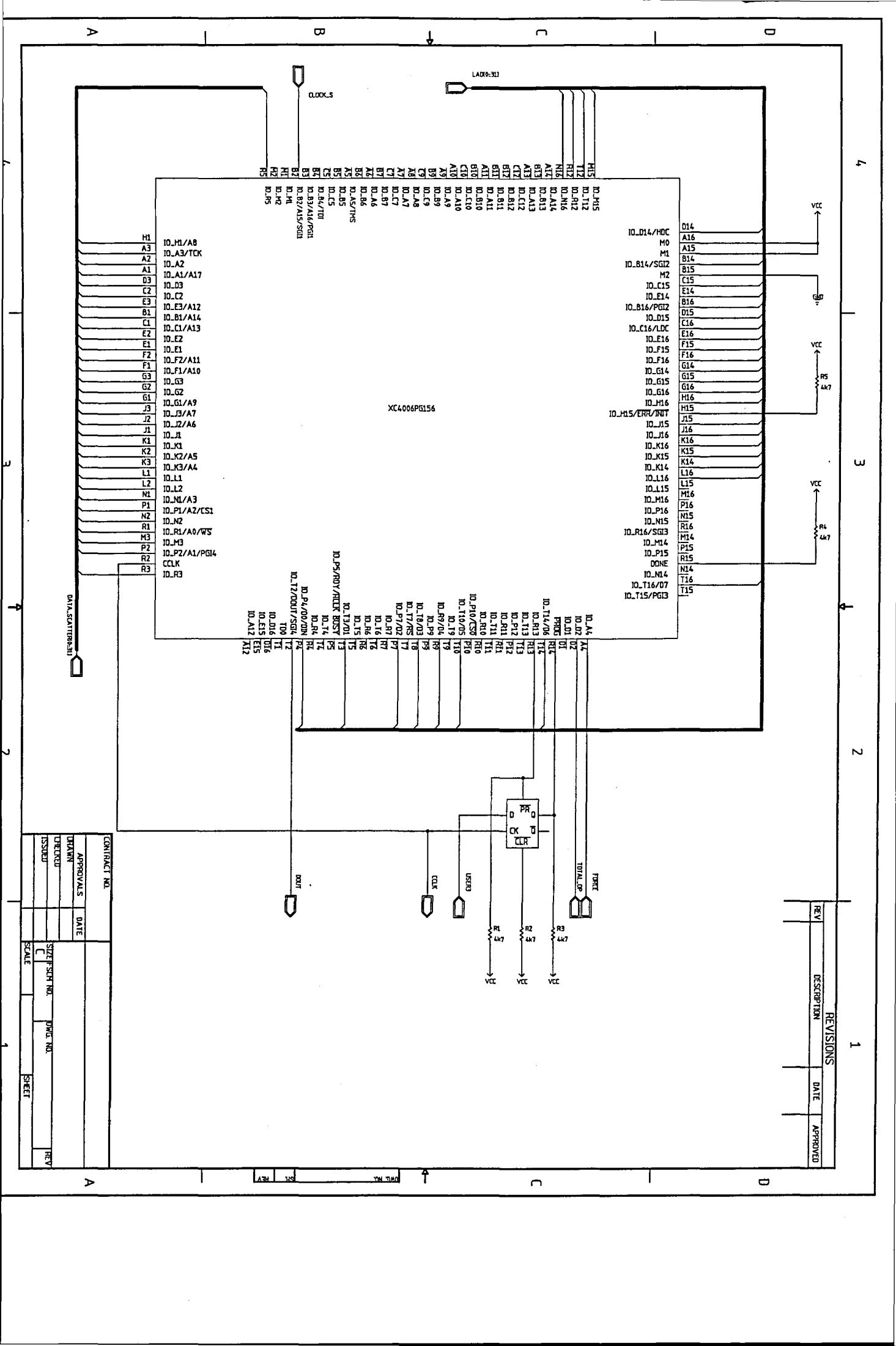


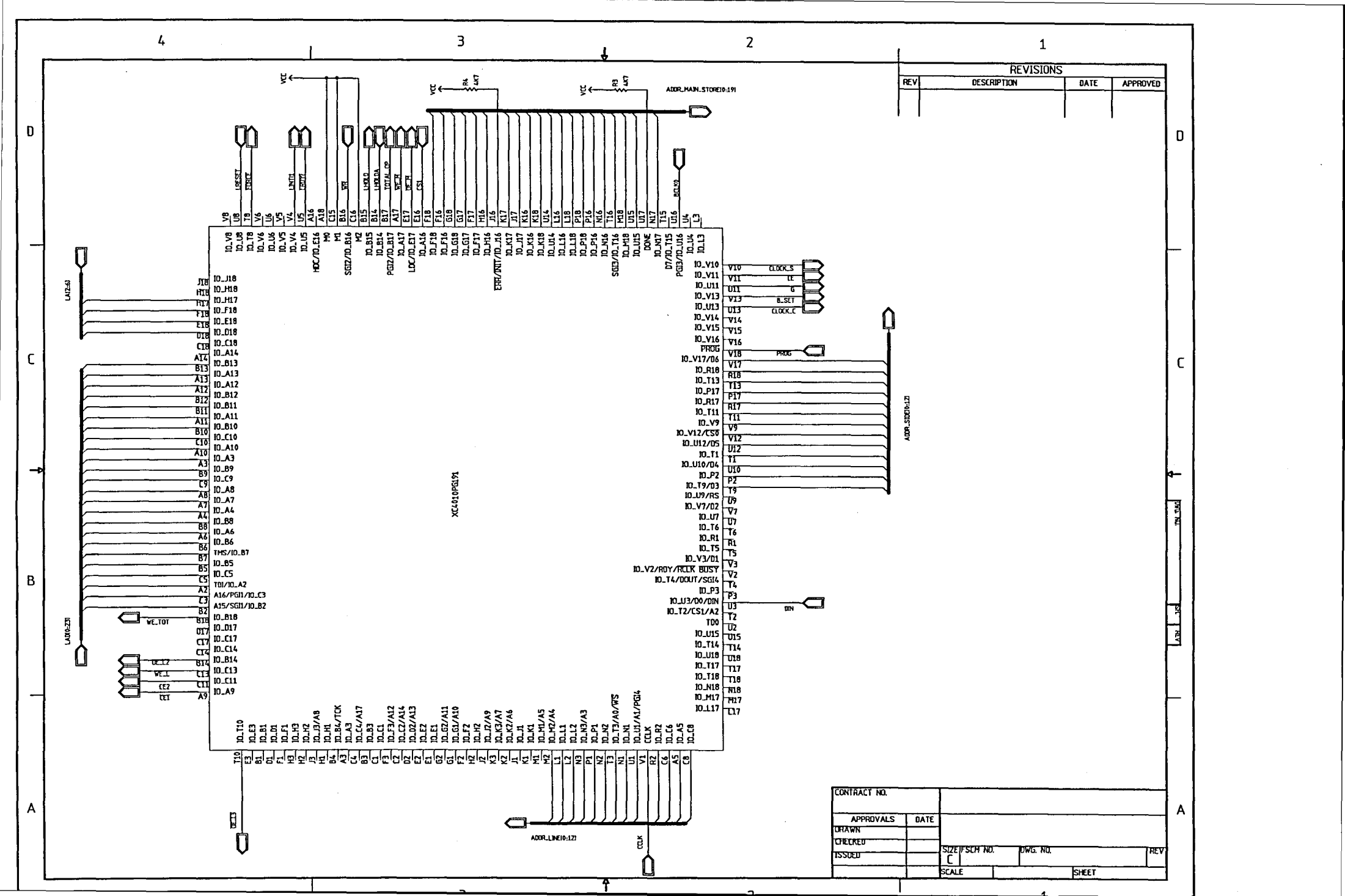
Figure 4 - Boundary Flag Generate/Store Logic Test Waveforms

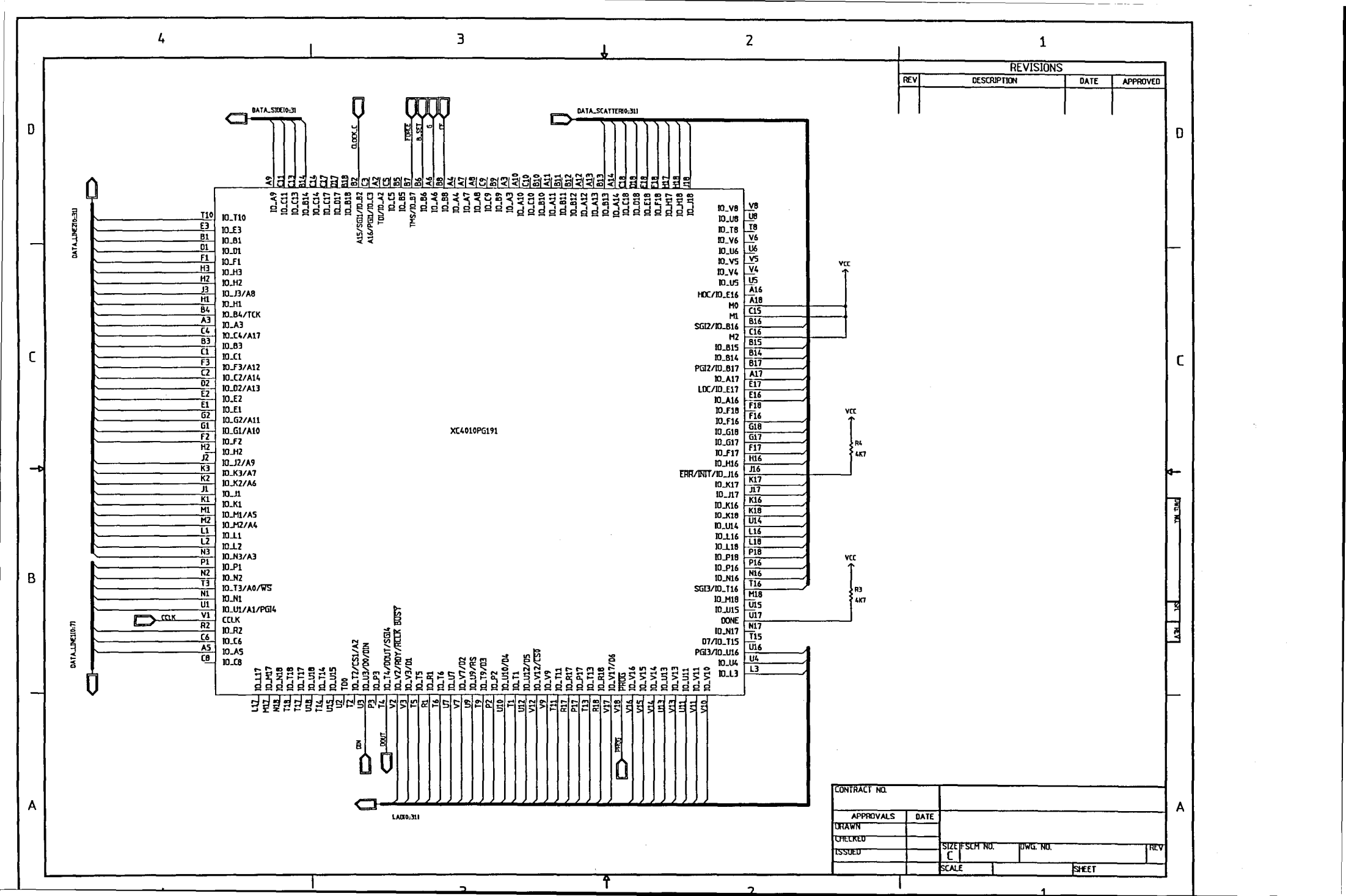
REVISIONS			
REV	DESCRIPTION	DATE	APPROVED

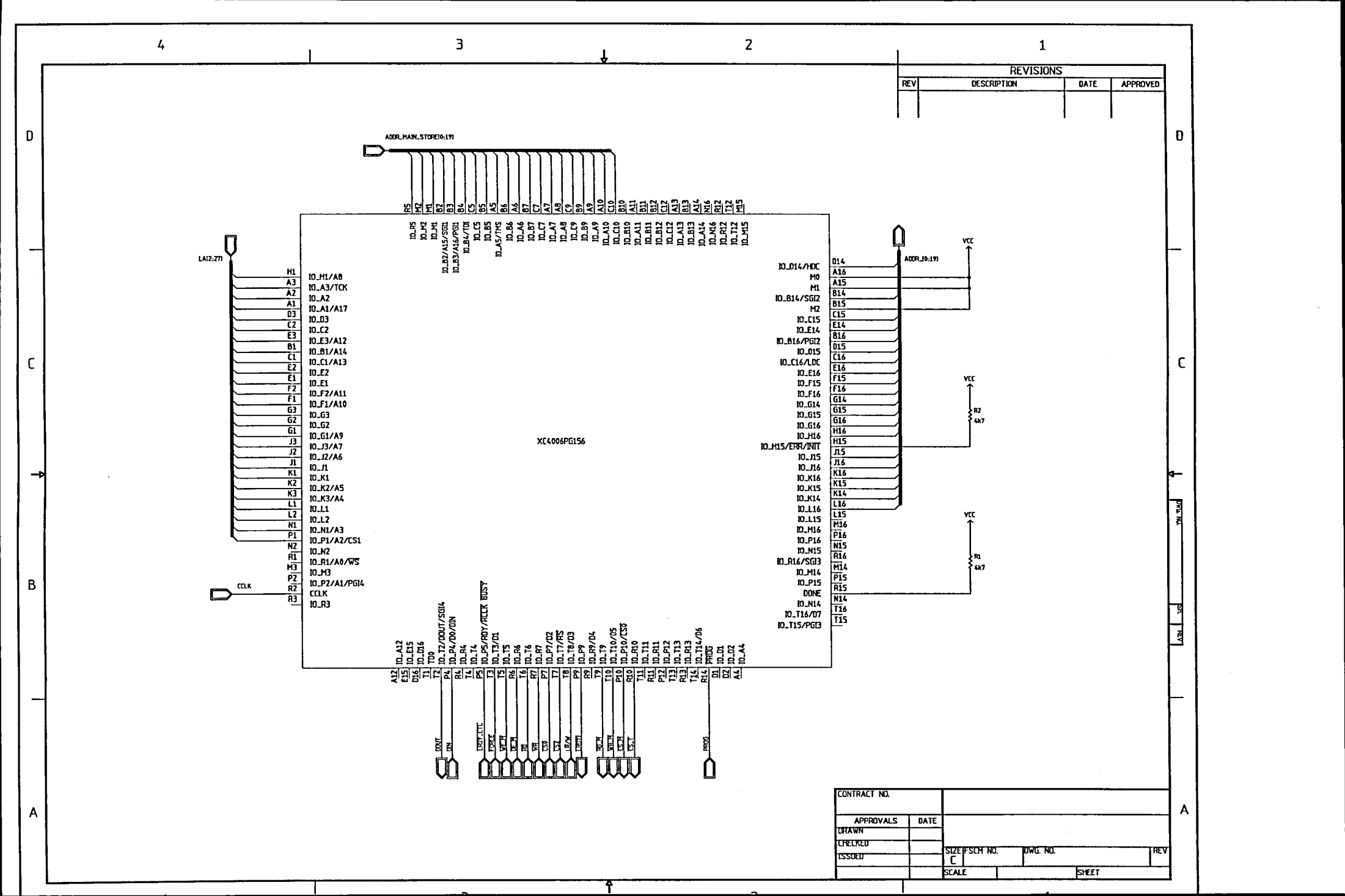


CONTRACT NO.			
APPROVALS	DATE		
DRAWN			
CHECKED		SIZE C	FSCM NO. / DWG. NO. / REV
ISSUED		SCALE	SHEET


XC4006PG156
CLOCK_S
LAD[0:31]
DATA_SCATTER[0:31]
DOUT
CCLK
USER0
TOTAL_OP
FORCE
VCC
GND
R1 4K7
R2 4K7
R3 4K7
R4 4K7
R5 4K7
CONTRACT NO.
APPROVALS
DRAWN
CHECKED
ISSUED
DATE
SCALE
SIZE
C
FSCM NO.
DWG. NO.
SHEET
REV
REVISIONS
DESCRIPTION
DATE
APPROVED


REVISIONS
REV
DESCRIPTION
DATE
APPROVED
XC4010PG191
ADDR_MAIN_STORE[0:19]
ADDR_LINE[0:12]
ADDR_STORE[0:12]
LADR[0:23]
LADR[0:6]
CONTRACT NO.
APPROVALS
DATE
DRAWN
CHECKED
ISSUED
SIZE
FSCM NO.
DWG. NO.
REV
SCALE
SHEET


REVISIONS
REV
DESCRIPTION
DATE
APPROVED
DATA_STOI0:31
CLOCK_C
FORCE
B_SET
G
CE
DATA_SCATTER10:311
DATA_LINE210:311
DATA_LINE110:71
CCLK
XC4010PG191
VCC
R4
4K7
R3
4K7
DONE
PROG
DIN
DOUT
LAD10:311
CONTRACT NO.
APPROVALS
DATE
DRAWN
CHECKED
ISSUED
SIZE
C
FSCM NO.
DWG. NO.
REV
SCALE
SHEET


REVISIONS
REV
DESCRIPTION
DATE
APPROVED
ADDR_MAIN_STORE[0:19]
LAI2::27]
CCLK
XC4006PG156
ADDR_[0:19]
VCC
R2
4k7
R1
4k7
CONTRACT NO.
APPROVALS
DATE
DRAWN
CHECKED
ISSUED
SIZE
C
SCH NO.
DWG. NO.
REV
SCALE
SHEET