Hardware/Software Codesign and Simulation with the Occam Programming Language

Roy L. Krans
Western Michigan University

Follow this and additional works at: https://scholarworks.wmich.edu/masters_theses
Part of the Electrical and Computer Engineering Commons

Recommended Citation
HARDWARE/SOFTWARE CODESIGN AND SIMULATION WITH THE OCCAM PROGRAMMING LANGUAGE

by

Roy L. Krans

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science in Engineering
Department of Electrical and Computer Engineering

Western Michigan University
Kalamazoo, Michigan
April 1995
HARDWARE/SOFTWARE CODESIGN AND SIMULATION WITH THE OCCAM PROGRAMMING LANGUAGE

Roy L. Krans, M.S.
Western Michigan University, 1995

Hardware/software codesign is an increasingly popular method for improving system performance and adaptability while minimizing cost. The goal of this design approach is to achieve the desired system characteristics (performance, fault tolerance, cost, power consumption, etc.) by carefully distributing the system functions among hardware and software components. Popular tools for describing this dual hardware/software functionality are behavioral languages such as C++ and Ada. These languages typically include facilities for both traditional sequential programming as well as behavioral modeling.

This thesis examines the effectiveness of occam as a behavioral language and its role in hardware/software codesign. The occam language is used to model the FPGA based CHS2x4 Custom Computer along with the its configuration and control software. The system model is verified through extensive fine and coarse grain simulations of the occam code, and the results are compared with experimental results from the CHS2x4. Through the verification and simulation results, it is shown that the occam programming language can successfully model both software and system hardware. Alternatively, through the modeling and simulation of an FPGA based system, it is also shown that FPGA systems themselves are well suited for implementing software with hardware.
# TABLE OF CONTENTS

LIST OF TABLES ........................................................................................................... v
LIST OF FIGURES........................................................................................................... vii

CHAPTER

I. INTRODUCTION ........................................................................................................ 1
   Hardware/Software Codesign ........................................................................ 1
   Behavioral Languages ............................................................................... 2
   The Thesis Structure ................................................................................ 2

II. OCCAM AS A BEHAVIORAL LANGUAGE .............................................................. 3
   Parallelism ................................................................................................. 3
   Communication ....................................................................................... 4
   Guards ........................................................................................................ 4
   Priorities ........................................................................................................ 5

III. THE CHS2X4 CUSTOM COMPUTER ................................................................. 7
   PC Interface and Control Subsystem ...................................................... 8
   Memory Subsystem ................................................................................ 10
   Computation Subsystem ...................................................................... 13
   The CAL FPGA Array ........................................................................... 13
   The CAL1024 FPGA Cell ................................................................. 14

IV. CHS2x4 OCCAM MODEL ................................................................................... 18
   Component Modeling Methodologies .................................................. 19
   The Cell Model ....................................................................................... 21
   The Cell Buffer Models ....................................................................... 29
Table of Contents -- Continued

CHAPTER

The Array Model .......................................................... 33
The Memory Model ................................................. 45
The Bus Master Model ............................................. 46
The System Function Models ..................................... 52
An Arbitrary Configuration Model ......................... 54
Model Description Conclusions ............................. 56

V. MODEL VERIFICATION ........................................ 58

Precision Verification ............................................. 58

Cellular Verification Methodology and Results ........ 59
Array Verification Methodology and Results .......... 61

Precision Verification Conclusions ........................ 61

Accuracy Verification ............................................... 63

Propagation Delay Calculation .............................. 63
Computational Delay Calculation ......................... 64

Test Case #1: Seven Segment Decoder .................... 64
Test Case #2: (Massively) Parallel Sorting ................ 67
Test Case #3: Bit Serial Adder ................................. 72

Accuracy Verification Conclusions ........................ 76

Model Verification Conclusions ............................. 77

VI. PERFORMANCE COMPARISONS ......................... 78

Bit Serial Adder Performance Comparisons ............. 78
Massively Parallel Sorter Performance Comparisons ...... 80
Neural Network Example and Performance Comparisons ... 81
# Table of Contents -- Continued

## CHAPTER

Performance Comparisons Summary and Conclusions .......................... 86

VII. CONCLUSIONS ................................................................................... 89

## APPENDICES

A. Occam Model System Constants and Definitions .................................... 92

B. Cell Verification Data ............................................................................... 94

C. Array Verification Data ............................................................................ 102

D. 7 Segment Decoder Linearity Data ........................................................ 106

E. cfg2occ Perl Program ............................................................................... 110

F. CHS2x4 (Massively) Parallel Sorter C Control Code ................................. 114

G. Occam (Massively) Parallel Sorter Control Code ...................................... 118

H. CHS2x4 Bit Serial Adder C Control Code ................................................ 120

I. Occam Bit Serial Adder Control Code ....................................................... 124

J. PC Integer Addition Timing Program ........................................................ 126

K. PC Sorter Timing Program ....................................................................... 128

L. Stochastic Neural Network C Control Code ............................................. 131

M. Stochastic Neural Network Example Output ............................................ 138

N. PC Neural Network Timing Comparison Program .................................... 143

O. Example CLARE .cfg File ........................................................................ 145

## BIBLIOGRAPHY ................................................................................................. 147
LIST OF TABLES

1. CHS2x4 Instructions .......................................................................................... 8
2. CHS2x4 Address Space ..................................................................................... 11
3. Bus Usage Modes .............................................................................................. 11
4. CAL Cell RAM Layout ...................................................................................... 15
5. CAL Cell Functions ........................................................................................... 17
6. Modeled Configuration RAM Layout ............................................................... 29
7. Modeled Bus Modes .......................................................................................... 39
8. Address Decoding for Local Transfers ............................................................... 45
9. Decoder Model Functional Results ..................................................................... 66
10. Decoder Model Linearity Results ...................................................................... 66
11. Parallel Sorter Results ..................................................................................... 70
12. Bit Serial Adder Results .................................................................................. 75
13. Summary of Performance Comparisons Results ................................................ 87
14. Cell Functions 0, 1 and 2 Verification Data ....................................................... 95
15. Cell Functions 3, 4 and 5 Verification Data ....................................................... 96
16. Cell Functions 6, 7 and 8 Verification Data ....................................................... 97
17. Cell Functions 9, 10 and 11 Verification Data ................................................... 98
18. Cell Functions 12, 13 and 14 Verification Data ................................................ 99
19. Cell Functions 15, 16 and 17 Verification Data ................................................ 100
20. Cell Functions 18 and 19 Verification Data ..................................................... 101
21. 3x3 Array Case 1 Routing Data ....................................................................... 103
22. 3x3 Array Case 2 Routing Data ....................................................................... 103
List of Tables -- Continued

23. 3x3 Array Case 3 Routing Data ................................................................. 104
24. 3x3 Array Case 4 Routing Data ................................................................. 105
25. 7 Segment Decoder x1 Clock Data .............................................................. 107
26. 7 Segment Decoder x2 Clock Data .............................................................. 107
27. 7 Segment Decoder x4 Clock Data .............................................................. 107
28. 7 Segment Decoder x8 Clock Data .............................................................. 108
29. 7 Segment Decoder /2 Clock Data .............................................................. 108
30. 7 Segment Decoder /4 Clock Data .............................................................. 109
31. 7 Segment Decoder /8 Clock Data .............................................................. 109
LIST OF FIGURES

1. CHS2x4 Custom Computer ................................................................. 7
2. CHS2x4 Bus Structure ......................................................................... 10
3. CHS2x4 Memory Subsystem ................................................................. 12
4. CAL 2x4 Array ..................................................................................... 14
5. A CAL Cell ............................................................................................ 15
6. Function Multiplexers ........................................................................... 16
7. Routing Multiplexers ............................................................................ 16
8. An Example Configured CAL Cell ......................................................... 17
9. Model Control Flow .............................................................................. 18
10. Model Data Flow .................................................................................. 19
11. Cell Execution Basic Flow Diagram .................................................... 21
12. Cell - Array Communication Channels ............................................... 22
13. Cell - Cell Communication Channels .................................................. 22
14. Cell Input Guards ................................................................................ 22
15. Cell - Cell Buffer ................................................................................ 30
16. Edge Cell or Null Buffer ..................................................................... 31
17. Cell Input Buffer ................................................................................ 31
18. Cell Output Buffer .............................................................................. 32
19. Array Procedure Basic Flow Diagram ............................................... 33
20. Modeled Cell 0,0 ................................................................................ 37
22. Cell Verification Results ..................................................................... 60
List of Figures -- Continued

23. Case 1 Routing Path ........................................................................................ 62
24. Case 2 Routing Path ........................................................................................ 62
25. Case 3 Routing Path ........................................................................................ 62
26. Case 4 Routing Path ........................................................................................ 62
27. Array Verification Results ........................................................................... 63
28. Bit Comparator's Control Signal Logic Diagrams - C and D Outputs .......... 68
29. Bit Comparator's Output Logic Diagram .................................................... 69
30. Bit Comparator's New State Logic Diagram .............................................. 69
31. Parallel Sorter Bit Processing Element ....................................................... 71
32. Parallel Sorter Circuit .................................................................................. 72
33. Bit Serial Adder Circuit ............................................................................... 73
34. Full Adder Schematic .................................................................................. 74
35. Stochastic Neural Network ........................................................................ 82
36. Bit 0 Modulator ......................................................................................... 83
37. Bit 1 Modulator ......................................................................................... 83
38. PRBS Generator ............................................................................................ 84
39. Stochastic Neuron ....................................................................................... 85
40. Stochastic Bit Stream Generator - Stage 1 ............................................... 86
CHAPTER I

INTRODUCTION

This chapter introduces the concept of hardware/software codesign and explains the relationship between behavioral languages and hardware/software codesign. It also establishes a foundation for the overall system modeling methodology employed in this thesis. The means to the model’s further construction and an overview of subsequent chapter contents complete the chapter.

Hardware/Software Codesign

In system development there are three distinct categories of system realization; hardware, software, and a combination of hardware and software. Typical hardware solutions employ ASICs and other forms of dedicated hardware to implement the desired system functions. Software solutions are generally codes derived from algorithms running on a general purpose microprocessor. Likewise, hardware/software codesign employs both of these techniques to arrive at an inexpensive and high performance solution.

The benefits associated with hardware/software systems stem from their integration of the best of both the hardware and software designs. Broadly speaking, these systems acquire speed from hardware side and flexibility from the software. It is then the goal of the system designer is to arrive at an optimal or adequate partitioning of the problem into hardware and software that satisfies the system requirements. One possible method to accomplish this is through the use of behavioral languages.
Behavioral Languages

A behavioral language describes a system's functionality while providing no information on the actual physical system structure. These languages typically resemble conventional sequential programming languages and are occasionally just that. They allow hardware/software systems to be described in one homogeneous structure. Assuming the language provides sufficient facilities, the entire system can be simulated and verified before any hardware/software partitioning has been considered. In the simulation process, modules could be simulated as hardware or, alternatively, they may be simulated as software. In this manner the system's performance with any arbitrary hardware/software implementation can be explored before any physical commitments have been made [Car93].

The Thesis Structure

This thesis develops an occam model that represents the software and hardware of the FPGA based CHS2x4 Custom Computer. The model includes all the components necessary to configure the board and execute a configuration. It is experimentally verified by testing the model components. Each component, ranging from the low level cellular units to the entire occam model, is individually verified using a selection of algorithms. The subsequent simulation results are compared to the experimental results obtained from running the same algorithms on the actual CHS2x4. From these comparisons, determinations are made as to the effectiveness of occam as a modeling tool, the adaptability of the occam model, and the role of behavioral languages in hardware/software codesign.
CHAPTER II

OCCAM AS A BEHAVIORAL LANGUAGE

Communicating sequential processes (CSP) [Hoare78] is a program structuring method that allows for the creation of multiple concurrent processes that can communicate with one another. Since the behavior of any digital system is dictated by the behavior of many interacting components, it follows that a language supporting CSP might be well suited for modeling digital systems. The key ingredients to CSP are parallelism and communication. First, processes or components must be able to operate in parallel as they typically do in hardware. Second, the processes or components that are operating in parallel must be able to communicate. By possessing these two elements, the occam language allows for the creation of communicating processes and therefore the behavioral modeling of digital systems.

Parallelism

The first element, parallelism, is incorporated in occam through the \textit{PAR} construct. This keyword creates a process that concurrently executes the processes within it. The main \textit{PAR} process terminates only when all of subprocesses in it have completed. In the example \textit{PAR} construct shown below, process A executes first, then B and C execute concurrently. When both B and C complete, process D executes.

\begin{verbatim}
A
\textbf{PAR}
B
C
D
\end{verbatim}
Communication

occam also provides for the second element, communication, through for the creation of channels between concurrent processes. Uni-directional channels between two processes can be defined by a simple statement such as: \textit{CHAN OF BYTE Data.bus :.} Two processes can then communicate across the channel \textit{Data.bus} by using two simple commands; ? and !. The command \textit{Data.bus! some.data} sends the byte \textit{some.data} across \textit{Data.bus}. The calling process then waits until another process executes a command similar to \textit{Data.bus? my.data} thereby receiving the transmitted data. There is also a wide range of channel protocols and extensive contention checking by the compiler. Since channels are not shared, the compile time checking is efficient and guarantees low overhead.

Guards

The, \textit{ALT}, provides input guards for channel communication. The purpose of a channel guard is to provide a process with the run time ability to determine if a channel is ready without actually committing to the communication. This is very similar to polling and in some cases one and the same. The basic structure of an \textit{ALT} construct lists any number of channel input statements and their associated code. Each input channel is checked to see if it is ready to transmit. When a channel is ready, the construct receives the data being sent across the channel and performs the code listed for that particular channel.
Priorities

When two or more channels become ready at the same time, one channel is randomly chosen to be executed first. Priorities can be assigned by order with the addition of the PRI keyword in front of the ALT. The construct terminates after any one guard is satisfied. A simple example is shown below:

```
ALT
  Data.bus ? some.data
  SEQ
    x := 5
    z := some.data
  Done ? done.flag
  SEQ
    I.am.done := TRUE
```

This code waits for data across Data.bus or Done. If Data.bus is ready first, x and z are assigned some value and the ALT construct completes. Likewise, if Done is ready first, I.am.done becomes TRUE. Boolean operators can be combined with the channel input statements, e.g. enable & Done ? done.flag, to increase their usefulness. Input guards are also commonly used with a TIMER channel as one of the input channels. TIMERs in occam are always ready and will therefore prevent the code from hanging by permitting the program execution to continue past ALT construct even though no other guards are ready.

A more detailed and concise description of the language can be found in the occam 2 Reference Manual [Inmos88], though the majority of the language's syntax is similar to C and Pascal, and should be familiar to most readers.

As well as possible, occam possesses the necessary facilities for the behavioral style of modeling. Occam also contributes perhaps an unwanted implicit description of certain hardware structures. The most notable of which is the channel communication implementation. All channels in occam are implicitly constructed as
one-way, synchronous, unbuffered communication links between two nodes. This is contrast to true behavioral languages that imply no structure in their descriptions. Because of occam's implied structures, it is more appropriate to consider occam as a hybrid behavioral language that combines features of both structural and behavioral modeling techniques.
CHAPTER III

THE CHS2X4 CUSTOM COMPUTER

The CHS2x4 Custom Computer, Figure 1, is based on the CAL1024 FPGA chip which provides a 32x32 array of reconfigurable cells. The FPGA acronym used here stands for Field Programmable Gate Array and generally refers to a field (static RAM) programmable sea of gates. Based on this definition, each CAL1024 can be considered a single field programmable gate, though each cell does possess additional functionality which will be discussed in later sections. At maximum capacity the CHS2x4 holds 9 CAL1024 chips, 8 of which are readily user programmable and the 9th being used in the PC Interface and Control Subsystem. It can also be equipped with up to 2 megabytes of static RAM. The CHS2x4 is a PC add on card that was designed as a development system for other CAL chips and products. Such as ASIC emulation, programmable systolic machines, or cellular automaton machines. It consists of a three major hardware subsystems: PC Interface and Control Subsystem, Computation Subsystem, and Memory Subsystem.

Figure 1. CHS2x4 Custom Computer.
PC Interface and Control Subsystem

The PC Interface and Control Subsystem provides users with a simple and effective means to execute and configure the 2x4 CAL array. The CHS2x4's control circuitry appears as locations 0x300 through 0x30F in the PC's I/O address space. Access to each of these locations invokes one of the instructions shown in Table 1.

Table 1
CHS2x4 Instructions

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reset</td>
<td>Forcs the array out of run mode and resets a number of internal registers and clocks. This instruction is issued during power on.</td>
</tr>
<tr>
<td>2</td>
<td>Load Address Byte 0</td>
<td>Loads address bits 0:7 into the Control CAL.</td>
</tr>
<tr>
<td>3</td>
<td>Load Address Byte 1</td>
<td>Loads address bits 8:15 into the Control CAL.</td>
</tr>
<tr>
<td>4</td>
<td>Load Address Byte 2</td>
<td>Loads address bits 16:21 into the Control CAL.</td>
</tr>
<tr>
<td>5</td>
<td>Set Base Address</td>
<td>Transfers the 3 address bytes from the Control CAL to the address registers on the address bus.</td>
</tr>
<tr>
<td>6</td>
<td>Toggle Auto-Increment</td>
<td>Enables/disables the automatic incrementing of the address registers after transfer operations.</td>
</tr>
<tr>
<td>7</td>
<td>Toggle Run Mode</td>
<td>Enables/disables run mode operation.</td>
</tr>
<tr>
<td>8</td>
<td>Toggle G1</td>
<td>Toggles the G1 signal to the array.</td>
</tr>
<tr>
<td>9</td>
<td>Toggle G2</td>
<td>Toggles the G2 signal to the array.</td>
</tr>
</tbody>
</table>
The system has 4 buses; one for addresses, two for data, and one for control. The address bus supplies a 22 bit address to access the control CAL, static RAM, or the computational array based on the address divisions shown in Table 2. Two data buses are provided for byte or word transfers from the static RAM or computational array. The control bus determines the configuration of the two data buses and supplies address and data valid strobes. This bus architecture allows the CHS2x4 to

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Read Byte</td>
<td>Reads a byte from the address in the address register. This may be a byte from the RAM or a byte of configuration data depending on the value in the address register.</td>
</tr>
<tr>
<td>11</td>
<td>Write Byte</td>
<td>Writes a byte to the address in the address register. This may write a byte to RAM or a byte of configuration data depending on the value in the address register.</td>
</tr>
<tr>
<td>12</td>
<td>Read Word</td>
<td>Reads a word from the address in the address register. This may be a byte from the RAM or a byte of configuration data depending on the value in the address register.</td>
</tr>
<tr>
<td>13</td>
<td>Write Word</td>
<td>Writes a word to the address in the address register. This may write a word to RAM or a word of configuration data depending on the value in the address register.</td>
</tr>
<tr>
<td>14</td>
<td>Local Transfer</td>
<td>Causes a transfer of 8-bit data from the array to the RAM on the low data bus and transfer from RAM to the array on the high data bus.</td>
</tr>
</tbody>
</table>
be either programmed or running a configuration at any given time. The basic structure of the buses used in programming and execution is shown in Figure 2. The bus configurations listed in Table 3 are dictated by the control bus and depend directly on the instruction issued by the user. Modes 3 through 6 are used either to configure the CAL array or for storing data in the RAM. Mode 2 occurs during 8 bit data transfers between the CAL cells and the RAM. It is normally used with auto-increment to stream data through the computation subsystem.

![Figure 2. CHS2x4 Bus Structure.](image)

Memory Subsystem

The Memory Subsystem is shown in Figure 3. It can contain up to 2 megabytes of SRAM that provide high speed storage for the Computational
Subsystem. The memory is contained in 4 sockets which can hold 128Kx8 or 512Kx8 SRAM chips. The configuration of the memory depends on the mode bits of the control bus. The CHS2x4 procedures allow the user to select three different memory configurations: byte, word, and dual byte.

Table 2

CHS2x4 Address Space

<table>
<thead>
<tr>
<th>Address</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000000</td>
<td>Computation Array</td>
</tr>
<tr>
<td>0x020000</td>
<td>Control CAL</td>
</tr>
<tr>
<td>0x200000</td>
<td>Memory</td>
</tr>
</tbody>
</table>

Table 3

Bus Usage Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reload Control CAL</td>
</tr>
<tr>
<td>1</td>
<td>Write Address Counter</td>
</tr>
<tr>
<td>2</td>
<td>8 bit Write on low data, Read on high data</td>
</tr>
<tr>
<td>3</td>
<td>16 bit Write</td>
</tr>
<tr>
<td>4</td>
<td>16 bit Read</td>
</tr>
<tr>
<td>5</td>
<td>8 bit Write</td>
</tr>
<tr>
<td>6</td>
<td>8 bit Read</td>
</tr>
<tr>
<td>7</td>
<td>No Operation</td>
</tr>
</tbody>
</table>
The byte configuration allows the address space to be accessed as a single array of bytes. Accesses are interleaved across the four sockets and the high and low data buses, i.e. sockets 0 and 2 are accessed using the low data bus while sockets 1 and 3 use the high data bus. The control circuitry handles the bus configuring so the user is unaware of the interleaving. This mode only allows one read or write to occur at one time.

Word transfers are similar to byte transfers except both data buses are used simultaneously to read or write. Each bus still accesses the same memory sockets, but during each operation two sockets are being used, e.g. writing a word to the first SRAM location produces a write to sockets 0 and 1 using both the low and high data buses respectively. As with the byte transfers, only one read or write can occur.

![Diagram of Memory Subsystem](image)

Figure 3. CHS2x4 Memory Subsystem.

In dual transfer mode, the memory sockets are mapped into two blocks of byte data. Sockets 0 and 2, which are accessed using the low data bus, become one block, while sockets 1 and 3 become the other. A Local Transfer instruction causes data to be written on the low data bus to socket 0 or 2, while simultaneously reading data from socket 1 or 3. The memory is two separate blocks that are identically addressed.
During the Local Transfer instruction, a read to any location also causes a write to the same location, but in a different block. In this manner, data can be streamed into and out of the array, supporting pipelined or systolic designs.

Computation Subsystem

The Computation Subsystem contains a 2x4 array of CAL1024 chips, address decoding, and the means to access edge cells. It is a regular structured array of CAL1024 chips that can be configured to provide a wide range of logic or computational structures. Once an entire design has been configured, the array is put in run mode and data in streamed from RAM, through the cells, and back to RAM. The resulting data can then be read from the RAM to PC memory, modified, updated, or whatever the user desires.

The CAL FPGA Array

The array found in the CHS2x4 is a regular sea of gates structure that provides an interface between the cells and the external world. It consists of a 4x2 mesh of CAL1024 FPGA chips shown in Figure 4 that each contain a 32x32 array of CAL cells. The array provides the interface to the address and data buses shown earlier in Figure 2 while monitoring control signals from the CAL control circuitry. This subsystem is controlled by the CAL control circuitry and operates in two modes: "program" and "run".

During program mode, configuration data may be read or written to any cell. The array's decoding circuitry determines a cell's row and column addresses from the current value on the address bus, and the configuration data is then sent or received
(for a read or a write) across the data bus. This mode modifies the CHS2x4's bus configuration and halts any executing configuration, thereby preventing on-the-fly configuring.

The second mode, run, enables the cells to begin executing the configuration loaded into the CAL array. Data values are streamed into and out of the array through the 4 data bus connections, shown in Figure 4, that exist on the array periphery. The low bus connections (LData) output data from the array while the high bus connections (HData) input data to the array. Typically, the Local Transfer instruction initiates this transfer and forces the global signal G1 to be high while the data is valid. G1 is normally used to latch the incoming data values so a stable result can be obtained.

![Figure 4. CAL 2x4 Array.](image)

**The CAL1024 FPGA Cell**

At heart of the CHS2x4's array and computational unit, are the up to 16k of CAL1024 cells shown in Figure 5. These cells perform 1 bit computations and routing based on the state of their configuration RAM. The RAM, shown in Table 4, controls
the select inputs of the multiplexers found in each cell, shown in Figures 6 and 7. This allows each cell to route any incoming signal to any outgoing signal (except back where it came from) and to compute some function of any two incoming signals and/or its current state.

Table 4

<table>
<thead>
<tr>
<th>Column</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 0</td>
<td>West</td>
<td>X2</td>
<td>Y2</td>
<td>Y3</td>
<td>South</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5. A CAL Cell.

Each cell's computational unit's functions, listed in Table 5, are realized through the multiplexers shown in Figure 6. The multiplexers are controlled by the Y1, Y2 and Y3 select bits shown in Table 4. The X1 and X2 variables represent the inputs to the functional unit and can accommodate the North, South, East, or West input channels. In addition to these, X1 can also use two global signals, G1 and G2. The G1 and G2 signals are low skew, global data lines that are intended to be used as clock inputs.
An example of a configure CAL1024 cell is shown in Figure 8. This cell performs an AND of the west and south inputs and returns the result to the east. It also routes the north input to the south output. This occurs simultaneously as each CAL1024 cell's configuration unit can operate independently of the routing and computational units of the cell. This allows good throughput and random access to any of cell configuration state through the control and data channels shown in Figure 5.
When configuration data is transferred between the cells and the array, the control signals indicate a read or write, and the particular memory location being accessed. The address decoding scheme is complex, but in essence selects 4 cells for two bit transfers. This allows for full utilization of byte wide data paths and more efficient data storage. The configuration, routing and functional units constitute the majority of CAL1024 cells.

![Diagram of a CAL cell with North, South, East, and West directions showing an AND symbol.]

Figure 8. An Example Configured CAL Cell.

### Table 5

<table>
<thead>
<tr>
<th>Name</th>
<th>#</th>
<th>Name</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZERO</td>
<td>0</td>
<td>OR</td>
<td>10</td>
</tr>
<tr>
<td>ONE</td>
<td>1</td>
<td>X1BARORX2</td>
<td>11</td>
</tr>
<tr>
<td>X1</td>
<td>2</td>
<td>X1ORX2BAR</td>
<td>12</td>
</tr>
<tr>
<td>X1BAR</td>
<td>3</td>
<td>NAND</td>
<td>13</td>
</tr>
<tr>
<td>X2</td>
<td>4</td>
<td>XOR</td>
<td>14</td>
</tr>
<tr>
<td>X2BAR</td>
<td>5</td>
<td>XNOR</td>
<td>15</td>
</tr>
<tr>
<td>AND</td>
<td>6</td>
<td>DLATCH</td>
<td>16</td>
</tr>
<tr>
<td>X1ANDX2BAR</td>
<td>7</td>
<td>DBARLATCH</td>
<td>17</td>
</tr>
<tr>
<td>X1BARANDX2</td>
<td>8</td>
<td>CBARLATCH</td>
<td>18</td>
</tr>
<tr>
<td>NOR</td>
<td>9</td>
<td>DCBARLATCH</td>
<td>19</td>
</tr>
</tbody>
</table>
CHAPTER IV

CHS2x4 OCCAM MODEL

This chapter develops the occam model for the CHS2x4 hardware described in Chapter III. The first section describes the modeling techniques employed in describing a number of common structures found in the CHS2x4. Following this are sections that describe the development and execution flow of the individual occam procedures that compromise the CHS2x4 model. These procedures combine to produce the control and data flows shown in Figures 9 and 10. Each procedure describes a particular component of the CHS2x4 starting with the more primitive structures, e.g. the CAL1024 cell and cell buffers, and then progressing to the higher level components including the array, bus system, and system calls. The chapter concludes by discussing the problems and successes associated with the development of the occam model.

Figure 9. Model Control Flow.
Component Modeling Methodologies

The first common structure that will be examined here are buses. Generally, bus communication is modeled by declaring a channel of the appropriate type, e.g. an 8-bit data bus would be implemented as a `CHAN OF BYTE my.bus`. Since there is no facility for buses in occam, arrays of individual channels between any two bus users are required.

Next are global signal or clocking lines. These are not explicitly supported in occam and require that a private link be constructed to each node requiring the global signal. This is generally accomplished through structures of the form:

```
PAR i = 0 FOR X
  Global.Channel.To.Node[i] ! global.signal
```
The results of this code are the creation of X uni-directional channels (output channels in this case) that execute in parallel and send the same value to each node.

The static RAM found in the CHS2x4 is modeled using the declaration [RAMSIZE]BYTE RAM which created RAMSIZE bytes of storage which can be addressed through an \( x := \text{RAM}[\text{address}] \) command.

The final structure covered in this section is a component or macro. The instantiation of a component is accomplished through a procedure call that includes the necessary input/output ports as parameters. The following is an example component declaration and instantiation. This component requires two 8-bit input channels and an 8-bit output channel:

```plaintext
-- declaration

    INT x, y :
    BYTE z :

    SEQ
        -- componentX's functional description

    -- instantiation inside another component
PROC ComponentY()

    CHAN OF INT Channel1, Channel2 :
    CHAN OF BYTE Channel3 :

    SEQ
        ComponentX(Channel1, Channel2, Channel3)
```

Once the procedure ComponentX is declared it can be instantiated any number of times in the above manner by simply declaring additional channels and calling the procedure again. Note that in this example the direction (input or output) of the channels has not yet been determined. The use of each channel will determine their direction as input or output with bidirectional usage causing a compiler error.
The Cell Model

Each CAL cell shown earlier in Figure 5 is modeled using the basic flow diagram shown in Figure 11 and the numerous occam listings that follow.

![Flow Diagram]

Figure 11. Cell Execution Basic Flow Diagram.
Upon creation, the cell is placed in a WHILE loop that continues until the cell is terminated by the Array procedure. While in the WHILE loop, the cell monitors the
I/O channels shown in Figures 12 and 13 by creating the six input guards shown in Figure 14. When a channel becomes active, the particular section of code associated with that signal is executed. If two channels are active at the same time, the channel with the higher priority is serviced.

```plaintext
PROC Cell(CHAN OF DISPLAY display, VAL INT myx, myy,
    CHAN OF BOOL Sync,
    CHAN OF BOOL RunMode, CHAN OF BOOL Selected,
    CHAN OF BYTE Data.in, Data.out,
    CHAN OF BYTE Address,
    CHAN OF BOOL Read, N.in, S.in, E.in, W.in,
    CHAN OF BOOL N.out, S.out, E.out, W.out,
    CHAN OF BOOL Gl.channel, G2.channel,
    CHAN OF BOOL Done)
```

```
TIMER clock :
INT time :
INT tmp, delay.index, buffer.index :
BYTE n.select, s.select, e.select, w.select, x1.select, x2.select :
BYTE function, my.new.data, my.address :
[4]BYTE my.config :

[COMPUTE DELAY]BOOL buffer :
BOOL x1, x2, x1.new, x2.new, out :
BOOL n.in, s.in, e.in, w.in, n.out, e.out, w.out :
BOOL cell.state, selected, latch.value :
BOOL done, read :
BOOL Gl, G2, run.mode :
SEQ
    done := FALSE
    run.mode := FALSE
    WHILE (NOT done)

The first three of the input guards, RunMode, Gl.channel, and G2.channel, simply allow the toggling of their global signals while the Done guard signals the termination of the procedure and initiates the termination sequence by setting done to TRUE.
```

```plaintext
PRI ALT
    RunMode ? run.mode
        SKIP
    Gl.channel ? G1
        SKIP
    G2.channel ? G2
        SKIP
    Done ? done
```
The fourth ALT guard is the *Selected* guard. Here, each cell's configuration data is read or written depending on the *Read* channel. The cell's configuration RAM layout, shown in Table 6, differs somewhat from that of the actual CAL cell shown in Table 4. These differences come from a simplification of the CAL layout where VLSI concerns may have been a factor.

```plaintext
Selected ? selected
SEQ
  Read ? read
  Address ? my.address
IF
  read = FALSE
  SEQ
    Data.in ? my.new.data
    tmp := INT my.address
    my.config[tmp] := my.new.data
    tmp := INT my.new.data
    CASE my.address
      BYTE (0)
        SEQ
        n.select := BYTE (tmp \ #07)
        s.select := BYTE ((tmp >> 3) \ #07)
      BYTE (1)
        SEQ
        e.select := BYTE (tmp \ #07)
        w.select := BYTE ((tmp >> 3) \ #07)
      BYTE (2)
        SEQ
        xl.select := BYTE (tmp \ #07)
        x2.select := BYTE ((tmp >> 3) \ #07)
      BYTE (3)
        SEQ
        function := BYTE (tmp \ #1F)
        cell.state := BOOL ((tmp >> 5) \ #01)
    read = TRUE
    SEQ
    tmp := INT my.address
    Data.out = my.config[tmp]
```

The last input guard controls the execution of the cell by monitoring *run.mode*, *selected*, and the cell's x and y coordinates. The use of *run.mode* and *selected* to control cell execution coincides with the CHS2x4's control implementation by allowing execution to only occur in configured cells while *run.mode* is *TRUE*. In addition to this, the *myx* and *myy* registers force cell 0,0 to always execute during run mode whether the cell has been configured or not. This allows the cell to supply the array
with the required synchronization pulses. Since the clock ? time guard is always ready, the execution is solely controlled by these registers (the clock ? time code is included to satisfy occam's requirement for a channel in a guard). If the guard is satisfied, cell execution begins. This guard and the synchronization code is shown below.

\[
\begin{align*}
&\text{(run.mode AND (selected OR ((myx=0)AND(myy=0))))} & \text{\& clock?time} \\
&\text{SEQ} \\
&\text{IF} \\
&\text{(myx = 0) AND (myy = 0)} \\
&\text{PAR} \\
&\text{Sync} \land \text{TRUE} \\
&\text{TRUE} \\
&\text{SKIP} \\
\end{align*}
\]

Following the execution guard, are each cell's two main parallel units: one for routing and another for computation. While the computational unit computes an output value for the current input values, the routing unit updates the X1 and X2 input registers as well as all four output registers. These new values are then transferred to current state registers at the beginning of every execution loop. This is shown below.

\[
\begin{align*}
&\text{PAR} \\
&x1 := x1.new \\
&x2 := x2.new \\
&\text{PAR} \\
&\text{SEQ} \\
&\text{PAR} \\
&N.out \land n.out \\
&S.out \land s.out \\
&E.out \land e.out \\
&W.out \land w.out \\
&\text{PAR} \\
&N.in \land n.in \\
&S.in \land s.in \\
&E.in \land e.in \\
&W.in \land w.in \\
\end{align*}
\]

Here, after sending its current output register values and receiving new input register values, the cell routes the input register values to their appropriate output registers and/or computation registers.

\[
\begin{align*}
&\text{PAR} \\
&\text{CASE n.select} \\
&\text{BYTE (south)}
\end{align*}
\]
n.out := s.in
BYTE (east)
n.out := e.in
BYTE (west)
n.out := w.in
BYTE (f.out)
n.out := cell.state
ELSE
SKIP

CASE s.select
  BYTE (north)
    s.out := n.in
  BYTE (east)
    s.out := e.in
  BYTE (west)
    s.out := w.in
  BYTE (f.out)
    s.out := cell.state
  ELSE
    s.out := cell.state
    SKIP

CASE e.select
  BYTE (north)
    e.out := n.in
  BYTE (south)
    e.out := s.in
  BYTE (west)
    e.out := w.in
  BYTE (f.out)
    e.out := cell.state
  ELSE
    e.out := cell.state
    SKIP

CASE w.select
  BYTE (north)
    w.out := n.in
  BYTE (south)
    w.out := s.in
  BYTE (east)
    w.out := e.in
  BYTE (f.out)
    w.out := cell.state
  ELSE
    w.out := cell.state
    SKIP

CASE xl.select
  BYTE (north)
    xl.new := n.in
  BYTE (south)
    xl.new := s.in
  BYTE (east)
    xl.new := e.in
  BYTE (west)
    xl.new := w.in
  BYTE (gl.val)
    xl.new := Gl
  BYTE (g2.val)
    xl.new := G2
  ELSE
    xl.new := G2
While the routing is taking place, the computational unit calculates the next value of the output register, `out`. The following CASE statement decodes the appropriate function based on Table 5's mapping.

```plaintext
SEQ
CASE function
  BYTE (0)
    out := FALSE
  BYTE (1)
    out := TRUE
  BYTE (2)
    out := x1
  BYTE (3)
    out := NOT x1
  BYTE (4)
    out := x2
  BYTE (5)
    out := NOT x2
  BYTE (6)
    out := x1 AND x2
  BYTE (7)
    out := x1 AND (NOT x2)
  BYTE (8)
    out := (NOT x1) AND x2
  BYTE (9)
    out := (NOT x1) AND (NOT x2)
  BYTE (10)
    out := x1 OR x2
  BYTE (11)
    out := x1 OR (NOT x2)
  BYTE (12)
    out := (NOT x1) OR x2
  BYTE (13)
    out := (x1 AND x2) OR ((NOT x1) AND (NOT x2))
  BYTE (14)
    out := (NOT x1) OR (NOT x2)
  BYTE (15)
    out := (x1 AND (NOT x2)) OR (x2 AND (NOT x1))
  BYTE (16)
SEQ
IF
  x1 = TRUE  -- clock high latch new value
SEQ
```
out := x2
latch.value := x2
TRUE -- clock low output latch value
out := latch.value
BYTE (17) -- D' clk latch
SEQ
IF
  x1 = TRUE
  SEQ
    out := NOT x2
    latch.value := NOT x2
    TRUE
    out := latch.value
BYTE (18) -- D clk' latch
SEQ
  IF
    x1 = FALSE
    SEQ
      out := x2
      latch.value := x2
    TRUE
    out := latch.value
BYTE (19) -- D' clk' latch
SEQ
  IF
    x1 = FALSE
    SEQ
      out := NOT x2
      latch.value := NOT x2
    TRUE
    out := latch.value
ELSE
  SKIP

The final section of cell code implements the delay associated with the CHS2x4's computation unit. The model incorporates a buffer that is the size of the desired computational delay relative to the propagation delay less one, e.g. if the cell to cell propagation delay is 2ns and the computational delay is 10ns, a buffer size of 4 would be required. With this structure, output values are initially taken from the end of the buffer while new values fill the beginning. As the two element counters increment, the output index eventually wraps around to the beginning of the buffer (before the input index does) and begins outputing valid values. The entire process continues to loop and effectively produces a fixed computational delay with no data loss.
buffer.index := delay.index - 1
IF
  buffer.index < 0
  buffer.index := COMPUTE.DELAY - 1
TRUE
  SKIP
  cell.state := buffer[delay.index]
my.config[2] := BYTE (((INT my.config[2]) \ (#7F) \ /
  (INT cell.state) << 7))
buffer[buffer.index] := out
delay.index := delay.index + 1
IF
  delay.index >= COMPUTE.DELAY
  delay.index := 0
TRUE
  SKIP

Table 6

<table>
<thead>
<tr>
<th>Modeled Configuration RAM Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte/Bit</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

The Cell Buffer Models

To construct the large array of parallel cells, a number of simple buffers were created to interconnect the cells. This was based primarily on the desire for a homogenous cell structure and the need for special edge requirements. There are four types of buffers that create connections between adjacent cells, edge cells, input cells, and output cells. Any particular cell will use four buffers whose functions depend on the cell's location within the array.
The cell to cell buffer is described in Figure 15. Each cell to cell connection requires one of these buffers with the array controlling buffer termination. Every communication sequence requires one bit to be sent across In1 or In2 before returning the corresponding data register across Out1 or Out2, respectively. A buffer of this sort was necessary to prevent livelock from occurring when neighboring cells had already been terminated or stopped. This was especially apparent during the toggling of the run.mode register to FALSE. The toggling of the register did not guarantee that all the cells would receive the new signal at the same time, thus some cells would be attempting to transmit data to neighbors that had stopped executing and would livelock.

![Figure 15. Cell - Cell Buffer.](image)

```
-- *******************************************************************************
PROC Buffer(CHAN OF BOOL In1, In2, Out1, Out2, Done)
-- *******************************************************************************

BOOL datal, data2, done :
SEQ
  done := FALSE
  WHILE (NOT done)
    ALT
      In1 ? datal
      Out1 ! data2
      In2 ? data2
      Out2 ! datal
      Done ? done
      SKIP

```

The second buffer, described in Figure 16 and listed below, prevents edge cells from the livelock caused from the lack of a neighboring cell or cells. This buffer is
simply a one-sided version of the cell to cell buffer described earlier. Its purpose is to produce a data source and data sink for all edge cells. Since all cells are homogenous, they are unaware of their location within the array and therefore expect communication with their neighbors.

![Edge Cell or Null Buffer](image)

Figure 16. Edge Cell or Null Buffer.

``` orgasm
PROC CreateNullBuffer(CHAN OF BOOL In1, Out1, Done)

SEQ
  done := FALSE
  WHILE (NOT done)
    ALT
      In1 ? data
      Out1 ! FALSE
      Done ? done
      SKIP
  ;

The final two buffers provide slight modifications to the previous buffer in order to facilitate data I/O (not configuration) between the array and cell. The input buffer, described in Figure 17, allows the array to send a new bit value across DataIn to the buffer's data1 register.

![Cell Input Buffer](image)

Figure 17. Cell Input Buffer.
The output buffer, described in Figure 18, allows the array to read the current value of
the cell's data register. The process is initiated when the array signals across the
Ready line and receives the value across DataOut.

Figure 18. Cell Output Buffer.
The Array Model

The CHS2x4 array code utilizes the flow diagram shown in Figure 19.

Figure 19. Array Procedure Basic Flow Diagram.
The main purpose of the *Array* procedure is to control the creation of each individual cell and its associated buffers while providing an interface between the cells and the external systems.

```plaintext
-- ************************************************************
PROC Array([XMAX][YMAX]CHAN OF DISPLAY disp.channels, 
             CHAN OF DISPLAY my.disp.channel, 
             CHAN OF BYTE High.data.in, 
             CHAN OF BYTE High.data.out, 
             CHAN OF BYTE Low.data.in, 
             CHAN OF BYTE Low.data.out, 
             CHAN OF BYTE Array.bus.mode, 
             CHAN OF INT Array.address )
-- ************************************************************

TIMER clock, clock2 :
INT x, y, i, j, k, l, selx, sely, time, array.address, tmp :
INT count :
CHAN OF INT Sync.value :

BYTE array.bus.mode, low.data, high.data, cell.address :
[XMAX][YMAX]CHAN OF BYTE Data.to.cell, Data.from.cell, Cell.address: 

CHAN OF BOOL g1.changing.to, g2.changing.to, Read.line, Done5 :
CHAN OF BOOL run.mode.changing, SDone, Sync.request, Sync.clear :
[XMAX][YMAX]CHAN OF BOOL n.out, s.out, e.out, w.out :
[XMAX][YMAX]CHAN OF BOOL n.in, s.in, e.in, w.in, RunMode, selected :
[XMAX][YMAX]CHAN OF BOOL g1.clock, g2.clock, read, Done :
[XMAX][YMAX]CHAN OF BOOL BDone, BDone :
[YMAX]CHAN OF BOOL B1Done :
[XMAX]CHAN OF BOOL B2Done :
[8]CHAN OF BOOL Ready, OutputPort, InputPort :
[8]BOOL bit :
BOOL array.run.mode, halt, running, flag :
BOOL g1.register, g2.register, new.g1, new.g2, new.run.mode :
BOOL done, index :

INT x1, x2, x3, y1, y2, y3, y4, y5, y6, y7, y8 :
[XMAX][YMAX]CHAN OF BOOL Sync :
SEQ
  flag := FALSE
  running := TRUE
  x := 0
  y := 0
  PAR

The CHS2x4's array is constructed by calling the *Cell* procedure with the appropriate surrounding buffers in a nested PAR for loop - thus generating a XMAX by YMAX array of concurrent cells. In an effort to generate a very regular structure,
all of the cells in the array possess the identical code and functionality regardless of their position within the array. The cell's four buffers, however, depend heavily on the cell's x and y location as well as the I/O port locations. As an example, Figure 20 shows the appropriate cell-buffer configuration necessary for the CAL cell 0,0.

\[
\begin{align*}
\text{PAR } x &= 0 \text{ FOR } XMAX \\
\text{PAR } y &= 0 \text{ FOR } YMAX \\
\text{PAR } &
\text{Cell}\{\text{disp.channels[x][y]}, \text{x, y, Sync[x][y],} \\
&\quad \text{RunMode[x][y], selected[x][y],} \\
&\quad \text{Data.to.cell[x][y], Data.from.cell[x][y],} \\
&\quad \text{Cell.address[x][y], read[x][y],} \\
&\quad \text{n.in[x][y], s.in[x][y],} \\
&\quad \text{e.in[x][y], w.in[x][y],} \\
&\quad \text{n.out[x][y], s.out[x][y],} \\
&\quad \text{e.out[x][y], w.out[x][y],} \\
&\quad \text{gl.clock[x][y], g2.clock[x][y], Done[x][y]} \\
\end{align*}
\]

\[
\begin{align*}
\text{PAR -- buffers for cell 0,0} \\
\text{CreateNullBuffer(s.out[0][0], s.in[0][0], B2Done[0])} \\
\text{Buffer(n.out[0][0], s.out[0][1], n.in[0][0], s.in[0][1],} \\
&\quad \text{BXDone[0][0])} \\
\text{OutputBuffer(Ready[0], w.out[0][0], OutputPort[0], w.in[0][0],} \\
&\quad \text{B1Done[0])} \\
\text{Buffer(e.out[0][0], w.out[1][0], e.in[0][0], w.in[1][0],} \\
&\quad \text{BYDone[0][0])} \\
\end{align*}
\]

\[
\begin{align*}
\text{PAR -- buffers for cell 0,YMAX-1} \\
\text{CreateNullBuffer(n.out[0][YMAX-1], n.in[0][YMAX-1],} \\
&\quad \text{BXDone[0][YMAX-1])} \\
\text{CreateNullBuffer(w.out[0][YMAX-1], w.in[0][YMAX-1],} \\
&\quad \text{B1Done[YMAX-1])} \\
\text{Buffer(e.out[0][YMAX-1], w.out[1][YMAX-1], e.in[0][YMAX-1],} \\
&\quad \text{w.in[1][YMAX-1], BYDone[0][YMAX-1])} \\
\end{align*}
\]

\[
\begin{align*}
\text{PAR -- buffers for cell XMAX-1,0} \\
\text{Buffer(n.out[XMAX-1][0], s.out[XMAX-1][1], n.in[XMAX-1][0],} \\
&\quad \text{s.in[XMAX-1][1], BXDone[XMAX-1][0])} \\
\text{CreateNullBuffer(e.out[XMAX-1][0], e.in[XMAX-1][0],} \\
&\quad \text{BYDone[XMAX-1][0])} \\
\text{CreateNullBuffer(s.out[XMAX-1][0], s.in[XMAX-1][0],} \\
&\quad \text{B2Done[XMAX-1])} \\
\end{align*}
\]

\[
\begin{align*}
\text{PAR -- buffers for cell XMAX-1,YMAX-1} \\
\text{CreateNullBuffer(e.out[XMAX-1][YMAX-1], e.in[XMAX-1][YMAX-1],} \\
&\quad \text{BYDone[XMAX-1][YMAX-1])} \\
\text{CreateNullBuffer(n.out[XMAX-1][YMAX-1], n.in[XMAX-1][YMAX-1],} \\
&\quad \text{BXDone[XMAX-1][YMAX-1])} \\
\end{align*}
\]

\[
\begin{align*}
\text{PAR y1 = 0 FOR 7 -- buffers for front edge cells (OUTPUT)} \\
\text{PAR} \\
\text{Buffer(n.out[0][((y1+1)*2), s.out[0][((y1+1)*2)+1],} \\
&\quad \text{n.in[0][((y1+1)*2), s.in[0][((y1+1)*2)+1],} \\
&\quad \text{BXDone[0][((y1+1)*2))} \\
\text{OutputBuffer(Ready[y1+1], w.out[0][((y1+1)*2],}
\end{align*}
\]
OutputPort[y1+1], w.in[0][(y1+1)*2], B1Done[(y1+1)*2])
Buffer(e.out[0][(y1+1)*2], w.out[1][(y1+1)*2],
e.in[0][(y1+1)*2], w.in[1][(y1+1)*2],
BYDone[0][(y1+1)*2])

PAR y6 = 0 FOR 8 -- buffers for front edge cells (INPUT)
PAR
Buffer(n.out[0][(y6*2)+16], s.out[0][(y6*2)+17],
n.in[0][(y6*2)+16], s.in[0][(y6*2)+17],
BXDone[0][(y6*2)+16])
InputBuffer(InputPort[y6], w.out[0][(y6*2)+16],
w.in[0][(y6*2)+16], B1Done[(y6*2)+16])
Buffer(e.out[0][(y6*2)+16], w.out[1][(y6*2)+16],
e.in[0][(y6*2)+16], w.in[1][(y6*2)+16],
BYDone[0][(y6*2)+16])

PAR y8 = 0 FOR 16 -- buffers for front edge cells (non I/O)
PAR
Buffer(n.out[0][(y8*2)+1], s.out[0][(y8*2)+2],
n.in[0][(y8*2)+1], s.in[0][(y8*2)+2], BXDone[0][(y8*2)+1])
CreateNullBuffer(w.out[0][(y8*2)+1], w.in[0][(y8*2)+1],
B1Done[(y8*2)+1])
Buffer(e.out[0][(y8*2)+1], w.out[1][(y8*2)+1],
e.in[0][(y8*2)+1], w.in[1][(y8*2)+1],
BYDone[0][(y8*2)+1])

PAR y7 = 0 FOR 31 -- buffers for rest of front edge (non I/O)
PAR
Buffer(n.out[0][y7+32], s.out[0][y7+33], n.in[0][y7+32],
s.in[0][y7+33], BXDone[0][y7+32])
CreateNullBuffer(w.out[0][y7+32], w.in[0][y7+32],
B1Done[y7+32])
Buffer(e.out[0][y7+32], w.out[1][y7+32], e.in[0][y7+32],
w.in[1][y7+32], BYDone[0][y7+32])

PAR x1 = 0 FOR XMAX-2 -- buffers for bottom edge cells
PAR
Buffer(n.out[x1+1][0], s.out[x1+1][1], n.in[x1+1][0],
s.in[x1+1][1], BXDone[x1+1][0])
CreateNullBuffer(s.out[x1+1][0], s.in[x1+1][0], B2Done[x1+1])
Buffer(e.out[x1+1][0], w.out[x1+2][0], e.in[x1+1][0],
w.in[x1+2][0], BYDone[x1+1][0])

PAR x2 = 0 FOR XMAX-2 -- buffers for top edge cells
PAR
CreateNullBuffer(n.out[x2+1][YMAX-1], n.in[x2+1][YMAX-1],
BXDone[x2+1][YMAX-1])
Buffer(e.out[x2+1][YMAX-1], w.out[x2+2][YMAX-1],
e.in[x2+1][YMAX-1], w.in[x2+2][YMAX-1],
BYDone[x2+1][YMAX-1])

PAR y2 = 0 FOR YMAX-2 -- buffers for back edge cells
PAR
Buffer(n.out[XMAX-1][y2+1], s.out[XMAX-1][y2+2],
n.in[XMAX-1][y2+1], s.in[XMAX-1][y2+2],
BXDone[XMAX-1][y2+1])
CreateNullBuffer(e.out[XMAX-1][y2+1], e.in[XMAX-1][y2+1],
BYDone[XMAX-1][y2+1])

PAR xx = 0 FOR XMAX-2 -- buffers for internal cells
PAR yy = 0 FOR YMAX-2
PAR
Buffer(n.out[xx+1][yy+1], s.out[xx+1][yy+2],
n.in[xx+1][yy+1], s.in[xx+1][yy+2], BXDone[xx+1][yy+1])
Buffer(e.out[xx+1][yy+1], w.out[xx+2][yy+1],
e.in[xx+1][yy+1], w.in[xx+2][yy+1], BYDone[xx+1][yy+1])

"Normal" Buffer

Output Buffer
W.in
W.out

Cell 0,0

E.out
E.in

S.out
S.in

Null Buffer

Figure 20. Modeled Cell 0,0.

Following the cell creation code (but executing concurrently), there is a short sequence to control system timing. The entire system timing is based on the execution time of cell 0,0. At the beginning of every execution loop, cell 0,0 sends a synchronization pulse to the array. The ALT construct allows the array to receive a new synchronization pulse, reset its counter, output the current value of the counter, or terminate. If a signal is ready from cell 0,0, the array increments its counter and, if the counter exceeds some threshold value, allows system calls to be received. This synchronization protocol forces the system calls to always occur at a fixed speed that is relative to the speed of a cell. The interval duration is derived from the minimum execution time of a LocalTransfer instruction on the actual CHS2x4 divided by the cell propagation delay listed in the CAL1024 datasheet [JPG91].

SEQ
WHILE (NOT done)
ALT
The next concurrent code segment controls the distribution of global signals. During the normal operation of any configuration, a number of global signals are available to all cells. These include run.mode, G1 and G2. This code operates by monitoring a number _changing.to_ channels that inform it when a global signal is changing and to what it is changing to. Received changes are then distributed to every cell in the array. The occam code for this is shown below.

```ocaml
WHILE (running)
  PAR
  SEQ
    flag := FALSE
    WHILE (NOT flag)
      ALT
        gl.changing.to ? new.gl
          PAR i = 0 FOR XMAX
          PAR j = 0 FOR YMAX
          gl.clock[i][j] := new.gl
        g2.changing.to ? new.g2
          PAR i = 0 FOR XMAX
          PAR j = 0 FOR YMAX
          g2.clock[i][j] := new.g2
        run.mode.changing ? new.run.mode
          SEQ
          PAR i = 0 FOR XMAX
          PAR j = 0 FOR YMAX
          RunMode[i][j] := new.run.mode
      Done5 ? flag
      SKIP
```

While the above code is executing, the remainder of the array procedure code monitors the control signal bus, _Array.bus.mode_. Control signals received across this channel are decoded to produce a variety results based on the received mode. The valid bus modes are shown in Table 7. These differ somewhat from those of the actual CHS2x4 in that they include signals to facilitate the toggling of auto increment, run
mode, G1, and G2. The remainder of the modes attempt to mimic the actions that occur in the actual CHS2x4 for the same control signal.

```seq
SEQ
WHILE ((time <= BUS.DELAY) AND array.run.mode)
SEQ
  Sync.request = TRUE
  Sync.value = time
IF
  array.run.mode
  my.disp.channel = 0;0;time
  TRUE
  SKIP
Array.bus.mode = array.bus.mode
CASE array.bus.mode
Table 7
Modeled Bus Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reset</td>
</tr>
<tr>
<td>1</td>
<td>Write Address Counter</td>
</tr>
<tr>
<td>2</td>
<td>8 bit Write on low data, Read on high data</td>
</tr>
<tr>
<td>3</td>
<td>16 bit Write</td>
</tr>
<tr>
<td>4</td>
<td>16 bit Read</td>
</tr>
<tr>
<td>5</td>
<td>8 bit Write</td>
</tr>
<tr>
<td>6</td>
<td>8 bit Read</td>
</tr>
<tr>
<td>7</td>
<td>No Operation</td>
</tr>
<tr>
<td>8*</td>
<td>Toggle Auto Increment</td>
</tr>
<tr>
<td>9*</td>
<td>Toggle Run Mode</td>
</tr>
<tr>
<td>10*</td>
<td>Toggle g1</td>
</tr>
<tr>
<td>11*</td>
<td>Toggle g2</td>
</tr>
</tbody>
</table>
```

Mode 0 performs a simple reset on the array structure. The address register is set to zero while run mode, G1 and G2 are set to FALSE. No configuration data or RAM data is lost during this operation so operation can simply continue. The reset
operation has more significance in the CHS2x4 where the CAL outputs are forced into a high impedance state to prevent contention prior to loading a valid configuration.

```
BYTE (0)
-- ***********************************************************************
-- Mode 0  -> Reset the CAL
-- ***********************************************************************
SEQ
  Array.address ? array.address
  array.run.mode := FALSE
  g1.register := FALSE
  g1.changing.to 1 FALSE
  g2.register := FALSE
  g2.changing.to 1 FALSE
  Done5 1 TRUE
```

Mode 1 writes a value to the address counter. The *BusMaster* procedure controls access to the address counter so the array only receives a copy of this value and places it in *Array.address*.

```
BYTE (1)
-- ***********************************************************************
-- Mode 1  -> Set address.counter
-- ***********************************************************************
SEQ
  Array.address ? array.address
  Done5 1 TRUE
```

Mode 2 initiates a local transfer of data from the cells I/O ports to the RAM. When the array receives this modes, it begins to transfer data from the cell output ports by signaling each output buffer across the *Ready* lines. The *OutputPort* channels then receive individual bits from each buffer and eventual construct a byte of data. This data byte is output across the *Low.data.out* channel while a byte from the RAM is received from the *High.data.in* channel. Each input buffer then receives its appropriate bit of the new data across the *InputPort* channels. The final step in the data transfer forces *GI* high for approximately half a cycle to allow for the possible latching of this data in the interior of the array. After the cycle has completed, *GI* returns to its previous value.

```
BYTE (2)
-- ***********************************************************************
-- Mode 2  -> Transfer data between array and ram
```
modes 3, 4, 5 and 6 all perform similar operations that read or write configuration data to or from the cells within the array structure. During any these operations, the address sent to the array will be in the correct range for a cell address (the BusMaster procedure takes care of this). The two variables selx and sely indicate the particular cell of interest and are found using the decode.address procedure shown below:

```
-- ******************************************************
PROC decode.address(VAL INT address, INT x, INT y, BYTE new.address)
-- ******************************************************

-- decode address
-- first two bits are the byte in the cell
-- second set of bits is the row
-- third set of bits is the column
PAR
new.address := BYTE (address \ #03)
x := INT((address >> 2) \ XMASK)
y := INT((address >> 2) >> YSHIFT)
```
This simple procedure determines the cell's x and y coordinates, and which byte of configuration data is referenced by a particular address. The first two bits of the address reference the cell's configuration data while the remaining bits are divided among the x and y coordinates based on the array's size. Once the cell's x and y coordinates are found, the cell is selected using the selected channel. The appropriate read/write signal across Read indicates the type of operation. For data writes, an address and a byte value are sent. Reads send only an address and receive a byte. Implementing 16 bit reads and writes basically requires using the 8 bit routines twice. The occam code for these functions is shown below.

BYTE (3)
-- *********************************************
-- Mode 3 -> 16 bit write
-- *********************************************
SEQ
Array.address ? array.address
PAR
  Low.data.in ? low.data
  High.data.in ? high.data
decode.address(array.address, selx, sely, cell.address)
selected[selx][sely] ! TRUE
read[selx][sely] ! FALSE
Cell.address[selx][sely] ! cell.address
Data.to.cell[selx][sely] ! low.data
selected[selx][sely] ! TRUE
read[selx][sely] ! FALSE
Cell.address[selx][sely] ! BYTE (INT cell.address)+1
Data.to.cell[selx][sely] ! high.data
Done5 ! TRUE

BYTE (4)
-- *********************************************
-- Mode 4 -> 16 bit read
-- *********************************************
SEQ
Array.address ? array.address
decode.address(array.address, selx, sely, cell.address)
selected[selx][sely] ! TRUE
read[selx][sely] ! TRUE
Cell.address[selx][sely] ! cell.address
Data.from.cell[selx][sely] ? low.data
selected[selx][sely] ! TRUE
read[selx][sely] ! TRUE
Cell.address[selx][sely] ! BYTE (INT cell.address)+1
The next three modes toggle registers controlled by the array. Every time these registers are toggled their values must be distributed to every cell in the array. The .changing.to channels inform the global signal distribution code of these occurrences while Done5 signals the last such change for this loop iteration.
-- *********************************************
SEQ
IF
(gl.register)
  gl.register := FALSE
NOT (gl.register)
  gl.register := TRUE
  gl.changing.to ! gl.register
  Done5 ! TRUE
BYTE (11)
-- *********************************************
-- Mode 11 -> Toggle G2
-- *********************************************
SEQ
IF
(g2.register)
  g2.register := FALSE
NOT (g2.register)
  g2.register := TRUE
  g2.changing.to ! g2.register
  Done5 ! TRUE

The final mode terminates the array procedure. In order to exit the array code, each procedure started with it must first be terminated. This is accomplished through the Done channels. Each cell and buffer is forced to terminate itself upon receiving a TRUE signal across its Done channel.

BYTE (TERMINATE)
-- *********************************************
-- Mode 255 -> Terminate all processes
-- *********************************************
SEQ
PAR k = 0 FOR XMAX
 PAR l = 0 FOR YMAX
 PAR
  Done[k][l] ! TRUE
PAR k = 0 FOR XMAX
 PAR l = 0 FOR YMAX
 PAR
  BYDone[k][l] ! TRUE
 BXDone[k][l] ! TRUE
PAR k = 0 FOR YMAX
 B1Done[k] ! TRUE
PAR k = 0 FOR XMAX
 B2Done[k] ! TRUE
running := FALSE
 Done5 ! TRUE
 SDone ! TRUE
The Memory Model

The physical memory of the CHS2x4 contains four RAM sockets that can be used in three different configurations depending on the current bus mode. The modeled memory listed below, simplifies this structure by implementing a single bank of byte memory. The constant \( \text{RAM.BASE.ADDRESS} \) specifies the start of the RAM addressing while \( \text{RAMSIZE} \) indicates the number of byte addresses contained in the memory. Normal reads and writes simply index the memory array (RAM) after the raw address is adjusted by \( \text{RAM.BASE.ADDRESS} \). The local transfer instruction, however, requires a slightly more complex address decoding scheme. For this instruction, the address space is divided into two halves with writes occurring on one and reads on the other. The decoding scheme shown in the Memory procedure was developed to maintain consistency with the CHS2x4's interleaving of socket addressing. The basic scheme is shown in Table 8.

Table 8

<table>
<thead>
<tr>
<th>Requested Address</th>
<th>Use</th>
<th>Actual Address</th>
<th>Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ( \leftrightarrow ) MAX/4-1</td>
<td>Write</td>
<td>0 ( \leftrightarrow ) MAX/4-1</td>
<td>1</td>
</tr>
<tr>
<td>0 ( \leftrightarrow ) MAX/4-1</td>
<td>Read</td>
<td>MAX/4 ( \leftrightarrow ) MAX/2-1</td>
<td>2</td>
</tr>
<tr>
<td>MAX/4 ( \leftrightarrow ) MAX/2-1</td>
<td>Write</td>
<td>MAX/2 ( \leftrightarrow ) MAX*3/4-1</td>
<td>3</td>
</tr>
<tr>
<td>MAX/4 ( \leftrightarrow ) MAX/2-1</td>
<td>Read</td>
<td>MAX*3/4 ( \leftrightarrow ) MAX-1</td>
<td>4</td>
</tr>
</tbody>
</table>

PROC Memory(CHAN OF INT Address, CHAN OF BYTE Low.data.in, Low.data.out, High.data.in, High.data.out, Bus.mode)
BYTE mode :
INT my.address, my.read.address, my.write.address :
[RAMSIZE]BYTE RAM :

WHILE (mode <> (BYTE TERMINATE))
SEQ
PAR
Bus.mode ? mode
Address ? my.address
CASE mode

BYTE (2) -- local transfer
SEQ
my.read.address := my.address + (INT #080000)
my.write.address := my.address
IF
my.address >= (INT #280000)
PAR
my.read.address := my.read.address + (INT #100000)
my.write.address := my.write.address + (INT #080000)
TRUE
SKIP
High.data.out 1 RAM[my.read.address-RAM.BASE.ADDRESS]
Low.data.in ? RAM[my.write.address-RAM.BASE.ADDRESS]

BYTE (3) -- 16 bit write to RAM
SEQ
my.address := my.address - RAM.BASE.ADDRESS
Low.data.in ? RAM[my.address]
High.data.in ? RAM[my.address + (INT 1)]

BYTE (4) -- 16 bit read from RAM
SEQ
my.address := my.address - RAM.BASE.ADDRESS
Low.data.out 1 RAM[my.address]
High.data.out 1 RAM[my.address + (INT 1)]

BYTE (5) -- 8 bit write to RAM
SEQ
my.address := my.address - RAM.BASE.ADDRESS
Low.data.in ? RAM[my.address]

BYTE (6) -- 8 bit read from RAM
SEQ
my.address := my.address - RAM.BASE.ADDRESS
Low.data.out 1 RAM[my.address]

ELSE
SKIP

The Bus Master Model

The following BusMaster procedure models the majority of the CHS2x4's control circuitry using the flow diagram shown below. Its primary purpose is to
produce the necessary control signals required to execute the functions listed earlier in Table 1.

Figure 21. BusMaster Procedure Basic Flow Diagram.

Prior to starting its main loop, the BusMaster procedure initiates the Memory and Array procedures, both of which it has direct control over.

-- ************************************************************************
PROC BusMaster([XMAX][YMAX]CHAN OF DISPLAY cell.disp.channels,
   CHAN OF DISPLAY array.disp.channel,
   CHAN OF BYTE Sys.bus.mode,
   Low.data.to.Sys, Low.data.from.Sys,
   High.data.to.Sys, High.data.from.Sys,
   CHAN OF INT Sys.address)
-- ************************************************************************

-- controls access to the system buses based on the
-- value of bus.mode which can only by changed by the system
CHAN OF INT Array.address, Memory.address:
INT address.counter, time:

CHAN OF BYTE Memory.bus.mode, Array.bus.mode,
Memory.low.data.in, Memory.low.data.out,
Memory.high.data.in, Memory.high.data.out,
Low.data.to.Array, Low.data.from.Array,
High.data.to.Array, High.data.from.Array:

BYTE low.data, high.data:
BYTE bus.mode:

BOOL auto.increment:

TIMER clock:

SEQ
PAR
  Memory(Memory.address, Memory.low.data.out,
         Memory.low.data.in, Memory.high.data.out,
         Memory.high.data.in, Memory.bus.mode)
  Array(cell.disp.channels, array.disp.channel,
         High.data.to.Array, High.data.from.Array,
         Low.data.to.Array, Low.data.from.Array,
         Array.bus.mode, Array.address)
  WHILE (bus.mode <> (BYTE TERMINATE))

The procedure then waits on a mode signal from a user function call. A large
CASE statement decodes each mode value into the appropriate control signals for both
the Memory and Array procedures.

SEQ
  Sys.bus.mode ? bus.mode -- receive any system mode
  CASE bus.mode
    BYTE (0)
      -- ****************************************************
      -- Mode 0 -> Reset CAL
      -- ****************************************************
      --
      SEQ
      PAR
       address.counter := INT 0
       auto.increment := TRUE
      PAR
       Array.bus.mode = bus.mode
       Array.address = address.counter
    BYTE (1)
      -- ****************************************************
      -- Mode 1 -> Write address counter
      -- ****************************************************
      --
      SEQ
      Sys.address ? address.counter
      PAR
       Array.address = address.counter
       Array.bus.mode = bus.mode
BYTE (2)
-- ******************************************************************************
-- Mode 2 -> 8 bit transfer from array to ram on LD
-- ram to array on HD
-- ******************************************************************************
SEQ
PAR
Array.bus.mode | bus.mode
Memory.bus.mode | bus.mode
Memory.address | address.counter
PAR
Low.data.from.Array ? low.data
Memory.high.data.in ? high.data
PAR
Memory.low.data.out | low.data
High.data.to.Array | high.data
IF
(auto.increment)
  address.counter := address.counter + (INT 1)
TRUE
SKIP

BYTE (3)
-- ******************************************************************************
-- Mode 3 -> 16 bit write
-- ******************************************************************************
SEQ
PAR
Low.data.from.Sys ? low.data
High.data.from.Sys ? high.data
IF
  address.counter >= RAM.BASE ADDRESS
PAR
  Memory.bus.mode | bus.mode
  Memory.address | address.counter
  Memory.low.data.out | low.data
  Memory.high.data.out | high.data
  address.counter < CONTROL.CAL.BASE ADDRESS
PAR
  Array.address | address.counter
  Array.bus.mode | bus.mode
  Low.data.to.Array | low.data
  High.data.to.Array | high.data
IF
  (auto.increment)
  address.counter := address.counter + (INT 2)
TRUE
SKIP

BYTE (4)
-- ******************************************************************************
-- Mode 4 -> 16 bit read from CAL
-- ******************************************************************************
SEQ
IF
  address.counter >= RAM.BASE ADDRESS
PAR
  Memory.bus.mode | bus.mode
  Memory.address | address.counter
Memory.low.data.in ? low.data
Memory.high.data.in ? high.data
address.counter < CONTROL.CAL.BASE.ADDRESS
PAR
    Array.bus.mode l bus.mode
    Array.address l address.counter
    Low.data.from.Array ? low.data
    High.data.from.Array ? high.data
PAR
Low.data.to.Sys l low.data
High.data.to.Sys l high.data
IF
    (auto.increment)
    address.counter := address.counter + (INT 2)
    TRUE
    SKIP

BYTE (5)
-- **************************************************************************
-- Mode 5 -> 8 bit write
-- **************************************************************************
SEQ
Low.data.from.Sys ? low.data
IF
    address.counter >= RAM.BASE.ADDRESS
    PAR
        Memory.bus.mode l bus.mode
        Memory.address l address.counter
        Memory.low.data.out l low.data
    address.counter < CONTROL.CAL.BASE.ADDRESS
    PAR
        Array.bus.mode l bus.mode
        Array.address l address.counter
        Low.data.to.Array l low.data
    IF
        (auto.increment)
        address.counter := address.counter + (INT 1)
        TRUE
        SKIP

BYTE (6)
-- **************************************************************************
-- Mode 6 -> 8 bit read from CAL
-- **************************************************************************
SEQ
IF
    address.counter >= RAM.BASE.ADDRESS
    PAR
        Memory.bus.mode l bus.mode
        Memory.address l address.counter
        Memory.low.data.in ? low.data
    address.counter < CONTROL.CAL.BASE.ADDRESS
    PAR
        Array.bus.mode l bus.mode
        Array.address l address.counter
        Low.data.from.Array ? low.data
    Low.data.to.Sys l low.data
    IF
        (auto.increment)
        address.counter := address.counter + (INT 1)
BYTE (7)
   -- *****************************************************
   -- Mode 7 -> No operation
   -- *****************************************************
SEQ
   SKIP

BYTE (8)
   -- *****************************************************
   -- Mode 8 -> Toggle auto increment
   -- *****************************************************
SEQ
   IF
     (auto.increment)
     auto.increment := FALSE
   NOT (auto.increment)
     auto.increment := TRUE

BYTE (9)
   -- *****************************************************
   -- Mode 9 -> Toggle run mode
   -- *****************************************************
SEQ
   Array.bus.mode ! (BYTE 9)

BYTE (10)
   -- *****************************************************
   -- Mode 10 -> Toggle G1
   -- *****************************************************
SEQ
   Array.bus.mode ! (BYTE 10)

BYTE (11)
   -- *****************************************************
   -- Mode 11 -> Toggle G2
   -- *****************************************************
SEQ
   Array.bus.mode ! (BYTE 11)

ELSE
   -- *****************************************************
   -- Mode 255 -> Terminate
   -- *****************************************************
SEQ
   PAR
     Memory.bus.mode ! (BYTE TERMINATE)
     Memory.address ! address.counter
     Array.bus.mode ! (BYTE TERMINATE)
The System Function Models

The 15 control instructions or functions used by the CHS2x4 are all reproduced with the individual occam procedures listed below. In the physical CHS2x4, the user initiates one of these functions by referencing a particular I/O address. Similarly, each modeled function is initiated by calling the function's occam procedure and passing any required data or channels as parameters.

The operation of the function procedures typically includes the transmission of a bus mode (from Table 7) and the sending or receiving of data values. Three functions; LoadAddressByte0, LoadAddressByte1, and LoadAddressByte2, do not require any communication with other systems because their register values were kept local to user's program until the entire address was constructed. These functions were incorporated in the SetBaseAddress instruction to facilitate array configuration and program size.

PROC Terminate( CHAN OF BYTE Mode)
-- procedure to halt all processes
PAR
    Mode ! (BYTE TERMINATE)
;

PROC Reset(CHAN OF BYTE Mode)
-- procedure to reset the CAL array
-- clears g1, g2 and run mode
-- this is bus mode 0
SEQ
    Mode ! (BYTE 0)
;

PROC SetBaseAddress(VAL INT address.counter,
CHAN OF BYTE Mode, CHAN OF INT Address)
--
SEQ
  Mode 1 (BYTE 1)
  Address 1 address.counter
:
-- *******************************************************************************
PROC ToggleAutoincrement( CHAN OF BYTE Mode)
-- *******************************************************************************

SEQ
  Mode 1 (BYTE 8)
:
-- *******************************************************************************
PROC ToggleRunMode(CHAN OF BYTE Mode)
-- *******************************************************************************

SEQ
  Mode 1 (BYTE 9)
:
-- *******************************************************************************
PROC ToggleGl(CHAN OF BYTE Mode)
-- *******************************************************************************

SEQ
  Mode 1 (BYTE 10)
:
-- *******************************************************************************
PROC ToggleG2(CHAN OF BYTE Mode)
-- *******************************************************************************

SEQ
  Mode 1 (BYTE 11)
:
-- *******************************************************************************
PROC ReadByte(BYTE returned.byte, CHAN OF BYTE Mode, LData.in)
-- *******************************************************************************

-- read a byte from the address in the address.counter
-- and place it in returned.byte
-- the address.counter is incremented if auto.increment is TRUE

SEQ
  Mode 1 (BYTE 6)
  LData.in ? returned.byte
:
-- *******************************************************************************
PROC WriteByte(VAL BYTE sent.byte, CHAN OF BYTE Mode, LData.out)
-- *******************************************************************************

-- writes the given byte to the address in the address.counter
-- the address.counter is increment if auto.increment is TRUE
-- this is used to store data in the CAL ram and to
-- configure the array

SEQ
  Mode ! (BYTE 5)
  LData.out ! sent.byte
;

-- *******************************************************
PROC ReadWord(INT returned.word,
               CHAN OF BYTE Mode, LData.in, HData.in)
-- *******************************************************

-- similar to read byte, excepts uses both the low bus
-- and the high data bus

BYTE bytel, byte2 :

SEQ
  PAR
  Mode ! (BYTE 4)
  LData.in ? bytel
  HData.in ? byte2
  returned.word := (INT bytel) + ((INT byte2) << 8)
;

-- *******************************************************
PROC WriteWord(VAL INT sent.word,
               CHAN OF BYTE Mode, LData.out, HData.out)
-- *******************************************************

-- similar to read byte, excepts uses both the low bus
-- and the high data bus

SEQ
  Mode ! (BYTE 3)
  LData.out ! (BYTE (sent.word \ #00FF))
  HData.out ! (BYTE (sent.word >> 8))
;

-- *******************************************************
PROC LocalTransfer(CHAN OF BYTE Mode)
-- *******************************************************

-- causes a transfer of data from the CAL array to
-- the RAM on the low bus, and from the RAM to the CAL array
-- on the high bus

SEQ
  Mode ! (BYTE 2) -- 8 bits write on low, read on high
;

An Arbitrary Configuration Model

By implementing the function calls described in the previous section, any arbitrary configuration of the CHS2x4 model can be achieved. A list or "user
program" of function calls is constructed to reset, configure, and execute the CHS2x4 model. Below is an example of a Main procedure where the user's program is contained. Also shown is the initiation of the BusMaster procedure that operates in parallel with the program and provides control signals for each function call.

```
PROC Main(CHAN OF SP fs, ts)

CHAN OF INT Address:
INT time, tmp :

CHAN OF BYTE Mode, LData.in, LData.out, HData.in, HData.out :
BYTE tmp2 :

CHAN OF BOOL Done :
TIMER clock :

[XMAX][YMAX]CHAN OF DISPLAY cell.disp.channels :
CHAN OF DISPLAY array.disp.channel :
CHAN OF DISPLAY main.disp.channel :
CHAN OF BOOL disp.done :

SEQ

SEQ
Reset(Mode)
-- the user's configuration is contained in "program"
Program(Mode, LData.in, LData.out, HData.in, HData.out, Address)
-- store initial data in RAM
SetBaseAddress(#280000, Mode, Address)
WriteWord(INT #5191, Mode, LData.out, HData.out)
SetBaseAddress(#200000, Mode, Address)
-- put cells in run mode
ToggleRunMode(Mode)
clock ? time
main.disp.channel ! 1;0;time
-- perform some Local Transfer (or any other) instructions
LocalTransfer(Mode)
LocalTransfer(Mode)
clock ? time
main.disp.channel ! 0;0;time
-- shut run mode off and terminate program
ToggleRunMode(Mode)
Terminate(Mode)
SEQ
BusMaster(cell.disp.channels, array.disp.channel, Mode,
LData.in, LData.out, HData.in, HData.out,
Address)
SEQ
Display(fs, ts, cell.disp.channels, array.disp.channel,
main.disp.channel, disp.done)
```
Figures 9 and 10 previously displayed the model's overall control and data flows. These indicated the signal directions and the hierarchical nature of the CHS2x4 model. Each level in this hierarchy controls the levels below it either directly or indirectly through the intermediary levels. The blocks shown in these figures represent the occam procedures that are executing in parallel. When one of these procedure receives a control signal from the level above, it typically performs some tasks (perhaps involving communication with lower levels), responds to a higher level, and then waits for another control signal.

Model Description

Conclusions

The occam language provided sufficient syntax and structure to accommodate the modeling of the CHS2x4's hardware and controlling software. The language's PAR, SEQ, and ALT constructs allowed concurrent hardware components to be easily defined and executed while the channel facilities permitted communication to occur between the concurrent components.

Though occam supplied many necessary language components, it also contributed a number of inconveniences and difficulties during the construction of the CHS2x4 model. First, occam only supports one type of global signal, a TIMER. A TIMER channel is not user alterable and simply returns an integer value representing the current time associated with that particular TIMER. Any controllable global signal, such as G1 and G2 in the CHS2x4, must be manually distributed to each applicable location. The second problem, synchronization, manifests itself during nearly all parallel tasking. Extreme care had to be taken to avoid deadlock, livelock, and a variety of other synchronization issues. Even though occam supplies barrier synchronization for the completion of concurrent processes in each PAR construct,
there was still a substantial amount of synchronization that had to be manually coded in the model. This problem is further compounded by the difficulties associated with debugging parallel processes.
CHAPTER V
MODEL VERIFICATION

The model verification process incorporates the principles of accuracy and precision to assure the validity of the model. It is desirable that the model be accurate, i.e. it returns consistent results that resemble those of the modeled system. The verification process described here utilizes the Algotronix CAD tool CLARE and its utility cfg2cal to create ASCII and binary configuration files (.cfg and .cal respectively). The .cal files are downloaded to the CHS2x4 while the .cfg files (Appendix O) are converted directly to occam by a Perl [Wall91] program called cfg2occ (Appendix E) and inserted into the model. Initially, the model's precision is determined through the analysis of near exhaustive configurations. The process then verifies the model's accuracy through the comparison of real and experimental results obtained from a running a selection of test cases on the CHS2x4 and model.

Precision Verification

The precision verification of the CHS2x4 model requires that it returns consistent results regardless of the configuration. These results are generally categorized to include functionality and timing. Any particular configuration should execute within certain time constraints and perform the intended function every time it is employed. Without these simple criteria the model is unstable and basically useless.

In an effort to detect aberrant behavior, the precision verification process employs a variety of configurations that constitute a large percentage of the possible
permutations for the given array. Because of the potential for an incredible number of configuration permutations, the process relies heavily on the symmetry of the array and cell structures.

The symmetry of the array structure stems from its implicit mesh pattern. Any array can, at most, contain cells of nine different localities. These include 4 corner cells, 4 different edge cells (north, south, east, and west edges), and an internal cell. A 3x3 array is the minimum sized array that contains all of these primary cell localities with no duplications. The verification of one cell in each of these locations provides sufficient results to qualify identical cells in similar locations. Arbitrarily large arrays can therefore be verified by examining the operation of a minimum sized 3x3 array.

A slightly finer grain symmetry exists within the internal structure of each cell. The routing multiplexers in every cell perform similar routing tasks and are all constructed with nearly identical occam code. Similarly, all cell functional units are also identical. It is therefore redundant to test all possible routing configurations and multiple functional units.

Based on the above array and cellular symmetries, the precision verification methodology was divided into a 1x1 array procedure and a 3x3 array procedure. The 1x1 procedure concentrated on the cellular functionality and intracell routing while the 3x3 procedure dealt more with intercell routing and buffer operations.

Cellular Verification Methodology and Results

The precise operation of the cell model consists of correct functionality and consistent execution times. To show both of these, a 1x1 (single cell) array structure was repeatedly configured to execute all possible cell functions. Sample data was
streamed through the cell, and the cell output and execution time were stored. The following pseudo code briefly outlines this procedure:

```
FOR function = 0 to 19
    configure.cell( function )
    store.input.data()
    record.start.time( start.time[ function ] )
    stream.data.through.cell()
    retrieve.output.data()
    record.end.time( end.time[ function ] )
NEXT function
```

The timing results from this process, summarized in Figure 22, represent the average amount of time (in occam time units) the array took to operate four \textit{LocalTransfer} instructions and then retrieve four bytes of data with the \textit{.ReadByte} instruction using a total of 50 iterations for each of the 20 CHS2x4 functions.

![Figure 22. Cell Verification Results.](image)

The functional outputs obtained from each function's 50 iterations were consistent with the output expected from similar logic functions, i.e. an AND configuration's output matched that of an AND logic gate. Appendix B contains each iteration's data output and timing in its entirety.
From these results it is apparent that the occam cell model returns function values consistent with its configuration and executes within a ± 2.5% time tolerance. The relatively small deviations exhibited by each function can mainly be attributed to the Unix operating system and, therefore, sufficiently demonstrate the model's timing stability while the consistent output values verify the cell's functionality. Based on the symmetry characteristics discussed earlier, it is now possible to verify larger scale arrays without dealing with the operation of each specific cell in the array.

Array Verification Methodology and Results

Once the operation of each cell has been validated, the main verification concerns lie with the cell interconnections and the array scalability. It is imperative that both of these items do not alter the overall array functionality and that they produce negligible or at least predictable time variances.

In the area of cell interconnections, there are four basic routing paths, shown in Figures 23, 24, 25, and 26, that comprise all larger paths. Each basic path was modeled and executed on a 3x3 array structure to yield timing and functionality data (listed in Appendix C). A procedure identical the one described earlier was used to stream data through the array and into RAM. The average execution times and their corresponding standard deviations for each of the four basic paths are shown in Figure 27. As with the cellular timing and functionality data, the 3x3 data demonstrated the validity of routing within the array structure.

Precision Verification Conclusions

The purpose of the precision verification process was to show the model to be stable and well-behaved. The cell analysis procedure indicated that the configuration
of any particular cell, both functional and routing, does not influence the execution time for that given cell. It was also shown through this procedure that the cell's computational unit produces output that is consistent with the cell's current configuration.

The array analysis procedure further verified the model's precision by routing data through four basic paths. Since the array structure is symmetrical and regular, the validity of the four basic paths can be extended to the longer, more complex paths that they may constitute. As with the cellular results, the array procedure's results demonstrated the consistency of the four basic paths to route data correctly and to do so within certain time constraints.
Accuracy Verification

An accurate model returns consistent results that (ideally) completely reproduce those of the modeled system. In the case of the CHS2x4 model, accurate functionality and execution timings are the primary objectives. The best course to achieving this verification and the method employed here is the use of test cases to compare actual and modeled results. The accuracy verification process will consists of the execution of selected test cases on the CHS2x4 hardware and model, and the comparison of their functional and timing results. It will also include a number of calculations that will be used to derive the model's timing parameters.

Propagation Delay Calculation

The CAL1024 datasheet and the CHS2x4 User Manual state that a typical cell to cell routing takes approximately 1.7 ns and that the CHS2x4 operates at the
standard 8 Mhz AT bus frequency [JPG91]. These two figures suggest that the maximum signal path should not exceed 73 cells, but due to system overhead and from experimental results it was found that is not a good approximation of the PC interface's speed. To achieve a more realistic instruction frequency and maximum path length, the minimum execution time of 20,000 $LocalTransfer$ instructions was recorded. This resulted in an average instruction issue frequency of 2.5µs and a maximum path length of approximately 1470 cells. Based on this, the model allows one instruction to be issued every 1470 cell iterations.

**Computational Delay Calculation**

Along with the propagation delay, the CAL1024 datasheet [JPG91] also lists the typical computation delay of 7ns. Dividing this by the nominal 1.7ns for the cell propagation delay results in a ratio of approximately 4:1. The model incorporates this delay through a 3 bit buffer in each cell. A 4 bit buffer is not required as a delay of 1 cell execution is encountered during routing.

**Test Case #1: Seven Segment Decoder**

A seven segment decoder was the first test case incorporated on the model and CHS2x4. The design's CAD file (*.cfg) was taken from the examples included with the CHS2x4's hardware [JPG91] and converted to occam format using cfg2occ. The use of this program allows any flattened (i.e. containing no externally referenced instances of other designs) CHS2x4 .cfg file to be placed directly into the model, compiled, and executed without any modifications. The decoder example mainly
served to verify the accuracy of the cfg2occ program and to demonstrate the linearity of the model.

The verification of the cfg2occ program was simply a byproduct of the model's execution, i.e. if the model returned results consistent with the operation of a seven segment decoder, then the program translates code correctly. On the other hand, the demonstration of the model's linearity involved executing the decoder configuration multiple times at varying execution speeds. Ideally, doubling the execution speed should half the execution time, halving the executing speed should double the execution time, and so on. For the decoder case, the execution speed was varied from 8 times normal to 1/8th of normal with each speed being repeated 10 times for an average. This variance in execution speed was accomplished by simply multiplying the $BUS.DELAY$ constant by the inverse of the desired speed up, e.g. a speed increase of 2 would be accomplished by multiplying the base delay (1470) by $1/2$, resulting in a new $BUS.DELAY$ of 735. The results of this process are listed in Appendix D and summarized in Tables 9 and 10.

Table 9 lists the inputs and the corresponding outputs obtained from execution the 7 segment decoder on the CHS2x4 and the model. Note that the model is reproducing the function of the CHS2x4.

The results shown in Table 10 indicate that the model performs linearly with its execution speed (or bus delay.) The Relative Execution Speed column indicates the inverse of the scaling performed on the $BUS.DELAY$ constant shown in the Cell Delays Per Cycle column while the Normalized Execution Speed column represents the model's execution time relative to the base execution time or that at a relative speed of 1. Ideally, these should be identical. The scaling error that is apparent toward the upper end of the execution speeds is caused by the execution time
approaching the lower bound of the model. The deviations that exist toward the lower execution speeds come from variations in system overhead as these models require extensive CPU time to complete.

Table 9

Decoder Model Functional Results

<table>
<thead>
<tr>
<th>Input</th>
<th>CHS2x4 Output</th>
<th>Model Output</th>
<th>Input</th>
<th>CHS2x4 Output</th>
<th>Model Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>01111111</td>
<td>01111111</td>
<td>0101</td>
<td>01101101</td>
<td>01101101</td>
</tr>
<tr>
<td>0001</td>
<td>00000110</td>
<td>00000110</td>
<td>0110</td>
<td>01111101</td>
<td>01111101</td>
</tr>
<tr>
<td>0010</td>
<td>01011011</td>
<td>01011011</td>
<td>0111</td>
<td>00000111</td>
<td>00000111</td>
</tr>
<tr>
<td>0011</td>
<td>01001111</td>
<td>01001111</td>
<td>1000</td>
<td>01111111</td>
<td>01111111</td>
</tr>
<tr>
<td>0100</td>
<td>01100110</td>
<td>01100110</td>
<td>1001</td>
<td>01101111</td>
<td>01101111</td>
</tr>
</tbody>
</table>

Table 10

Decoder Model Linearity Results

<table>
<thead>
<tr>
<th>Relative Execution Speed</th>
<th>Cell Delays Per Cycle</th>
<th>Average Execution Time</th>
<th>Standard Deviation</th>
<th>Normalized Execution Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>184</td>
<td>908094</td>
<td>1166.5</td>
<td>7.63</td>
</tr>
<tr>
<td>4</td>
<td>368</td>
<td>1779594</td>
<td>2689.6</td>
<td>3.89</td>
</tr>
<tr>
<td>2</td>
<td>735</td>
<td>3508874</td>
<td>2503.9</td>
<td>1.97</td>
</tr>
<tr>
<td>1</td>
<td>1470</td>
<td>6928276</td>
<td>1661.6</td>
<td>1.00</td>
</tr>
<tr>
<td>1/2</td>
<td>2940</td>
<td>13879516</td>
<td>72743.6</td>
<td>1/2.00</td>
</tr>
<tr>
<td>1/4</td>
<td>5880</td>
<td>27847874</td>
<td>20612.3</td>
<td>1/4.02</td>
</tr>
<tr>
<td>1/8</td>
<td>11760</td>
<td>55916609</td>
<td>66128.9</td>
<td>1/8.07</td>
</tr>
</tbody>
</table>
Test Case #2: (Massively) Parallel Sorting

The second test case applied to the CHS2x4 and model was that of a massively parallel bit sorting algorithm. Because of the limitations of the CHS2x4's `cfg2cal` program, the algorithm was (severely) limited to sorting only three 8 bit numbers. Nevertheless, the purpose here was to take a new design, develop it using the CHS2x4's CAD tools, run the completed design on both the CHS2x4 and model, and then compare both results for conformity.

The sorting algorithm employed uses a $n \times m$ mesh array of 1 bit processors to sort $n$, $m$ bit numbers. Each bit processor uses the following rules:

1. A bit is received from the left and a message (L, R, or $=$) from above.
2. If L is received; output the newly received bit to the right and L to the bottom.
3. If R is received; output the stored bit to the right and R to the bottom, and store the newly received bit.
4. If $=$ is received; compare the newly received bit and the stored bit, output the smaller bit to the right, and the appropriate L, R, or $=$ to the bottom depending on the result of the comparison.
5. Repeat

The above rules translated into the logic diagrams shown in Figures 28-30. The A and B inputs correlate to the control signal received from above while C and D are the output signals sent to the cell below. A control signal of L equates to $AB = 00$, R correlates to $AB = 10$, and $=$ maps to $AB = x0$. The output signals, C and D, are mapped in a similar fashion. The two control signals, A and B, along with the input, In, and the comparator's current state, Latch State, determine the comparator's output and it's next state as shown in Figures 29 and 30. Note that the C, D, and
Latch State signals are delayed one cycle. This prevents a race from occurring with the Latch State signal and provides for signal synchronization with the C and D control signals. All four logic diagrams mapped onto the 5x6 cellular structure shown in Figure 31. The mapping was extremely efficient with each 5x6 structure having only 5 unused cells for an 83.3% usage. These 1-bit structures were placed in a mesh-like array where delay blocks were added to correctly skew the incoming and outgoing data. The resulting circuit is shown in Figure 32. Similar to the 1-bit structures, the entire design also exhibited efficient usage of cells by permitting the cellular structures to be vertically adjacent and nearly horizontally adjacent. Typically, this is not the case as poor utilization density is a common problem found in most FPGA designs.

![Logic Diagrams](image)

**Figure 28.** Bit Comparator's Control Signal Logic Diagrams - C and D Outputs.

To get the design to operate correctly, the circuitry directly at the top of each column was added to facilitate the flushing of the array. Since the state of the CHS2x4's latches are initially unknown, it was imperative that each bit processor's
state be set to an initial condition of zero. This does create a discrepancy between the CHS2x4 and model in that the model's latches (and registers) default to an initial state of zero. The implications of this discrepancy are by far the greatest for this particular class of algorithms (comparisons) and rather minimal for the majority of others. The additional circuitry, however, did successfully flush the array and allow the sorting to proceed correctly.

![Diagram of Bit Comparator's Output Logic Diagram](image)

Figure 29. Bit Comparator's Output Logic Diagram.

![Diagram of Bit Comparator's New State Logic Diagram](image)

Figure 30. Bit Comparator's New State Logic Diagram.
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Input</th>
<th>CHS2x4 Output</th>
<th>Model Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>155</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>Toggle G2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>34</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>150</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>25</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>250</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>55</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>255</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>255</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>255</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>255</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>155</td>
<td>155</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The control codes for this circuitry, 'C' code for the CHS2x4 and occam code for the model, shown in Appendices F and G, identically executed 9 *LocalTransfer* instructions to initially flush the array, toggled the global signal G2 high to signal the completion of the flushing, and finally issued 19 *LocalTransfer* instruction to stream data values through the array and back to memory. Table 11 displays these processes and their results. Note that the CHS2x4 and model outputs agree completely during the instructions that follow the flushing operations.

Figure 31. Parallel Sorter Bit Processing Element.
Figure 32. Parallel Sorter Circuit.

**Test Case #3: Bit Serial Adder**

The final test case involved a bit serial adder that could add 8 numbers of virtually any bit width at the same time. This was accomplished through the adder configuration shown in Figures 33 and 34. The individual bits of all 8 numbers are
serially pipelined in from the upper input ports and cascaded through the serial adders. These bits are added and forwarded to the next stage while the carry feeds backs for the next operation. After three stages (when using $2^3$ numbers), a 1 bit serial result is returned out the lowest output port.

![Bit Serial Adder Circuit](image)

Figure 33. Bit Serial Adder Circuit.
The purpose of this example was identical to that of the previous parallel sorter in that it further proved to verify the accuracy of the CHS2x4 model. As with the parallel sorter, the bit serial adder's logic was completely designed in CLARE, converted by cfg2cal and cfg2occ, and executed on the CHS2x4 and occam model. The results of adding seven, 8 bit numbers are shown in Table 12. The control codes for the CHS2x4 and model are listed in Appendices H and I, respectively.

This particular implementation of the algorithm added the numbers 1, 22, 112, 39, 12, 60, 100, and 59, resulting in a final sum of #110010101 or 405. The binary values listed in the Input column of Table 12 represent the one bit from each of the eight numbers that is being added during that particular cycle. The values in the column also represents the format used to store the input numbers in memory. Initially, zeroes were streamed into the adder to clear the latches. This period is
indicated in the CHS2x4 Output column with a number of question marks to show that the output during the initial instructions is invalid because of the unknown initial state of the CHS2x4's latches. Aside from the invalid initial values, the CHS2x4 and occam model outputs once again coincided and verified the accuracy of model.

Table 12

Bit Serial Adder Results

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Input</th>
<th>CHS2x4 Output</th>
<th>Model Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>0</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>00010001</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>01010001</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>01011110</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>00001101</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>01100101</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>00110111</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>00100010</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>00000000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>00000000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local Transfer</td>
<td>?</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Accuracy Verification Conclusions

An accurate model predicts the performance and behavior of the modeled system to the degree it was designed. In this case, the occam model predicts the functionality and execution timing of any valid CHS2x4 configuration. Initially, the model's timing parameters were derived from timing values given in the CAL1024 datasheet and the CHS2x4 User Manual. These were used in the modeling of three test cases that demonstrated the accuracy of the model by comparing CHS2x4 results to those of the occam model.

The 7 segment decoder test case used a Perl program called cfg2occ to convert the CHS2x4's CAD tool, CLARE's output to occam code. The decoder model operated correctly and returned values identical to those of the CHS2x4. The values were also received during the correct LocalTransfer instructions, indicating a degree of timing accuracy. This example also served to prove the linearity of the model. It was shown that there is a near 1:1 relationship between clock scaling and model execution time. The 1:1 relationship does not hold true, however, as the clock speed approaches the minimum time required to execute the model. This phenomenon resembles that of logic circuits where the clock frequency is near or exceeds the operating speed of the logic gates themselves. Such issues have little impact as long as clock frequencies remain in tolerable ranges or device speeds increase accordingly.

The second test case, the (massively) parallel sorter was designed entirely through the use of CLARE and followed the same conversion processes as the seven segment decoder. Much like the decoder, the sorting algorithm also demonstrated the model's functionality and timing accuracy, but it did so using a significantly larger design. Once again the model's and CHS2x4's results were identical in both functionality and timing. The success of this algorithm demonstrates that the model's
timing accuracy holds for small designs were signal paths are relatively short, and for large designs were signals paths are long and propagation delays may critically approach the system's clock period.

The final test case, the bit serial adder, served also to verify the model's functional and timing accuracy. It was developed similar to the parallel sorter through CLARE and the two conversion programs, cfg2cal and cfg2occ. As before, the model and CHS2x4 both returned identical results.

Model Verification Conclusions

Through the precision and accuracy verification methodologies discussed above, it was shown that the CHS2x4 occam model successfully predicts the functionality and execution timing of the CHS2x4 Custom Computer with very few deviations. Multiple iterations of cell configuration and routing examples displayed correct cellular functionality, routing, and execution times. Likewise, iterations of the basic array routing paths resulted in consistent data transmission and path delays. Three arbitrary test cases confirmed the model's functional and execution timing accuracy by successfully comparing the model's output to that of the CHS2x4.

The verification process did discover one important deviation that exists between the model and the CHS2x4. It was observed that the initial state of the CHS2x4 latches is not preset to any predetermined value upon configuration while the model initializes variables to a default value of zero or FALSE. To alleviate this problem additional circuitry was necessary in a number of the test cases. This circuitry provided for the correct operation of the design on the CHS2x4, but was not necessary for the correct operation of the occam model.
CHAPTER VI

PERFORMANCE COMPARISONS

FPGA based systems are commonly used to implement a variety of computing devices ranging from peripheral processors to experimental custom architecture computers. The FPGA system examined here, the CHS2x4 Custom Computer was designed to perform a variety of computation and design functions, but for the purpose of performance comparisons will be utilized as a peripheral processor for a PC. This section compares the performance of the CHS2x4 and a Mobius 486DX2 PC running at 66Mhz as both execute a selection of computing algorithms. The goal here is to reveal the possible advantages and disadvantages in using FPGAs as computing devices.

Bit Serial Adder Performance Comparisons

The bit serial adder algorithm shown earlier in Figure 33 represents a compact, efficient circuit that is capable of adding 8 integer numbers of any length rather quickly. By comparison, a 486 PC probably uses some parallel combination of carry save adders and carry propagate adders for integer operations.

The bit serial adder implemented on the CHS2x4 used 7 clock cycles to flush and initialize the latches and 3 cycles to fill the pipeline before valid data arrived at the output port. Subsequent valid data bits then arrived one every clock cycle. This allowed the CHS2x4 to add 8 n-bit numbers in n+10 clock cycles. Assuming 32-bit numbers and a clock period of 2.5 µs, the CHS2x4 performed one 32-bit integer
addition every 26.25 µs. As the size of the integers increases, this value increases linearly, e.g. 64-bit integers require 46.25 µs, 128-bit integers require 86.25 µs. Assuming that 4, 32-bit integers can be represented by an 128-bit integer, the time per 32-bit addition eventually approaches 20 µs.

Theoretically, the bit serial adder algorithm scales at a rate of $\log_2 x$, where $x$ is the number of values to be added. This produces a total addition cycles of $n + 7 + \log_2 x$ for adding $x n$-bit numbers. As an example, an adder with $x = 32$ could easily be implemented on one CAL1024 chip. The circuit's clock speed would then only be limited by the maximum propagation delay of one of the adder's stages. A stage's maximum delay path is calculated using the computation and routing timings given in the CAL1024 datasheets and is under 20ns. This computes to a maximum operating clock frequency of 50 Mhz and a throughput of 1,136,363, 32-bit integer adds per second.

Though the FPGA based systems' addition speeds are substantial, they do not possess the luxury of the custom addition logic found in the PC's ALU. The speed of this logic was found by using the simple addition loop shown in Appendix J. This program produced timing results that indicated the PC's average addition time per 32-bit addition was approximately 0.343 µs. To compete with this kind of speed, the CHS2x4 bit serial adder configuration would need to accommodate nearly 350 values at one time, thereby necessitating an extremely large and perhaps unrealizable array. Even the theoretical FPGA system would require a significantly larger and perhaps unrealistic array. A detailed summary of these figures is shown in Table 13.
The (massively) parallel sorting algorithm implemented earlier during accuracy verification was also used for performance comparison purposes. This sorting configuration realized only a very small scale representation of the parallel sorter, yet it did supply vital evidence that this class of algorithms was possible in FPGA systems.

The parallel sorter example sorted three numbers during one Local Transfer instruction (once the pipeline was flushed and then filled). For larger scale sorters, the success of this design will be limited due to the increasing propagation delays. This will necessitate a slightly modified design that incorporates latches between successive sorter stages. With the additional latches, the design shown earlier in Figure 32 can be easily scaled by adding additional stages with little fear of data loss. The speed of this design stems mainly from its use of many (hundreds or thousands) one-bit processing elements. Though this particular design only implements a few dozen elements, its concepts are easily extended to larger scale systems.

Initially, this sorter required an initial clearing period to preset the latch values to a known state. This can generally be represented by $8 + x$ cycles where the 8 cycles represent the delay time associated with filling the pipeline and $x$ represents one cycle for each stage in the sorter. Similarly, the input values that followed were sorted at a rate of 1 per cycle with 8 cycles used to fill the pipeline and an additional $x - 1$ cycles incurred in flushing the sorted values out of the sorter. This results in $x$ 8-bit numbers being sorted in approximately $(8 + x) + (8 + 2x - 1)$ or $15 + 3x$ cycles. For larger bit numbers, the total cycles to sort $x$ $n$-bit numbers is $2n - 1 + 3x$. As $x$ increases and $n$ remains constant, the time per value sorted approaches 3 cycles or 7.5 $\mu$s assuming a 2.5 $\mu$s cycle period.
These figures, however, are significantly restricted by the CHS2x4's low instruction issue rate. By assuming a theoretical issue rate based on the design's maximum propagation delay, a better estimate of the CAL1024 chip's performance can be obtained. The parallel sorter's maximum propagation delay per stage was calculated to be approximately 30ns which assumes a computation delay of 7ns and a routing delay of 1.7ns per cell. With this higher clock frequency, the algorithm's speed approaches 1 sort every 90ns or over 1.1 million 8-bit sorts per second; substantially more than its current configuration.

The theoretical and experimental performance figures were then compared to the performance of the 486 66Mhz PC using the bubble sort program shown in Appendix K. With this algorithm, the PC sorted 4096 8-bit values at an average speed of 700.41 µs per sort - over 100 times slower than that of the CHS2x4. Unlike before, the PC's microprocessor does not contain custom logic to accommodate sorting. Instead, it must repeatedly load two values, compare them, and swap if necessary - doing this a total of 2x - 1 times for x numbers. The tremendous amount of memory accesses associated with all of these operations virtually eliminates any device speed superiority that exists in the PC while favoring the systolic FPGA designs.

Neural Network Example and Performance Comparisons

The final example implemented on the CHS2x4 and PC was that of the small neural network shown in Figure 35. The architecture of these networks is such that they require extensive circuitry to accomplish the multiplication of input and weight values. By incorporating the stochastic architecture described in [Daalen93], the neural network's heavy reliance on multiplication is transferred to the use of simple XOR gates. The primary portion of this architecture that incorporates dynamically
reconfigurable FPGAs deals with the generation of stochastic bit streams. These bit streams are the transformed real values that are fed into the XOR gate. The exclusive-ORing of the stochastic bit streams equates to the multiplication of the corresponding real values. It is the job of FPGAs to create the hardware necessary to generate the required bit streams. The example demonstrated in this section creates not only the bit stream hardware but also the remaining neural circuitry on the CHS2x4.

Figure 35. Stochastic Neural Network.

The bit stream or modulation hardware, shown in Figures 36-38, includes only three basic components: a bit 1 modulator element, a bit 0 modulator element, and a pseudo-random bit sequence (PRBS) generator. Any particular bit stream generator is comprised of a series of these bit 1 or bit 0 modulators. The length of the modulation series and the pattern of modulators determine the stream's resolution and probability, respectively. These values are directly influenced by the bit stream's real value counterpart as each bit stream corresponds to specific real value, i.e. each input value and weight must be converted to a separate stochastic bit stream. This is where the
use dynamically reconfigurable FPGAs becomes paramount - the input values (and perhaps the weights) to successive layers in the neural network are dynamic. This means that the required "hardware" bit stream modulators can not efficiently be constructed with fixed logic while the use of FPGAs allows the dynamic reconfiguration necessary to produce the required bit streams.

Figure 36. Bit 0 Modulator.

Figure 37. Bit 1 Modulator.

By using stochastic bit streams, the neural architecture can then employ a rather simple neuron. This neuron, shown in Figure 39, differs slightly from the one in [Daalen93] in that it multiplexes the threshold and weighted input channels. The neuron's primary circuitry is a simple counter that is preset to a value representing some threshold. The threshold is set in such a manner as to overflow the counter when enough 1's have been counted. When all the input bits have been counted, the neuron's output is obtained from checking the counter's overflow.
The CHS2x4 implementation of this architecture required the use of multiple stages to configure and compute the various bit streams and a final configuration to sum their values and determine and output. Each stage consisted of the modulation hardware and the PRBS shown earlier in Figure 38. The exact composition of the modulation hardware was determined arbitrarily with no concern for the corresponding real values. The PRBS contained 8 flip-flops with taps at positions 4, 5, and 6 for a maximal "random" sequence length of 255. For each stage, five arbitrary 4-bit stream generators and one PRBS were combined. The resulting four stages appeared similar to stage 1 which is shown in Figure 39. The five generators contained in each stage represent $I_j$ and $W_{nj}$ - where $n$ ranges from 1 to the number of neurons per layer (4 in this case), $I_j$ represents input value $j$, and $W_{nj}$ represents neuron $n$'s weight for $I_j$. Each stage was configured and then clocked to generate a total of 16 bit streams. The bit streams were then fed into the one layered neural network shown earlier in Figure 35. The neurons in this network were preset to
contain arbitrary threshold values by shifting bits through each neuron's weight inputs with Set held high. At the completion of the 16 input streams (or perhaps earlier) each neuron produced an output value. The 'C' control code and a sample output are listed in Appendices L and M, respectively.

![Stochastic Neuron](image)

Figure 39. Stochastic Neuron.

Though this limited realization of a stochastic neural architecture does not fully demonstrate the usefulness of dynamically reconfigurable FPGAs, it does create another significant avenue for further investigation. The performance side of this issue still favors the custom logic circuits, but again may yield to the architecture's scalability and efficiency. For this particular example, the 4, layer 1 neurons performed the equivalent of 16 4-bit multiplications and 12 4-bit additions using 40 local transfer operations. This includes 16 local transfer operations to generate the bit streams (4 stages, 4 bits/stage), 8 (4+4) operations to clear and set the threshold values, and 16 operations to feed the bit streams through the neurons (4, 4-bit streams). Using 32-bit values this value becomes approximately 320 local transfer operations or 800µs. Compare this to a PC running the simple multiply and sum routine shown in Appendix N at a rate of 17.6µs for similar multiply and add operations, and you will notice a significant disparity.
As before, the example demonstrated here incurs many shortcomings due to the architecture of the CHS2x4 though this is not the main concern of this example. Instead, particular attention should be paid to the efficiency achievable by incorporating FPGAs into an architecture. With this example, the need for parallel multipliers and adders in each neuron is eliminated, thus allowing more space for additional neurons and significantly reducing the cost of such devices.

Performance Comparisons Summary and Conclusions

The three examples explored in this thesis demonstrated a number of shortcomings and advantages to using FPGA based systems. It was shown through
the bit serial adder example that FPGAs systems can not match the speed of custom logic. On the other hand, because of the diversity of their functionality, FPGAs can produce semi-custom logic that can exceed the performance of existing non-custom hardware, e.g. the parallel sorter where the PC's processor was poorly suited for sorting. The third example did not primarily deal with possible speed advantages, but instead it explored the use of FPGAs to minimize or reduce the silicon required to realize a neural network. The reduction of silicon was due primarily to the use of stochastic bit streams which were most efficiently realized through the use with FPGAs.

Table 13

Summary of Performance Comparisons Results

<table>
<thead>
<tr>
<th>Platform</th>
<th>32-bit Additions/sec</th>
<th>8-bit Sorts/sec</th>
<th>32-bit Neuron Operations/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHS2x4</td>
<td>38,095</td>
<td>133,333</td>
<td>1,250</td>
</tr>
<tr>
<td>486 66Mhz PC</td>
<td>2,915,541</td>
<td>1,428</td>
<td>56,818</td>
</tr>
<tr>
<td>Theoretical FPGA System</td>
<td>1,136,363</td>
<td>1,111,111</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Taking all of this into consideration, a designer must decide on the appropriate speed, size, and cost tradeoffs that are required of the desired system. An FPGA based design can offer relatively high performance while requiring larger, expensive arrays, yet it can also offer the same high performance with a much smaller, inexpensive array. With an adequately sized array for the particular algorithm, an
FPGA based system can provide reconfigurable semi-custom logic that approaches custom logic speeds.
CHAPTER VII

CONCLUSIONS

The behavioral model developed in this thesis employed the occam language to describe the Algotronix CHS2x4 Custom Computer. The use of occam allowed the individual components of the CHS2x4 along with its controlling software to be modeled with communicating sequential processes. The individual component procedures were then executed concurrently to create the behavioral model of the CHS2x4.

To verify that this model reproduced the behavior of the CHS2x4, a number of test cases and verification procedures were used. The first verification procedure succeeded in verifying the model's stability and cellular functionality through iterative execution of every possible cell function. Each configuration performed its intended function within an acceptable time range. The second verification procedure verified the model's various routing paths. Data was successfully routed through the four basic routing paths within an acceptable time range and completely intact.

After the cell functionality and routing verifications, the operation of the entire model was examined. The simplest and most efficient means to accomplish this was through the use of actual CHS2x4 configurations. The results of a seven segment decoder, bit serial adder, and a parallel sorter were obtained from the CHS2x4 and the model. Comparisons of this data revealed an exact correlation between the model's and CHS2x4's results in both functionality and timing. In addition to the model verification, this thesis also explored the relative speeds of a 486DX2 66Mhz PC and
an FPGA based system executing a variety of algorithms. The results of these comparisons indicated that an FPGA based system configured for sorting can exceed the performance of a PC running a simple bubble sort. Conversely, results were also obtained that indicated the same is not true for integer addition. This implies that FPGA based system speeds can exceed those of a PC, but only when the desired operation is not directly implemented by the PC with custom logic.

The simulation and verification results demonstrated that the occam language can satisfactorily be employed to create a behavioral model. The language's ability to create communicating sequential processes permitted all of the system's components, hardware and software, to be modeled on a single, homogeneous platform. Once such a platform was obtained, system parameters (i.e. system configuration, system timings, routing delays, configuration delays, etc.) were easily modified, compiled, and executed - quickly yielding new system characteristics.

In addition to the many positive contributions that occam supplied, it was found that occam added a number of inconveniences and unwanted assumptions. All of which stemmed from the implied structure of the occam communication channels. By limiting the communication links to uni-directional, one-way channels, the language forced the inefficient implementation of buses and global signals. As an example, consider the global clocking lines G1 and G2 that are routed to every cell in the CHS2x4's array. The implementation of these lines in occam necessitated creating a private link from the array to each cell - that is 16k of links used to model one wire. This is not to say the communication links are a detriment to the occam programming language, but rather a constraint. They provide efficient implementation for the majority of communications, process synchronization, and greatly aid in debugging. What is apparent from this is that the implied structural implementations found in
occam provided both a positive and negative impact on the resulting (pseudo-behavioral) model.

In an effort to alleviate some its inefficiencies, occam could be modified to provide global signals and bus communication. By presetting a semaphore to represent the number of processes synchronizing on a global signal, each receiving process can decrement the semaphore. When the semaphore reaches zero, the source process knows all communications have completed. Similarly, bus communication could use a form of memory-mapped I/O and encode the ready signal to select the signal's destination. Once the appropriate destination is located, communication occurs normally.

Both of the modifications suggested above imply an underlying structural implementation much the same way as occam does with its communication links. It is this characteristic that prevents occam from being a true behavioral languages, and places it in a gray area between structural and behavioral languages. Nevertheless, occam's proven ability as a programming language and its ability as a behavioral language allows virtually any system design to be explored with an arbitrary hardware/software implementation. It is this ability that has made behavioral languages an invaluable tool in hardware/software codesign.
Appendix A

Occam Model System Constants and Definitions
#INCLUDE "hostio.inc"
#USE "hostio.lib"

-- global constants
VAL RAMSIZE IS INT 2097152 ; -- 2 Megabytes
VAL TERMINATE IS 255 :
VAL RAM.BASE.ADDRESS IS INT #00200000 :
VAL CONTROL.CAL.BASE.ADDRESS IS INT #00020000 :
VAL north IS #00 :
VAL south IS #01 :
VAL east IS #02 :
VAL west IS #03 :
VAL f.out IS #04 :
VAL gl.val IS #05 :
VAL g2.val IS #06 :
VAL COMPUTE.DELAY IS 3 :
   -- approx cell prop. delay is 1.7 ns
   -- approx comput. delay is 7 ns
   -- from CHS2x4 timings the ratio of instruction issues to
   -- cell propagations is approx. 1:1470
   -- data will be valid for 50% of this time (735)
VAL BUS.DELAY IS 1470 :
VAL DATA.VALID IS 735 :
VAL XMAX IS 128 :
VAL YMAX IS 64 :
VAL XMASK IS #7F ; -- 7 bits used for x to mask off address
VAL YSHIFT IS 7 ; -- 7 bit shift to remove x
PROTOCOL DISPLAY IS INT; INT; INT :
Appendix B

Cell Verification Data
Table 14  
Cell Functions 0, 1 and 2 Verification Data

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Function 0 Execution Time</th>
<th>Function 1 Execution Time</th>
<th>Function X1 Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000 232032</td>
<td>1111 231557</td>
<td>0100 231243</td>
</tr>
<tr>
<td>2</td>
<td>0000 232810</td>
<td>1111 233123</td>
<td>0100 233898</td>
</tr>
<tr>
<td>3</td>
<td>0000 232967</td>
<td>1111 232489</td>
<td>0100 231865</td>
</tr>
<tr>
<td>4</td>
<td>0000 232186</td>
<td>1111 231708</td>
<td>0100 231705</td>
</tr>
<tr>
<td>5</td>
<td>0000 231561</td>
<td>1111 231868</td>
<td>0100 231865</td>
</tr>
<tr>
<td>6</td>
<td>0000 248752</td>
<td>1111 231244</td>
<td>0100 240460</td>
</tr>
<tr>
<td>7</td>
<td>0000 246410</td>
<td>1111 232020</td>
<td>0100 231708</td>
</tr>
<tr>
<td>8</td>
<td>0000 232167</td>
<td>1111 232803</td>
<td>0100 231707</td>
</tr>
<tr>
<td>9</td>
<td>0000 231874</td>
<td>1111 231875</td>
<td>0100 231563</td>
</tr>
<tr>
<td>10</td>
<td>0000 232188</td>
<td>1111 231559</td>
<td>0100 231556</td>
</tr>
<tr>
<td>11</td>
<td>0000 242972</td>
<td>1111 231746</td>
<td>0100 231096</td>
</tr>
<tr>
<td>12</td>
<td>0000 232342</td>
<td>1111 234839</td>
<td>0100 243896</td>
</tr>
<tr>
<td>13</td>
<td>0000 231407</td>
<td>1111 231716</td>
<td>0100 231869</td>
</tr>
<tr>
<td>14</td>
<td>0000 231246</td>
<td>1111 232331</td>
<td>0100 231392</td>
</tr>
<tr>
<td>15</td>
<td>0000 231565</td>
<td>1111 231862</td>
<td>0100 231394</td>
</tr>
<tr>
<td>16</td>
<td>0000 231408</td>
<td>1111 232497</td>
<td>0100 231556</td>
</tr>
<tr>
<td>17</td>
<td>0000 231782</td>
<td>1111 240462</td>
<td>0100 232017</td>
</tr>
<tr>
<td>18</td>
<td>0000 231251</td>
<td>1111 231866</td>
<td>0100 232801</td>
</tr>
<tr>
<td>19</td>
<td>0000 231402</td>
<td>1111 232028</td>
<td>0100 231247</td>
</tr>
<tr>
<td>20</td>
<td>0000 232035</td>
<td>1111 231555</td>
<td>0100 231406</td>
</tr>
<tr>
<td>21</td>
<td>0000 233755</td>
<td>1111 239839</td>
<td>0100 232181</td>
</tr>
<tr>
<td>22</td>
<td>0000 232038</td>
<td>1111 231862</td>
<td>0100 231864</td>
</tr>
<tr>
<td>23</td>
<td>0000 232813</td>
<td>1111 232493</td>
<td>0100 231550</td>
</tr>
<tr>
<td>24</td>
<td>0000 232878</td>
<td>1111 231872</td>
<td>0100 232332</td>
</tr>
<tr>
<td>25</td>
<td>0000 232346</td>
<td>1111 231555</td>
<td>0100 232799</td>
</tr>
<tr>
<td>26</td>
<td>0000 231922</td>
<td>1111 231248</td>
<td>0100 231136</td>
</tr>
<tr>
<td>27</td>
<td>0000 232656</td>
<td>1111 232178</td>
<td>0100 231530</td>
</tr>
<tr>
<td>28</td>
<td>0000 232031</td>
<td>1111 231874</td>
<td>0100 231870</td>
</tr>
<tr>
<td>29</td>
<td>0000 231719</td>
<td>1111 232343</td>
<td>0100 231555</td>
</tr>
<tr>
<td>30</td>
<td>0000 232187</td>
<td>1111 232501</td>
<td>0100 232184</td>
</tr>
<tr>
<td>31</td>
<td>0000 237817</td>
<td>1111 254216</td>
<td>0100 232189</td>
</tr>
<tr>
<td>32</td>
<td>0000 231876</td>
<td>1111 231678</td>
<td>0100 232347</td>
</tr>
<tr>
<td>33</td>
<td>0000 231404</td>
<td>1111 232189</td>
<td>0100 232188</td>
</tr>
<tr>
<td>34</td>
<td>0000 231719</td>
<td>1111 232034</td>
<td>0100 232314</td>
</tr>
<tr>
<td>35</td>
<td>0000 235934</td>
<td>1111 234160</td>
<td>0100 237487</td>
</tr>
<tr>
<td>36</td>
<td>0000 231717</td>
<td>1111 231877</td>
<td>0100 232032</td>
</tr>
<tr>
<td>37</td>
<td>0000 231872</td>
<td>1111 231715</td>
<td>0100 231563</td>
</tr>
<tr>
<td>38</td>
<td>0000 231824</td>
<td>1111 237016</td>
<td>0100 232179</td>
</tr>
<tr>
<td>39</td>
<td>0000 231400</td>
<td>1111 232338</td>
<td>0100 232012</td>
</tr>
<tr>
<td>40</td>
<td>0000 234375</td>
<td>1111 231879</td>
<td>0100 233283</td>
</tr>
<tr>
<td>41</td>
<td>0000 244375</td>
<td>1111 236591</td>
<td>0100 233907</td>
</tr>
<tr>
<td>42</td>
<td>0000 236875</td>
<td>1111 233893</td>
<td>0100 235613</td>
</tr>
<tr>
<td>43</td>
<td>0000 233750</td>
<td>1111 231251</td>
<td>0100 237345</td>
</tr>
<tr>
<td>44</td>
<td>0000 232967</td>
<td>1111 233111</td>
<td>0100 235305</td>
</tr>
<tr>
<td>45</td>
<td>0000 235157</td>
<td>1111 233752</td>
<td>0100 235461</td>
</tr>
<tr>
<td>46</td>
<td>0000 231873</td>
<td>1111 233749</td>
<td>0100 231207</td>
</tr>
<tr>
<td>47</td>
<td>0000 234375</td>
<td>1111 234061</td>
<td>0100 233899</td>
</tr>
<tr>
<td>48</td>
<td>0000 235625</td>
<td>1111 235619</td>
<td>0100 235158</td>
</tr>
<tr>
<td>49</td>
<td>0000 234057</td>
<td>1111 234215</td>
<td>0100 236251</td>
</tr>
<tr>
<td>50</td>
<td>0000 234063</td>
<td>1111 231991</td>
<td>0100 234369</td>
</tr>
</tbody>
</table>
### Table 15
Cell Functions 3, 4 and 5 Verification Data

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Function X1BAR</th>
<th>Execution Time</th>
<th>Function X2</th>
<th>Execution Time</th>
<th>Function X2BAR</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1011</td>
<td>236812</td>
<td>0100</td>
<td>231555</td>
<td>1011</td>
<td>232821</td>
</tr>
<tr>
<td>2</td>
<td>1011</td>
<td>232499</td>
<td>0100</td>
<td>231558</td>
<td>1011</td>
<td>232028</td>
</tr>
<tr>
<td>3</td>
<td>1011</td>
<td>231246</td>
<td>0100</td>
<td>231905</td>
<td>1011</td>
<td>236732</td>
</tr>
<tr>
<td>4</td>
<td>1011</td>
<td>231831</td>
<td>0100</td>
<td>232146</td>
<td>1011</td>
<td>231865</td>
</tr>
<tr>
<td>5</td>
<td>1011</td>
<td>231243</td>
<td>0100</td>
<td>232300</td>
<td>1011</td>
<td>231866</td>
</tr>
<tr>
<td>6</td>
<td>1011</td>
<td>259212</td>
<td>0100</td>
<td>245967</td>
<td>1011</td>
<td>234196</td>
</tr>
<tr>
<td>7</td>
<td>1011</td>
<td>232652</td>
<td>0100</td>
<td>232021</td>
<td>1011</td>
<td>231405</td>
</tr>
<tr>
<td>8</td>
<td>1011</td>
<td>231118</td>
<td>0100</td>
<td>245401</td>
<td>1011</td>
<td>231709</td>
</tr>
<tr>
<td>9</td>
<td>1011</td>
<td>232181</td>
<td>0100</td>
<td>231712</td>
<td>1011</td>
<td>232031</td>
</tr>
<tr>
<td>10</td>
<td>1011</td>
<td>232030</td>
<td>0100</td>
<td>231406</td>
<td>1011</td>
<td>231403</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>231093</td>
<td>0100</td>
<td>231404</td>
<td>1011</td>
<td>231261</td>
</tr>
<tr>
<td>12</td>
<td>1011</td>
<td>234064</td>
<td>0100</td>
<td>233277</td>
<td>1011</td>
<td>232332</td>
</tr>
<tr>
<td>13</td>
<td>1011</td>
<td>231255</td>
<td>0100</td>
<td>231709</td>
<td>1011</td>
<td>231562</td>
</tr>
<tr>
<td>14</td>
<td>1011</td>
<td>231120</td>
<td>0100</td>
<td>232340</td>
<td>1011</td>
<td>232492</td>
</tr>
<tr>
<td>15</td>
<td>1011</td>
<td>231866</td>
<td>0100</td>
<td>232155</td>
<td>1011</td>
<td>232333</td>
</tr>
<tr>
<td>16</td>
<td>1011</td>
<td>234207</td>
<td>0100</td>
<td>256397</td>
<td>1011</td>
<td>243741</td>
</tr>
<tr>
<td>17</td>
<td>1011</td>
<td>232505</td>
<td>0100</td>
<td>231713</td>
<td>1011</td>
<td>232180</td>
</tr>
<tr>
<td>18</td>
<td>1011</td>
<td>232180</td>
<td>0100</td>
<td>232329</td>
<td>1011</td>
<td>232180</td>
</tr>
<tr>
<td>19</td>
<td>1011</td>
<td>232964</td>
<td>0100</td>
<td>232336</td>
<td>1011</td>
<td>232030</td>
</tr>
<tr>
<td>20</td>
<td>1011</td>
<td>232439</td>
<td>0100</td>
<td>231554</td>
<td>1011</td>
<td>231407</td>
</tr>
<tr>
<td>21</td>
<td>1011</td>
<td>254052</td>
<td>0100</td>
<td>232020</td>
<td>1011</td>
<td>232651</td>
</tr>
<tr>
<td>22</td>
<td>1011</td>
<td>232179</td>
<td>0100</td>
<td>231991</td>
<td>1011</td>
<td>232282</td>
</tr>
<tr>
<td>23</td>
<td>1011</td>
<td>232190</td>
<td>0100</td>
<td>232436</td>
<td>1011</td>
<td>231868</td>
</tr>
<tr>
<td>24</td>
<td>1011</td>
<td>231403</td>
<td>0100</td>
<td>231676</td>
<td>1011</td>
<td>232346</td>
</tr>
<tr>
<td>25</td>
<td>1011</td>
<td>231244</td>
<td>0100</td>
<td>232464</td>
<td>1011</td>
<td>231870</td>
</tr>
<tr>
<td>26</td>
<td>1011</td>
<td>231370</td>
<td>0100</td>
<td>231845</td>
<td>1011</td>
<td>231874</td>
</tr>
<tr>
<td>27</td>
<td>1011</td>
<td>232500</td>
<td>0100</td>
<td>233586</td>
<td>1011</td>
<td>232392</td>
</tr>
<tr>
<td>28</td>
<td>1011</td>
<td>233753</td>
<td>0100</td>
<td>231557</td>
<td>1011</td>
<td>232338</td>
</tr>
<tr>
<td>29</td>
<td>1011</td>
<td>232500</td>
<td>0100</td>
<td>231868</td>
<td>1011</td>
<td>231722</td>
</tr>
<tr>
<td>30</td>
<td>1011</td>
<td>232657</td>
<td>0100</td>
<td>232339</td>
<td>1011</td>
<td>231874</td>
</tr>
<tr>
<td>31</td>
<td>1011</td>
<td>252491</td>
<td>0100</td>
<td>233414</td>
<td>1011</td>
<td>231554</td>
</tr>
<tr>
<td>32</td>
<td>1011</td>
<td>232035</td>
<td>0100</td>
<td>232170</td>
<td>1011</td>
<td>232099</td>
</tr>
<tr>
<td>33</td>
<td>1011</td>
<td>231562</td>
<td>0100</td>
<td>231720</td>
<td>1011</td>
<td>232147</td>
</tr>
<tr>
<td>34</td>
<td>1011</td>
<td>231886</td>
<td>0100</td>
<td>232176</td>
<td>1011</td>
<td>231869</td>
</tr>
<tr>
<td>35</td>
<td>1011</td>
<td>234277</td>
<td>0100</td>
<td>235620</td>
<td>1011</td>
<td>235624</td>
</tr>
<tr>
<td>36</td>
<td>1011</td>
<td>232034</td>
<td>0100</td>
<td>232186</td>
<td>1011</td>
<td>231565</td>
</tr>
<tr>
<td>37</td>
<td>1011</td>
<td>232030</td>
<td>0100</td>
<td>231409</td>
<td>1011</td>
<td>231562</td>
</tr>
<tr>
<td>38</td>
<td>1011</td>
<td>232960</td>
<td>0100</td>
<td>231868</td>
<td>1011</td>
<td>232024</td>
</tr>
<tr>
<td>39</td>
<td>1011</td>
<td>232195</td>
<td>0100</td>
<td>231867</td>
<td>1011</td>
<td>232333</td>
</tr>
<tr>
<td>40</td>
<td>1011</td>
<td>235465</td>
<td>0100</td>
<td>235302</td>
<td>1011</td>
<td>232185</td>
</tr>
<tr>
<td>41</td>
<td>1011</td>
<td>231404</td>
<td>0100</td>
<td>234982</td>
<td>1011</td>
<td>250770</td>
</tr>
<tr>
<td>42</td>
<td>1011</td>
<td>234065</td>
<td>0100</td>
<td>231866</td>
<td>1011</td>
<td>236410</td>
</tr>
<tr>
<td>43</td>
<td>1011</td>
<td>234516</td>
<td>0100</td>
<td>233417</td>
<td>1011</td>
<td>234688</td>
</tr>
<tr>
<td>44</td>
<td>1011</td>
<td>235925</td>
<td>0100</td>
<td>236399</td>
<td>1011</td>
<td>233430</td>
</tr>
<tr>
<td>45</td>
<td>1011</td>
<td>234064</td>
<td>0100</td>
<td>234080</td>
<td>1011</td>
<td>232503</td>
</tr>
<tr>
<td>46</td>
<td>1011</td>
<td>231401</td>
<td>0100</td>
<td>231861</td>
<td>1011</td>
<td>232506</td>
</tr>
<tr>
<td>47</td>
<td>1011</td>
<td>234049</td>
<td>0100</td>
<td>235461</td>
<td>1011</td>
<td>231833</td>
</tr>
<tr>
<td>48</td>
<td>1011</td>
<td>235306</td>
<td>0100</td>
<td>231248</td>
<td>1011</td>
<td>234718</td>
</tr>
<tr>
<td>49</td>
<td>1011</td>
<td>234065</td>
<td>0100</td>
<td>233740</td>
<td>1011</td>
<td>235160</td>
</tr>
<tr>
<td>50</td>
<td>1011</td>
<td>233751</td>
<td>0100</td>
<td>232656</td>
<td>1011</td>
<td>235609</td>
</tr>
<tr>
<td>Iteration</td>
<td>Function AND</td>
<td>Execution Time</td>
<td>Function X1X2BAR</td>
<td>Execution Time</td>
<td>Function X1BARX2</td>
<td>Execution Time</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>----------------</td>
<td>------------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>1</td>
<td>0100</td>
<td>232812</td>
<td>0000</td>
<td>231242</td>
<td>0000</td>
<td>231871</td>
</tr>
<tr>
<td>2</td>
<td>0100</td>
<td>232499</td>
<td>0000</td>
<td>232025</td>
<td>0000</td>
<td>232028</td>
</tr>
<tr>
<td>3</td>
<td>0100</td>
<td>231719</td>
<td>0000</td>
<td>231718</td>
<td>0000</td>
<td>231093</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>231553</td>
<td>0000</td>
<td>232328</td>
<td>0000</td>
<td>235853</td>
</tr>
<tr>
<td>5</td>
<td>0100</td>
<td>232807</td>
<td>0000</td>
<td>233425</td>
<td>0000</td>
<td>232179</td>
</tr>
<tr>
<td>6</td>
<td>0100</td>
<td>231549</td>
<td>0000</td>
<td>231865</td>
<td>0000</td>
<td>232330</td>
</tr>
<tr>
<td>7</td>
<td>0100</td>
<td>231558</td>
<td>0000</td>
<td>232015</td>
<td>0000</td>
<td>231555</td>
</tr>
<tr>
<td>8</td>
<td>0100</td>
<td>233119</td>
<td>0000</td>
<td>232495</td>
<td>0000</td>
<td>232179</td>
</tr>
<tr>
<td>9</td>
<td>0100</td>
<td>231404</td>
<td>0000</td>
<td>232871</td>
<td>0000</td>
<td>231696</td>
</tr>
<tr>
<td>10</td>
<td>0100</td>
<td>232814</td>
<td>0000</td>
<td>231667</td>
<td>0000</td>
<td>232192</td>
</tr>
<tr>
<td>11</td>
<td>0100</td>
<td>231719</td>
<td>0000</td>
<td>231243</td>
<td>0000</td>
<td>231873</td>
</tr>
<tr>
<td>12</td>
<td>0100</td>
<td>232345</td>
<td>0000</td>
<td>231243</td>
<td>0000</td>
<td>232501</td>
</tr>
<tr>
<td>13</td>
<td>0100</td>
<td>232188</td>
<td>0000</td>
<td>231556</td>
<td>0000</td>
<td>231876</td>
</tr>
<tr>
<td>14</td>
<td>0100</td>
<td>231884</td>
<td>0000</td>
<td>232180</td>
<td>0000</td>
<td>232184</td>
</tr>
<tr>
<td>15</td>
<td>0100</td>
<td>232491</td>
<td>0000</td>
<td>232719</td>
<td>0000</td>
<td>233885</td>
</tr>
<tr>
<td>16</td>
<td>0100</td>
<td>240772</td>
<td>0000</td>
<td>251397</td>
<td>0000</td>
<td>231704</td>
</tr>
<tr>
<td>17</td>
<td>0100</td>
<td>232648</td>
<td>0000</td>
<td>231860</td>
<td>0000</td>
<td>231709</td>
</tr>
<tr>
<td>18</td>
<td>0100</td>
<td>231714</td>
<td>0000</td>
<td>232801</td>
<td>0000</td>
<td>232179</td>
</tr>
<tr>
<td>19</td>
<td>0100</td>
<td>231777</td>
<td>0000</td>
<td>231564</td>
<td>0000</td>
<td>231129</td>
</tr>
<tr>
<td>20</td>
<td>0100</td>
<td>232967</td>
<td>0000</td>
<td>233125</td>
<td>0000</td>
<td>231565</td>
</tr>
<tr>
<td>21</td>
<td>0100</td>
<td>254990</td>
<td>0000</td>
<td>246551</td>
<td>0000</td>
<td>231219</td>
</tr>
<tr>
<td>22</td>
<td>0100</td>
<td>231876</td>
<td>0000</td>
<td>235600</td>
<td>0000</td>
<td>233273</td>
</tr>
<tr>
<td>23</td>
<td>0100</td>
<td>232966</td>
<td>0000</td>
<td>231860</td>
<td>0000</td>
<td>231869</td>
</tr>
<tr>
<td>24</td>
<td>0100</td>
<td>231070</td>
<td>0000</td>
<td>231717</td>
<td>0000</td>
<td>232490</td>
</tr>
<tr>
<td>25</td>
<td>0100</td>
<td>231869</td>
<td>0000</td>
<td>232650</td>
<td>0000</td>
<td>232800</td>
</tr>
<tr>
<td>26</td>
<td>0100</td>
<td>232187</td>
<td>0000</td>
<td>231714</td>
<td>0000</td>
<td>231877</td>
</tr>
<tr>
<td>27</td>
<td>0100</td>
<td>231095</td>
<td>0000</td>
<td>235618</td>
<td>0000</td>
<td>232968</td>
</tr>
<tr>
<td>28</td>
<td>0100</td>
<td>232189</td>
<td>0000</td>
<td>231557</td>
<td>0000</td>
<td>232193</td>
</tr>
<tr>
<td>29</td>
<td>0100</td>
<td>232183</td>
<td>0000</td>
<td>234057</td>
<td>0000</td>
<td>231404</td>
</tr>
<tr>
<td>30</td>
<td>0100</td>
<td>231871</td>
<td>0000</td>
<td>232808</td>
<td>0000</td>
<td>233750</td>
</tr>
<tr>
<td>31</td>
<td>0100</td>
<td>231395</td>
<td>0000</td>
<td>232025</td>
<td>0000</td>
<td>231875</td>
</tr>
<tr>
<td>32</td>
<td>0100</td>
<td>232181</td>
<td>0000</td>
<td>232184</td>
<td>0000</td>
<td>231705</td>
</tr>
<tr>
<td>33</td>
<td>0100</td>
<td>236409</td>
<td>0000</td>
<td>232676</td>
<td>0000</td>
<td>233931</td>
</tr>
<tr>
<td>34</td>
<td>0100</td>
<td>233437</td>
<td>0000</td>
<td>232026</td>
<td>0000</td>
<td>233429</td>
</tr>
<tr>
<td>35</td>
<td>0100</td>
<td>232805</td>
<td>0000</td>
<td>236565</td>
<td>0000</td>
<td>238753</td>
</tr>
<tr>
<td>36</td>
<td>0100</td>
<td>231563</td>
<td>0000</td>
<td>231561</td>
<td>0000</td>
<td>231714</td>
</tr>
<tr>
<td>37</td>
<td>0100</td>
<td>232162</td>
<td>0000</td>
<td>232029</td>
<td>0000</td>
<td>231876</td>
</tr>
<tr>
<td>38</td>
<td>0100</td>
<td>236554</td>
<td>0000</td>
<td>232338</td>
<td>0000</td>
<td>232628</td>
</tr>
<tr>
<td>39</td>
<td>0100</td>
<td>231873</td>
<td>0000</td>
<td>232737</td>
<td>0000</td>
<td>241224</td>
</tr>
<tr>
<td>40</td>
<td>0100</td>
<td>236708</td>
<td>0000</td>
<td>234516</td>
<td>0000</td>
<td>236714</td>
</tr>
<tr>
<td>41</td>
<td>0100</td>
<td>232336</td>
<td>0000</td>
<td>236068</td>
<td>0000</td>
<td>234038</td>
</tr>
<tr>
<td>42</td>
<td>0100</td>
<td>235784</td>
<td>0000</td>
<td>233583</td>
<td>0000</td>
<td>233746</td>
</tr>
<tr>
<td>43</td>
<td>0100</td>
<td>233909</td>
<td>0000</td>
<td>233436</td>
<td>0000</td>
<td>234531</td>
</tr>
<tr>
<td>44</td>
<td>0100</td>
<td>233725</td>
<td>0000</td>
<td>232213</td>
<td>0000</td>
<td>234393</td>
</tr>
<tr>
<td>45</td>
<td>0100</td>
<td>235002</td>
<td>0000</td>
<td>232804</td>
<td>0000</td>
<td>232189</td>
</tr>
<tr>
<td>46</td>
<td>0100</td>
<td>231705</td>
<td>0000</td>
<td>233908</td>
<td>0000</td>
<td>233899</td>
</tr>
<tr>
<td>47</td>
<td>0100</td>
<td>236237</td>
<td>0000</td>
<td>233594</td>
<td>0000</td>
<td>234370</td>
</tr>
<tr>
<td>48</td>
<td>0100</td>
<td>236243</td>
<td>0000</td>
<td>235296</td>
<td>0000</td>
<td>234008</td>
</tr>
<tr>
<td>49</td>
<td>0100</td>
<td>233752</td>
<td>0000</td>
<td>234378</td>
<td>0000</td>
<td>233907</td>
</tr>
<tr>
<td>50</td>
<td>0100</td>
<td>234983</td>
<td>0000</td>
<td>234688</td>
<td>0000</td>
<td>236722</td>
</tr>
<tr>
<td>Iteration</td>
<td>Function NOR</td>
<td>Execution Time</td>
<td>Function OR</td>
<td>Execution Time</td>
<td>Function X1ORX2BAR</td>
<td>Execution Time</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>----------------</td>
<td>-------------</td>
<td>----------------</td>
<td>-------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>1</td>
<td>1011</td>
<td>231555</td>
<td>0100</td>
<td>232028</td>
<td>1111</td>
<td>232180</td>
</tr>
<tr>
<td>2</td>
<td>1011</td>
<td>231868</td>
<td>0100</td>
<td>231868</td>
<td>1111</td>
<td>232186</td>
</tr>
<tr>
<td>3</td>
<td>1011</td>
<td>231712</td>
<td>0100</td>
<td>231713</td>
<td>1111</td>
<td>231876</td>
</tr>
<tr>
<td>4</td>
<td>1011</td>
<td>231206</td>
<td>0100</td>
<td>245613</td>
<td>1111</td>
<td>232807</td>
</tr>
<tr>
<td>5</td>
<td>1011</td>
<td>231704</td>
<td>0100</td>
<td>232808</td>
<td>1111</td>
<td>231920</td>
</tr>
<tr>
<td>6</td>
<td>1011</td>
<td>231707</td>
<td>0100</td>
<td>232021</td>
<td>1111</td>
<td>231553</td>
</tr>
<tr>
<td>7</td>
<td>1011</td>
<td>236259</td>
<td>0100</td>
<td>231808</td>
<td>1111</td>
<td>232495</td>
</tr>
<tr>
<td>8</td>
<td>1011</td>
<td>232644</td>
<td>0100</td>
<td>231869</td>
<td>1111</td>
<td>233428</td>
</tr>
<tr>
<td>9</td>
<td>1011</td>
<td>241546</td>
<td>0100</td>
<td>233125</td>
<td>1111</td>
<td>234521</td>
</tr>
<tr>
<td>10</td>
<td>1011</td>
<td>231556</td>
<td>0100</td>
<td>231405</td>
<td>1111</td>
<td>232341</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>231244</td>
<td>0100</td>
<td>231868</td>
<td>1111</td>
<td>232032</td>
</tr>
<tr>
<td>12</td>
<td>1011</td>
<td>232025</td>
<td>0100</td>
<td>231557</td>
<td>1111</td>
<td>231718</td>
</tr>
<tr>
<td>13</td>
<td>1011</td>
<td>232026</td>
<td>0100</td>
<td>232496</td>
<td>1111</td>
<td>231722</td>
</tr>
<tr>
<td>14</td>
<td>1011</td>
<td>232642</td>
<td>0100</td>
<td>232173</td>
<td>1111</td>
<td>232181</td>
</tr>
<tr>
<td>15</td>
<td>1011</td>
<td>233272</td>
<td>0100</td>
<td>232331</td>
<td>1111</td>
<td>232025</td>
</tr>
<tr>
<td>16</td>
<td>1011</td>
<td>232016</td>
<td>0100</td>
<td>232330</td>
<td>1111</td>
<td>231401</td>
</tr>
<tr>
<td>17</td>
<td>1011</td>
<td>232287</td>
<td>0100</td>
<td>232019</td>
<td>1111</td>
<td>231396</td>
</tr>
<tr>
<td>18</td>
<td>1011</td>
<td>232175</td>
<td>0100</td>
<td>231393</td>
<td>1111</td>
<td>232022</td>
</tr>
<tr>
<td>19</td>
<td>1011</td>
<td>232339</td>
<td>0100</td>
<td>231863</td>
<td>1111</td>
<td>232033</td>
</tr>
<tr>
<td>20</td>
<td>1011</td>
<td>233282</td>
<td>0100</td>
<td>231404</td>
<td>1111</td>
<td>231407</td>
</tr>
<tr>
<td>21</td>
<td>1011</td>
<td>232909</td>
<td>0100</td>
<td>232499</td>
<td>1111</td>
<td>232778</td>
</tr>
<tr>
<td>22</td>
<td>1011</td>
<td>232176</td>
<td>0100</td>
<td>240465</td>
<td>1111</td>
<td>232331</td>
</tr>
<tr>
<td>23</td>
<td>1011</td>
<td>231707</td>
<td>0100</td>
<td>231960</td>
<td>1111</td>
<td>233431</td>
</tr>
<tr>
<td>24</td>
<td>1011</td>
<td>233269</td>
<td>0100</td>
<td>232638</td>
<td>1111</td>
<td>232961</td>
</tr>
<tr>
<td>25</td>
<td>1011</td>
<td>231629</td>
<td>0100</td>
<td>231863</td>
<td>1111</td>
<td>232026</td>
</tr>
<tr>
<td>26</td>
<td>1011</td>
<td>231446</td>
<td>0100</td>
<td>232181</td>
<td>1111</td>
<td>231875</td>
</tr>
<tr>
<td>27</td>
<td>1011</td>
<td>242503</td>
<td>0100</td>
<td>232337</td>
<td>1111</td>
<td>232031</td>
</tr>
<tr>
<td>28</td>
<td>1011</td>
<td>23958</td>
<td>0100</td>
<td>232029</td>
<td>1111</td>
<td>231872</td>
</tr>
<tr>
<td>29</td>
<td>1011</td>
<td>233121</td>
<td>0100</td>
<td>231401</td>
<td>1111</td>
<td>231873</td>
</tr>
<tr>
<td>30</td>
<td>1011</td>
<td>232338</td>
<td>0100</td>
<td>231405</td>
<td>1111</td>
<td>232652</td>
</tr>
<tr>
<td>31</td>
<td>1011</td>
<td>246244</td>
<td>0100</td>
<td>232346</td>
<td>1111</td>
<td>231700</td>
</tr>
<tr>
<td>32</td>
<td>1011</td>
<td>232176</td>
<td>0100</td>
<td>232018</td>
<td>1111</td>
<td>241493</td>
</tr>
<tr>
<td>33</td>
<td>1011</td>
<td>236245</td>
<td>0100</td>
<td>231718</td>
<td>1111</td>
<td>234689</td>
</tr>
<tr>
<td>34</td>
<td>1011</td>
<td>232981</td>
<td>0100</td>
<td>231864</td>
<td>1111</td>
<td>231875</td>
</tr>
<tr>
<td>35</td>
<td>1011</td>
<td>235002</td>
<td>0100</td>
<td>233136</td>
<td>1111</td>
<td>234378</td>
</tr>
<tr>
<td>36</td>
<td>1011</td>
<td>231873</td>
<td>0100</td>
<td>232485</td>
<td>1111</td>
<td>231407</td>
</tr>
<tr>
<td>37</td>
<td>1011</td>
<td>231404</td>
<td>0100</td>
<td>231561</td>
<td>1111</td>
<td>236583</td>
</tr>
<tr>
<td>38</td>
<td>1011</td>
<td>232897</td>
<td>0100</td>
<td>232732</td>
<td>1111</td>
<td>233434</td>
</tr>
<tr>
<td>39</td>
<td>1011</td>
<td>232660</td>
<td>0100</td>
<td>231549</td>
<td>1111</td>
<td>231563</td>
</tr>
<tr>
<td>40</td>
<td>1011</td>
<td>239837</td>
<td>0100</td>
<td>232484</td>
<td>1111</td>
<td>237177</td>
</tr>
<tr>
<td>41</td>
<td>1011</td>
<td>252958</td>
<td>0100</td>
<td>237174</td>
<td>1111</td>
<td>233903</td>
</tr>
<tr>
<td>42</td>
<td>1011</td>
<td>234393</td>
<td>0100</td>
<td>233929</td>
<td>1111</td>
<td>237652</td>
</tr>
<tr>
<td>43</td>
<td>1011</td>
<td>234040</td>
<td>0100</td>
<td>233593</td>
<td>1111</td>
<td>233288</td>
</tr>
<tr>
<td>44</td>
<td>1011</td>
<td>238745</td>
<td>0100</td>
<td>233446</td>
<td>1111</td>
<td>232310</td>
</tr>
<tr>
<td>45</td>
<td>1011</td>
<td>232347</td>
<td>0100</td>
<td>234219</td>
<td>1111</td>
<td>237163</td>
</tr>
<tr>
<td>46</td>
<td>1011</td>
<td>233575</td>
<td>0100</td>
<td>233892</td>
<td>1111</td>
<td>231717</td>
</tr>
<tr>
<td>47</td>
<td>1011</td>
<td>232640</td>
<td>0100</td>
<td>234531</td>
<td>1111</td>
<td>234689</td>
</tr>
<tr>
<td>48</td>
<td>1011</td>
<td>236404</td>
<td>0100</td>
<td>233435</td>
<td>1111</td>
<td>233123</td>
</tr>
<tr>
<td>49</td>
<td>1011</td>
<td>233002</td>
<td>0100</td>
<td>234840</td>
<td>1111</td>
<td>232191</td>
</tr>
<tr>
<td>50</td>
<td>1011</td>
<td>234052</td>
<td>0100</td>
<td>232682</td>
<td>1111</td>
<td>235809</td>
</tr>
<tr>
<td>Iteration</td>
<td>Function Execution XBARORX2 Time</td>
<td>Function Execution XNOR Time</td>
<td>Function Execution NAND Time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1111</td>
<td>231399</td>
<td>1111</td>
<td>231716</td>
<td>1011</td>
<td>231556</td>
</tr>
<tr>
<td>2</td>
<td>1111</td>
<td>231711</td>
<td>1111</td>
<td>232967</td>
<td>1011</td>
<td>231905</td>
</tr>
<tr>
<td>3</td>
<td>1111</td>
<td>232499</td>
<td>1111</td>
<td>232971</td>
<td>1011</td>
<td>232008</td>
</tr>
<tr>
<td>4</td>
<td>1111</td>
<td>232188</td>
<td>1111</td>
<td>231717</td>
<td>1011</td>
<td>232805</td>
</tr>
<tr>
<td>5</td>
<td>1111</td>
<td>231543</td>
<td>1111</td>
<td>232021</td>
<td>1011</td>
<td>232017</td>
</tr>
<tr>
<td>6</td>
<td>1111</td>
<td>231863</td>
<td>1111</td>
<td>232019</td>
<td>1011</td>
<td>232333</td>
</tr>
<tr>
<td>7</td>
<td>1111</td>
<td>232491</td>
<td>1111</td>
<td>232333</td>
<td>1011</td>
<td>232335</td>
</tr>
<tr>
<td>8</td>
<td>1111</td>
<td>234673</td>
<td>1111</td>
<td>232042</td>
<td>1011</td>
<td>231645</td>
</tr>
<tr>
<td>9</td>
<td>1111</td>
<td>231700</td>
<td>1111</td>
<td>231552</td>
<td>1011</td>
<td>232642</td>
</tr>
<tr>
<td>10</td>
<td>1111</td>
<td>232962</td>
<td>1111</td>
<td>232026</td>
<td>1011</td>
<td>231122</td>
</tr>
<tr>
<td>11</td>
<td>1111</td>
<td>232342</td>
<td>1111</td>
<td>232182</td>
<td>1011</td>
<td>231713</td>
</tr>
<tr>
<td>12</td>
<td>1111</td>
<td>232025</td>
<td>1111</td>
<td>232344</td>
<td>1011</td>
<td>231714</td>
</tr>
<tr>
<td>13</td>
<td>1111</td>
<td>231865</td>
<td>1111</td>
<td>231720</td>
<td>1011</td>
<td>232652</td>
</tr>
<tr>
<td>14</td>
<td>1111</td>
<td>233268</td>
<td>1111</td>
<td>231868</td>
<td>1011</td>
<td>232380</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>232645</td>
<td>1111</td>
<td>231710</td>
<td>1011</td>
<td>232333</td>
</tr>
<tr>
<td>16</td>
<td>1111</td>
<td>232799</td>
<td>1111</td>
<td>232341</td>
<td>1011</td>
<td>232022</td>
</tr>
<tr>
<td>17</td>
<td>1111</td>
<td>232641</td>
<td>1111</td>
<td>232023</td>
<td>1011</td>
<td>232810</td>
</tr>
<tr>
<td>18</td>
<td>1111</td>
<td>231858</td>
<td>1111</td>
<td>232334</td>
<td>1011</td>
<td>231392</td>
</tr>
<tr>
<td>19</td>
<td>1111</td>
<td>232024</td>
<td>1111</td>
<td>232309</td>
<td>1011</td>
<td>231874</td>
</tr>
<tr>
<td>20</td>
<td>1111</td>
<td>231875</td>
<td>1111</td>
<td>232032</td>
<td>1011</td>
<td>232032</td>
</tr>
<tr>
<td>21</td>
<td>1111</td>
<td>231860</td>
<td>1111</td>
<td>233234</td>
<td>1011</td>
<td>232262</td>
</tr>
<tr>
<td>22</td>
<td>1111</td>
<td>231388</td>
<td>1111</td>
<td>232960</td>
<td>1011</td>
<td>231392</td>
</tr>
<tr>
<td>23</td>
<td>1111</td>
<td>231285</td>
<td>1111</td>
<td>232498</td>
<td>1011</td>
<td>232020</td>
</tr>
<tr>
<td>24</td>
<td>1111</td>
<td>233268</td>
<td>1111</td>
<td>232180</td>
<td>1011</td>
<td>233234</td>
</tr>
<tr>
<td>25</td>
<td>1111</td>
<td>232020</td>
<td>1111</td>
<td>232340</td>
<td>1011</td>
<td>231977</td>
</tr>
<tr>
<td>26</td>
<td>1111</td>
<td>231404</td>
<td>1111</td>
<td>233899</td>
<td>1011</td>
<td>235153</td>
</tr>
<tr>
<td>27</td>
<td>1111</td>
<td>231709</td>
<td>1111</td>
<td>231719</td>
<td>1011</td>
<td>232496</td>
</tr>
<tr>
<td>28</td>
<td>1111</td>
<td>232182</td>
<td>1111</td>
<td>231563</td>
<td>1011</td>
<td>231713</td>
</tr>
<tr>
<td>29</td>
<td>1111</td>
<td>232337</td>
<td>1111</td>
<td>232187</td>
<td>1011</td>
<td>231865</td>
</tr>
<tr>
<td>30</td>
<td>1111</td>
<td>231251</td>
<td>1111</td>
<td>231959</td>
<td>1011</td>
<td>232339</td>
</tr>
<tr>
<td>31</td>
<td>1111</td>
<td>232344</td>
<td>1111</td>
<td>231870</td>
<td>1011</td>
<td>231879</td>
</tr>
<tr>
<td>32</td>
<td>1111</td>
<td>231871</td>
<td>1111</td>
<td>232493</td>
<td>1011</td>
<td>232433</td>
</tr>
<tr>
<td>33</td>
<td>1111</td>
<td>231561</td>
<td>1111</td>
<td>231554</td>
<td>1011</td>
<td>232190</td>
</tr>
<tr>
<td>34</td>
<td>1111</td>
<td>231871</td>
<td>1111</td>
<td>231856</td>
<td>1011</td>
<td>232816</td>
</tr>
<tr>
<td>35</td>
<td>1111</td>
<td>231576</td>
<td>1111</td>
<td>232025</td>
<td>1011</td>
<td>235145</td>
</tr>
<tr>
<td>36</td>
<td>1111</td>
<td>232030</td>
<td>1111</td>
<td>231865</td>
<td>1011</td>
<td>232032</td>
</tr>
<tr>
<td>37</td>
<td>1111</td>
<td>231560</td>
<td>1111</td>
<td>232334</td>
<td>1011</td>
<td>232902</td>
</tr>
<tr>
<td>38</td>
<td>1111</td>
<td>231873</td>
<td>1111</td>
<td>231403</td>
<td>1011</td>
<td>231410</td>
</tr>
<tr>
<td>39</td>
<td>1111</td>
<td>232170</td>
<td>1111</td>
<td>231408</td>
<td>1011</td>
<td>231720</td>
</tr>
<tr>
<td>40</td>
<td>1111</td>
<td>233611</td>
<td>1111</td>
<td>231860</td>
<td>1011</td>
<td>233741</td>
</tr>
<tr>
<td>41</td>
<td>1111</td>
<td>246222</td>
<td>1111</td>
<td>235780</td>
<td>1011</td>
<td>237162</td>
</tr>
<tr>
<td>42</td>
<td>1111</td>
<td>233909</td>
<td>1111</td>
<td>234372</td>
<td>1011</td>
<td>235304</td>
</tr>
<tr>
<td>43</td>
<td>1111</td>
<td>235456</td>
<td>1111</td>
<td>233579</td>
<td>1011</td>
<td>235332</td>
</tr>
<tr>
<td>44</td>
<td>1111</td>
<td>234212</td>
<td>1111</td>
<td>234056</td>
<td>1011</td>
<td>234368</td>
</tr>
<tr>
<td>45</td>
<td>1111</td>
<td>235313</td>
<td>1111</td>
<td>234038</td>
<td>1011</td>
<td>234050</td>
</tr>
<tr>
<td>46</td>
<td>1111</td>
<td>232667</td>
<td>1111</td>
<td>232346</td>
<td>1011</td>
<td>234207</td>
</tr>
<tr>
<td>47</td>
<td>1111</td>
<td>236086</td>
<td>1111</td>
<td>235925</td>
<td>1011</td>
<td>233111</td>
</tr>
<tr>
<td>48</td>
<td>1111</td>
<td>234064</td>
<td>1111</td>
<td>235310</td>
<td>1011</td>
<td>232800</td>
</tr>
<tr>
<td>49</td>
<td>1111</td>
<td>234064</td>
<td>1111</td>
<td>234065</td>
<td>1011</td>
<td>233907</td>
</tr>
<tr>
<td>50</td>
<td>1111</td>
<td>237190</td>
<td>1111</td>
<td>234066</td>
<td>1011</td>
<td>234054</td>
</tr>
<tr>
<td>Iteration</td>
<td>Function</td>
<td>Execution XOR Time</td>
<td>Function DLATCH Time</td>
<td>Function DBARLATCH Time</td>
<td>Execution Time</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>--------------------</td>
<td>----------------------</td>
<td>-------------------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0000</td>
<td>232492</td>
<td>0111</td>
<td>233123</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0000</td>
<td>232342</td>
<td>0111</td>
<td>231247</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0000</td>
<td>232175</td>
<td>0111</td>
<td>232160</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0000</td>
<td>231712</td>
<td>0111</td>
<td>232498</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0000</td>
<td>231452</td>
<td>0111</td>
<td>231119</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0000</td>
<td>238737</td>
<td>0111</td>
<td>236865</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0000</td>
<td>232015</td>
<td>0111</td>
<td>232495</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0000</td>
<td>231874</td>
<td>0111</td>
<td>231875</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0000</td>
<td>232801</td>
<td>0111</td>
<td>232180</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0000</td>
<td>232189</td>
<td>0111</td>
<td>231451</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0000</td>
<td>231713</td>
<td>0111</td>
<td>232029</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0000</td>
<td>231399</td>
<td>0111</td>
<td>231876</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0000</td>
<td>231716</td>
<td>0111</td>
<td>232657</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0000</td>
<td>232374</td>
<td>0111</td>
<td>231875</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0000</td>
<td>232487</td>
<td>0111</td>
<td>233429</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0000</td>
<td>232960</td>
<td>0111</td>
<td>242490</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0000</td>
<td>232174</td>
<td>0111</td>
<td>231869</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0000</td>
<td>232017</td>
<td>0111</td>
<td>232647</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0000</td>
<td>232072</td>
<td>0111</td>
<td>231720</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0000</td>
<td>232706</td>
<td>0111</td>
<td>232332</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>0000</td>
<td>234940</td>
<td>0111</td>
<td>242016</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>0000</td>
<td>231604</td>
<td>0111</td>
<td>239357</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>0000</td>
<td>232486</td>
<td>0111</td>
<td>232646</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>0000</td>
<td>231544</td>
<td>0111</td>
<td>232647</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>0000</td>
<td>232333</td>
<td>0111</td>
<td>231873</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>0000</td>
<td>246404</td>
<td>0111</td>
<td>232022</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>0000</td>
<td>232186</td>
<td>0111</td>
<td>232642</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>0000</td>
<td>232494</td>
<td>0111</td>
<td>234689</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>0000</td>
<td>232964</td>
<td>0111</td>
<td>232344</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0000</td>
<td>231558</td>
<td>0111</td>
<td>232249</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>0000</td>
<td>232031</td>
<td>0111</td>
<td>232344</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>0000</td>
<td>232189</td>
<td>0111</td>
<td>231731</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>0000</td>
<td>231706</td>
<td>0111</td>
<td>233295</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>0000</td>
<td>231558</td>
<td>0111</td>
<td>231877</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>0000</td>
<td>231861</td>
<td>0111</td>
<td>232501</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>0000</td>
<td>231877</td>
<td>0111</td>
<td>232189</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>0000</td>
<td>232030</td>
<td>0111</td>
<td>231877</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>0000</td>
<td>231391</td>
<td>0111</td>
<td>232044</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>0000</td>
<td>232031</td>
<td>0111</td>
<td>232305</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>0000</td>
<td>231858</td>
<td>0111</td>
<td>232185</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>0000</td>
<td>235466</td>
<td>0111</td>
<td>232496</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>0000</td>
<td>233897</td>
<td>0111</td>
<td>232502</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>0000</td>
<td>236096</td>
<td>0111</td>
<td>234360</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>0000</td>
<td>236574</td>
<td>0111</td>
<td>237956</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>0000</td>
<td>236862</td>
<td>0111</td>
<td>232949</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>0000</td>
<td>232371</td>
<td>0111</td>
<td>232023</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>0000</td>
<td>232340</td>
<td>0111</td>
<td>232954</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>0000</td>
<td>233719</td>
<td>0111</td>
<td>234054</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>0000</td>
<td>233743</td>
<td>0111</td>
<td>236218</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>0000</td>
<td>232020</td>
<td>0111</td>
<td>235784</td>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>

Table 19

Cell Functions 15, 16 and 17 Verification Data
<table>
<thead>
<tr>
<th>Iteration</th>
<th>Function Execution</th>
<th>Function Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CBARLATCH Time</td>
<td>DCBARLATCH Time</td>
</tr>
<tr>
<td>1</td>
<td>0000    232187</td>
<td>1111    242610</td>
</tr>
<tr>
<td>2</td>
<td>0000    232969</td>
<td>1111    232502</td>
</tr>
<tr>
<td>3</td>
<td>0000    232490</td>
<td>1111    232645</td>
</tr>
<tr>
<td>4</td>
<td>0000    232499</td>
<td>1111    231249</td>
</tr>
<tr>
<td>5</td>
<td>0000    235998</td>
<td>1111    233574</td>
</tr>
<tr>
<td>6</td>
<td>0000    233746</td>
<td>1111    234062</td>
</tr>
<tr>
<td>7</td>
<td>0000    234213</td>
<td>1111    240929</td>
</tr>
<tr>
<td>8</td>
<td>0000    231249</td>
<td>1111    232489</td>
</tr>
<tr>
<td>9</td>
<td>0000    231868</td>
<td>1111    239681</td>
</tr>
<tr>
<td>10</td>
<td>0000    232348</td>
<td>1111    232020</td>
</tr>
<tr>
<td>11</td>
<td>0000    232037</td>
<td>1111    236512</td>
</tr>
<tr>
<td>12</td>
<td>0000    231407</td>
<td>1111    231249</td>
</tr>
<tr>
<td>13</td>
<td>0000    232189</td>
<td>1111    231408</td>
</tr>
<tr>
<td>14</td>
<td>0000    231556</td>
<td>1111    231243</td>
</tr>
<tr>
<td>15</td>
<td>0000    233117</td>
<td>1111    231713</td>
</tr>
<tr>
<td>16</td>
<td>0000    232025</td>
<td>1111    256429</td>
</tr>
<tr>
<td>17</td>
<td>0000    231244</td>
<td>1111    235149</td>
</tr>
<tr>
<td>18</td>
<td>0000    231556</td>
<td>1111    237241</td>
</tr>
<tr>
<td>19</td>
<td>0000    231717</td>
<td>1111    231078</td>
</tr>
<tr>
<td>20</td>
<td>0000    231562</td>
<td>1111    231249</td>
</tr>
<tr>
<td>21</td>
<td>0000    232025</td>
<td>1111    231401</td>
</tr>
<tr>
<td>22</td>
<td>0000    233120</td>
<td>1111    231869</td>
</tr>
<tr>
<td>23</td>
<td>0000    231714</td>
<td>1111    232341</td>
</tr>
<tr>
<td>24</td>
<td>0000    231554</td>
<td>1111    232494</td>
</tr>
<tr>
<td>25</td>
<td>0000    235145</td>
<td>1111    232339</td>
</tr>
<tr>
<td>26</td>
<td>0000    232346</td>
<td>1111    232335</td>
</tr>
<tr>
<td>27</td>
<td>0000    232184</td>
<td>1111    231877</td>
</tr>
<tr>
<td>28</td>
<td>0000    231719</td>
<td>1111    232191</td>
</tr>
<tr>
<td>29</td>
<td>0000    232025</td>
<td>1111    231721</td>
</tr>
<tr>
<td>30</td>
<td>0000    231865</td>
<td>1111    232029</td>
</tr>
<tr>
<td>31</td>
<td>0000    231713</td>
<td>1111    232024</td>
</tr>
<tr>
<td>32</td>
<td>0000    231711</td>
<td>1111    232654</td>
</tr>
<tr>
<td>33</td>
<td>0000    231548</td>
<td>1111    235621</td>
</tr>
<tr>
<td>34</td>
<td>0000    233518</td>
<td>1111    232962</td>
</tr>
<tr>
<td>35</td>
<td>0000    231717</td>
<td>1111    231714</td>
</tr>
<tr>
<td>36</td>
<td>0000    233429</td>
<td>1111    231881</td>
</tr>
<tr>
<td>37</td>
<td>0000    231721</td>
<td>1111    232029</td>
</tr>
<tr>
<td>38</td>
<td>0000    232017</td>
<td>1111    237202</td>
</tr>
<tr>
<td>39</td>
<td>0000    233264</td>
<td>1111    233756</td>
</tr>
<tr>
<td>40</td>
<td>0000    231557</td>
<td>1111    231871</td>
</tr>
<tr>
<td>41</td>
<td>0000    234196</td>
<td>1111    231541</td>
</tr>
<tr>
<td>42</td>
<td>0000    232188</td>
<td>1111    232846</td>
</tr>
<tr>
<td>43</td>
<td>0000    232786</td>
<td>1111    234374</td>
</tr>
<tr>
<td>44</td>
<td>0000    233128</td>
<td>1111    233349</td>
</tr>
<tr>
<td>45</td>
<td>0000    233587</td>
<td>1111    235601</td>
</tr>
<tr>
<td>46</td>
<td>0000    235791</td>
<td>1111    233071</td>
</tr>
<tr>
<td>47</td>
<td>0000    233580</td>
<td>1111    234987</td>
</tr>
<tr>
<td>48</td>
<td>0000    236384</td>
<td>1111    233976</td>
</tr>
<tr>
<td>49</td>
<td>0000    237040</td>
<td>1111    232498</td>
</tr>
<tr>
<td>50</td>
<td>0000    231410</td>
<td>1111    235159</td>
</tr>
</tbody>
</table>
Appendix C

Array Verification Data
### Table 21
3x3 Array Case 1 Routing Data

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Execution Time</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>231249</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>2</td>
<td>231249</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>3</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>4</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>6</td>
<td>231247</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>7</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>8</td>
<td>231255</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>9</td>
<td>231249</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>10</td>
<td>231249</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>11</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>12</td>
<td>231094</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>13</td>
<td>231249</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>14</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>15</td>
<td>231249</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>16</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>17</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>18</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>19</td>
<td>231094</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>20</td>
<td>231251</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>21</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>22</td>
<td>231249</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>23</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>24</td>
<td>231249</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>25</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
</tbody>
</table>

### Table 22
3x3 Array Case 2 Routing Data

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Execution Time</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>2</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>3</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>4</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>231251</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>6</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>7</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>8</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>9</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>10</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>11</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>12</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>13</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
</tbody>
</table>
## Table 22--Continued

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Execution Time</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>15</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>16</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>17</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>18</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>19</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>20</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>21</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>22</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>23</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>24</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>25</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
</tbody>
</table>

## Table 23

3x3 Array Case 3 Routing Data

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Execution Time</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>2</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>3</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>4</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>6</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>7</td>
<td>231249</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>8</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>9</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>10</td>
<td>231249</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>11</td>
<td>231254</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>12</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>13</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>14</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>15</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>16</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>17</td>
<td>231254</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>18</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>19</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>20</td>
<td>231251</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>21</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>22</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>23</td>
<td>231249</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>24</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>25</td>
<td>231094</td>
<td>0100</td>
<td>0100</td>
</tr>
</tbody>
</table>
Table 24
3x3 Array Case 4 Routing Data

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Execution Time</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>2</td>
<td>231243</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>3</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>4</td>
<td>231249</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>231251</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>6</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>7</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>8</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>9</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>10</td>
<td>231249</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>11</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>12</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>13</td>
<td>231251</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>14</td>
<td>231246</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>15</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>16</td>
<td>231251</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>17</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>18</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>19</td>
<td>231094</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>20</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>21</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>22</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>23</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>24</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>25</td>
<td>231250</td>
<td>0100</td>
<td>0100</td>
</tr>
</tbody>
</table>
Appendix D

7 Segment Decoder Linearity Data
### Table 25
#### 7 Segment Decoder x1 Clock Data

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Start Time</th>
<th>End Time</th>
<th>Running Time</th>
<th>Input Values</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>53413</td>
<td>6979663</td>
<td>6926250</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>2</td>
<td>45827</td>
<td>6973641</td>
<td>6927814</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>3</td>
<td>45806</td>
<td>6974087</td>
<td>6928281</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>4</td>
<td>45816</td>
<td>6973943</td>
<td>6928127</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>5</td>
<td>45766</td>
<td>6972948</td>
<td>6927182</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>6</td>
<td>45777</td>
<td>6972028</td>
<td>6926251</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>7</td>
<td>45832</td>
<td>6977864</td>
<td>6932032</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>8</td>
<td>45876</td>
<td>6974315</td>
<td>6928439</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>9</td>
<td>45742</td>
<td>6973086</td>
<td>6927344</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>10</td>
<td>45788</td>
<td>6974851</td>
<td>6929063</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
</tbody>
</table>

### Table 26
#### 7 Segment Decoder x2 Clock Data

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Start Time</th>
<th>End Time</th>
<th>Running Time</th>
<th>Input Values</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>68267</td>
<td>3577329</td>
<td>3509062</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>2</td>
<td>48319</td>
<td>3555672</td>
<td>3507353</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>3</td>
<td>48172</td>
<td>3555985</td>
<td>3507813</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>4</td>
<td>49167</td>
<td>3558698</td>
<td>3509531</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>5</td>
<td>48363</td>
<td>3556643</td>
<td>3508280</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>6</td>
<td>48262</td>
<td>3555755</td>
<td>3507493</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>7</td>
<td>48334</td>
<td>3556145</td>
<td>3507811</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>8</td>
<td>48546</td>
<td>3557920</td>
<td>3509374</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>9</td>
<td>48102</td>
<td>3554665</td>
<td>3506563</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>10</td>
<td>48188</td>
<td>3563657</td>
<td>3515469</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
</tbody>
</table>

### Table 27
#### 7 Segment Decoder x4 Clock Data

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Start Time</th>
<th>End Time</th>
<th>Running Time</th>
<th>Input Values</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50448</td>
<td>1828101</td>
<td>1777653</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>2</td>
<td>48390</td>
<td>1823861</td>
<td>1775471</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>3</td>
<td>48168</td>
<td>1824418</td>
<td>1776250</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>4</td>
<td>48385</td>
<td>1825418</td>
<td>1776971</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
</tbody>
</table>
Table 27--Continued

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Start Time</th>
<th>End Time</th>
<th>Running Time</th>
<th>Input Values</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>48447</td>
<td>1831416</td>
<td>1782969</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>6</td>
<td>48495</td>
<td>1829902</td>
<td>1781352</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>7</td>
<td>48550</td>
<td>1829331</td>
<td>1780781</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>8</td>
<td>48698</td>
<td>1829649</td>
<td>1780951</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>9</td>
<td>48690</td>
<td>1829953</td>
<td>1781263</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>10</td>
<td>48511</td>
<td>1830074</td>
<td>1781563</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
</tbody>
</table>

Table 28
7 Segment Decoder x8 Clock Data

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Start Time</th>
<th>End Time</th>
<th>Running Time</th>
<th>Input Values</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>49272</td>
<td>958649</td>
<td>909377</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>2</td>
<td>48233</td>
<td>956516</td>
<td>908283</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>3</td>
<td>48215</td>
<td>956185</td>
<td>907970</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>4</td>
<td>48989</td>
<td>955396</td>
<td>906407</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>5</td>
<td>48413</td>
<td>957008</td>
<td>908595</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>6</td>
<td>48168</td>
<td>955512</td>
<td>907344</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>7</td>
<td>48050</td>
<td>955862</td>
<td>907812</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>8</td>
<td>48268</td>
<td>955456</td>
<td>907188</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>9</td>
<td>48305</td>
<td>955805</td>
<td>907500</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>10</td>
<td>48350</td>
<td>958819</td>
<td>910469</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
</tbody>
</table>

Table 29
7 Segment Decoder /2 Clock Data

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Start Time</th>
<th>End Time</th>
<th>Running Time</th>
<th>Input Values</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>45967</td>
<td>13860030</td>
<td>13814063</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>2</td>
<td>45894</td>
<td>13857144</td>
<td>13811250</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>3</td>
<td>45979</td>
<td>13861292</td>
<td>13815313</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>4</td>
<td>45913</td>
<td>13856851</td>
<td>13810938</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>5</td>
<td>45765</td>
<td>13859359</td>
<td>13813594</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>6</td>
<td>48541</td>
<td>14034635</td>
<td>13986094</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>7</td>
<td>49198</td>
<td>13976228</td>
<td>13927030</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>8</td>
<td>48153</td>
<td>13958310</td>
<td>13910157</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>9</td>
<td>48311</td>
<td>14006593</td>
<td>13958252</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
<tr>
<td>10</td>
<td>48164</td>
<td>13996602</td>
<td>13948438</td>
<td>1 2 3 4 5 6</td>
<td>63 6 91 79 102</td>
</tr>
</tbody>
</table>
Table 30

7 Segment Decoder /4 Clock Data

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Start Time</th>
<th>End Time</th>
<th>Running Time</th>
<th>Input Values</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>86858</td>
<td>27887791</td>
<td>27800933</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>2</td>
<td>50829</td>
<td>27902860</td>
<td>27852031</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>3</td>
<td>48205</td>
<td>27901332</td>
<td>27853127</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>4</td>
<td>48742</td>
<td>27935931</td>
<td>27887109</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>5</td>
<td>48211</td>
<td>27899462</td>
<td>27851251</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>6</td>
<td>48243</td>
<td>27893399</td>
<td>27845156</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>7</td>
<td>48185</td>
<td>27896154</td>
<td>27847969</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>8</td>
<td>48341</td>
<td>27897716</td>
<td>27849463</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>9</td>
<td>48253</td>
<td>27894648</td>
<td>27846395</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>10</td>
<td>48395</td>
<td>27893708</td>
<td>27845313</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
</tbody>
</table>

Table 31

7 Segment Decoder /8 Clock Data

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Start Time</th>
<th>End Time</th>
<th>Running Time</th>
<th>Input Values</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>61748</td>
<td>55915028</td>
<td>55853280</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>2</td>
<td>49181</td>
<td>55941523</td>
<td>55892342</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>3</td>
<td>48798</td>
<td>55941142</td>
<td>55892344</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>4</td>
<td>48473</td>
<td>55941756</td>
<td>55893280</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>5</td>
<td>48602</td>
<td>55951728</td>
<td>55903126</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>6</td>
<td>48549</td>
<td>55941987</td>
<td>55893438</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>7</td>
<td>48612</td>
<td>55951763</td>
<td>55909019</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>8</td>
<td>48655</td>
<td>55951780</td>
<td>55903125</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>9</td>
<td>48591</td>
<td>56145466</td>
<td>56096515</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
<tr>
<td>10</td>
<td>48675</td>
<td>55978206</td>
<td>55929531</td>
<td>7 8 9 0 1 2</td>
<td>63 7 127 111 63</td>
</tr>
</tbody>
</table>
Appendix E

cfg2occ Perl Program
#! /usr/local/bin/perl
# cfg2occ : an extendable CAL to Occam conversion utility
#
# usage :
#    cfg2occ <in-file> <xmax>
#
# infile = the cfg source file to be converted
# xmax  = the maximum x size of the array
#
# based upon specifications provided by Roy Krans
#
# Richard Taylor, June 1994
#
# Modification History
# 6/17/94 R.Krans changed address output to represent
# the actual Occam procedure call;
# also indented all output 8 spaces
#
# 6/20/94 R.Krans reorder functions to match Table 1
# in the CAL1024 Datasheet
#
# 6/24/94 R.Krans modified the split functions to ignore
# leading whitespaces:
# split(/ /...) to split(' '...)
#
# 7/8/94 R.Krans modified function names to meet those
# used by Clare
#
# 7/20/94 R.Krans changed indentation to 4 spaces
#
# check the number of parameters - there should be 2 (inc 0)
if ($#ARGV != 1){
  die "usage : cfg2occ <in-file> <xmax>\n";
}

# open the input file for reading
open(INPUT, $ARGV[0]) || die "could not find : $ARGV[0]\n";

# check that the maximum size of the array > 0
if ($ARGV[1] < 1){
  die "the CAL array must have some cells in it\n";
}
$XMAX = $ARGV[1];

# OK, thats out of the way, define the constants (as per Krans
# document
#
# begin by defining the tables that will be used to associate
# directions and functions with numbers.
#
# the source/sink for a signal
%direction = (  
  'NORTH', 0,  
  'SOUTH', 1,  
  'EAST', 2,  
  'WEST', 3,  
  'SELF', 4,  
  'G1', 5,  
  'G2', 6,  
);

# the function itself  
#function = (  
  'ZERO',0,  
  'ONE',1,  
  'X1',2,  
  'X1BAR',3,  
  'X2',4,  
  'X2BAR',5,  
  'AND',6,  
  'X1ANDX2BAR',7,  
  'X1BARANDX2',8,  
  'NOR',9,  
  'OR',10,  
  'X1ORX2BAR',11,  
  'X1BARORX2',12,  
  'XNOR',13,  
  'NAND',14,  
  'XOR',15,  
  'DLATCH',16,  
  'DBARLATCH',17,  
  'CBARLATCH',18,  
  'DCBARLATCH',19,  
);

# now the loop  
# read past the bumf at the beginning of each file  
# for each CELL/ENDCELL  
# compute address  
# collect sources  
# collect function  
# organize word  
# write out data to an array of addresses (for sorting)  
# sort the data  
# extract addresses and data and then write them to stdout  
# terminate on end of file

while (<INPUT>){  
  file  
  chop;  
  if (/CELL/){  
    if (/ENDCELL/){  
      ($dummy, $X, $Y) = split(' ', $);  
      # get X and Y  
      location of cell  
      $SSOURCE = $NSOURCE = $WSOURCE = $ESOURCE = 0;  
      $X2SOURCE = $X1SOURCE = $FUNCTION = 0;  
      # zero fields  
    }  
  }
}
if (/FUNCTION/) {
    ($dummy, $fn_number) = split(' ', $);
    $FUNCTION = $function{$fn_number};
}
if (/SSOURCE/) {
    ($dummy, $source_number) = split(' ', $);
    $SSOURCE = $direction{$source_number};
}
if (/NSOURCE/) {
    ($dummy, $source_number) = split(' ', $);
    $NSOURCE = $direction{$source_number};
}
if (/ESOURCE/) {
    ($dummy, $source_number) = split(' ', $);
    $ESOURCE = $direction{$source_number};
}
if (/WSOURCE/) {
    ($dummy, $source_number) = split(' ', $);
    $WSOURCE = $direction{$source_number};
}
if (/X2SOURCE/) {
    ($dummy, $source_number) = split(' ', $);
    $X2SOURCE = $direction{$source_number};
}
if (/X1SOURCE/) {
    ($dummy, $source_number) = split(' ', $);
    $X1SOURCE = $direction{$source_number};
}
if (/ENDCELL/) {
    # pack the words and calculate the addresses
    $ADDRESS1 = $Y * ($XMAX * 4) + ($X * 4);
    $MSW = 0 + ($WSOURCE << 3) + ($ESOURCE);
    $MSW = ($MSW << 8) + ($SSOURCE << 3) + ($NSOURCE);
    $NSW = 0 + ($FUNCTION << 8) + ($X2SOURCE << 3) + ($X1SOURCE);
    # set up a composite address and data word
    $DATUM = sprintf("\06d \04x \04x", $ADDRESS1, $MSW, $NSW);
    # add it to the complete list of address and data words
    push(@CODE_LIST, $DATUM);
}

# sort the list of addresses in ascending order
@SORT_LIST = sort @CODE_LIST;
# for each line in the list, split out the address, MSW and NSW and
# print them in the correct format
foreach $line (@SORT_LIST) {
    ($ADDRESS1, $MSW, $NSW) = split(/ /, $line);
    $ADDRESS2 = $ADDRESS1 + 2;
    printf(" SetBaseAddress(#\06x, Mode, Address)\n", $ADDRESS1);
    printf(" WriteWord(INT #$MSW, Mode, LData.out, HData.out)\n");
    printf(" WriteWord(INT #$NSW, Mode, LData.out, HData.out)\n");
}
# quit
Appendix F

CHS2x4 (Massively) Parallel Sorter C Control Code
mpp_sort.c

Program to implement a (massively) parallel sorting algorithm on the CHS2x4

7-14-94 R. Krans

#include "util.h"
#include "lowlvel.h"
#include "progchip.h"
#include "readcal.h"
#include "graph.h"
#include <varargs.h>
#include "readcnf.h"

static Bool read_cal();
static void mpp_sort();
static void program_from_file();

ChipRam ram;
CHS2X4 * config;

void main()
{
    FILE *f;

    f=NULL;
    config=read_cnf(f);
    set_io_address(config->board_add);
    reset_chs2x4();
    mpp_sort();
} /* main */

/****************************************************************************/
static void mpp_sort()
/****************************************************************************/
{
    long slot_0_add,slot_1_add;
    int i;
    unsigned char b;
    unsigned d;
    int data[24] = {0,0,0,0,0,0,0,0,155,34,150,25,250,55,255,
                   255,255,255 };;

    if (config->mems[0]==MEM_128K)
    {
        slot_0_add=mem_128_byte_addresses[0];
        slot_1_add=mem_128_byte_addresses[1];
    }
    else
    {
        slot_0_add=mem_512_byte_addresses[0];
        slot_1_add=mem_512_byte_addresses[1];
    }
program_from_file("mpp_sort.cal");

for (i=0; i<18; i++) {
  printf("\nOutputting %d to %ld ",data[i], slot_1_add+i);
  out_byte(slot_1_add+i,(char) data[i]);
}

/* Set board memory address counter */
set_counter_address(slot_0_add);
set_run_mode(TRUE);
initiate_local_transfer(); initiate_local_transfer(); initiate_local_transfer(); initiate_local_transfer(); initiate_local_transfer(); initiate_local_transfer(); initiate_local_transfer(); initiate_local_transfer();
toggle_g2(); initiate_local_transfer(); initiate_local_transfer(); initiate_local_transfer(); initiate_local_transfer(); initiate_local_transfer(); initiate_local_transfer(); initiate_local_transfer(); initiate_local_transfer();
set_run_mode(FALSE);

/* Copy data from output buffer back to input buffer */
for (i=0; i<33; i++) {
  b=in_byte(slot_0_add+i);
  /* b=b; */
  printf("\nRead byte = %d from %ld ",b,slot_0_add+i);
}

/*****************************/
static void program_from_file(char *file_name)
  char *file_name;
/*****************************/

{ FILE *f;
  int i,b,d,j,cl,cj;
  char *full_name;
full_name=find_file(file_name,".");
if (full_name==NULL)
{
    fprintf(stderr,"Cannot find input file %s in current
directory.\n",file_name);
    exit(1);
}
f=fopen(full_name,"r");
if (f==NULL)
{
    fprintf(stderr,"Cannot open input file for reading.\n");
    exit(1);
}

printf("Reading CAL File\n");
for (;read_cal_chip(&ram,&ci,&cj,f);)
{
    if (config->cals[ci][cj]!=SLOT_CAL)
    {
        fprintf(stderr,".CAL file attempts to program slot (%d,%d)
which does not contain a CAL chip.\n",ci,cj);
        fprintf(stderr,"Fatal Error - Cannot Continue.\n");
        exit(1);
    }
    printf("Configuring Chip At (%d,%d).\n",ci,cj);
    program_chip(&ram,ci,cj);
}
fclose(f);

if ((config->mems[0]==MEM_EMPTY)||((config->mems[1]==MEM_EMPTY))
{
    printf("Must have memory in slot 0 and 1 - cannot run test.\n");
    set_run_mode(FALSE);
    return;
}

} /* program_from_file */

/**************************
/* Errors handled by library text_error() function */
/*VARARGS0*/
void error(va_args)
va dcl
/**************************/

{ va_list args;

    va_start(args);

    text_error(args);
} /* error */
Appendix G

Occam (Massively) Parallel Sorter Control Code
SEQ
  Reset(Mode)
  Program1(Mode, LData.in, LData.out, HData.in, HData.out, Address)
  Program2(Mode, LData.in, LData.out, HData.in, HData.out, Address)
  -- configuration file was too large to compile as one program
  -- so it was split into two.
  SetBaseAddress(#280000, Mode, Address)
  WriteWord(INT #0000, Mode, LData.out, HData.out)
  WriteWord(INT #0000, Mode, LData.out, HData.out)
  WriteWord(INT #0000, Mode, LData.out, HData.out)
  WriteWord(INT #229B, Mode, LData.out, HData.out)
  WriteWord(INT #1996, Mode, LData.out, HData.out)
  WriteWord(INT #37FA, Mode, LData.out, HData.out)
  WriteWord(INT #FFFF, Mode, LData.out, HData.out)
  WriteWord(INT #FFFF, Mode, LData.out, HData.out)
  SetBaseAddress(#200000, Mode, Address)
  ToggleRunMode(Mode)
  clock ? time
  main.disp.channel ! 1;0;time
  -- data received during local transfers is
  -- displayed from the array
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  ToggleG2(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  LocalTransfer(Mode)
  clock ? time
  main.disp.channel ! 0;0;time
  ToggleRunMode(Mode)
  Terminate(Mode)
Appendix H

CHS2x4 Bit Serial Adder C Control Code
ser_addr.c

Program to implement a bit serial adder
on the CHS2x4

7-27-94 R. Krans

#include "util.h"
#include "lowlevel.h"
#include "propchip.h"
#include "readcal.h"
#include "graph.h"
#include "readcnf.h"

static Bool read_cal();
static void ser_addr();
static void program_from_file();

ChipRam ram;
CHS2X4 * config;

void main()
{
    FILE *f;

    f=NULL;
    config=read cnf(f);
    set io address(config->board_add);
    reset chs2x4();
    ser_addr();
} /* main */

/*****************************/
static void ser_addr()
/*****************************/
{
    long slot_0_add,slot_1_add;
    int i;
    unsigned char b;
    unsigned d;

    /* six zeroes to clear latches and data items */
    /* the data items shown below are not the values */
    /* that will directly added. They represent the */
    /* respective bit position sums since this is a */
    /* serial adder */

    int data[20] = {0,0,0,0,0,0,145,81,94,13,101,55,34,0,0,0,0,0};

    if (config->mems[0]==MEM_128K)
    {
        slot_0_add=mem_128_byte_addresses[0];
        slot_1_add=mem_128_byte_addresses[1];
    }
else
{
    slot_0_add=mem_512_byte_addresses[0];
    slot_1_add=mem_512_byte_addresses[1];
}

program_from_file("ser_addr.cal");

for (i=0; i<18; i++) {
    printf("\nOutputting %d to %ld ",data[i], slot_1_add+i);
    out_byte(slot_1_add+i,(char) data[i]); }

/* Set board memory address counter */
set_counter_address(slot_0_add);
set_run_mode(TRUE);

initiate_local_transfer();
initiate_local_transfer();
initiate_local_transfer();
initiate_local_transfer();
initiate_local_transfer();
initiate_local_transfer();
initiate_local_transfer();
initiate_local_transfer();
initiate_local_transfer();
initiate_local_transfer();
initiate_local_transfer();
initiate_local_transfer();
initiate_local_transfer();
initiate_local_transfer();
initiate_local_transfer();
initiate_local_transfer();
set_run_mode(FALSE);

/* Copy data from output buffer back to input buffer */

printf("\nValid data should begin at #11");
for (i=0; i<20; i++) {
    b=in_byte(slot_0_add+i);
    /* b=b; */
    printf("\n%d. Read byte = %d from %ld ",i+1,b,slot_0_add+i);
}

static void program_from_file(file_name)
char *file_name;

{
FILE *f;
int i,b,d,j,ci,cj;
char *full_name;

full_name=find_file(file_name, ".");
if (full_name==NULL)
{
    fprintf(stderr,"Cannot find input file %s in current
directory.\n", file_name);
    exit(1);
}
f=fopen(full_name, "r");
if (f==NULL)
{
    fprintf(stderr,"Cannot open input file for reading.\n");
    exit(1);
}

printf("Reading CAL File\n");
for (;read_cal_chip(&ram, &ci, &cj, f);)
{
    if (config->cals[ci][cj]!=SLOT_CAL)
    {
        fprintf(stderr,".CAL file attempts to program slot (%d, %d)
which does not contain a CAL chip.\n", ci, cj);
        fprintf(stderr,"Fatal Error - Cannot Continue.\n");
        exit(1);
    }
    printf("Configuring Chip At (%d, %d).\n", ci, cj);
    program_chip(&ram, ci, cj);
}
fclose(f);

if ((config->mems[0]==MEM_EMPTY) || (config->mems[1]==MEM_EMPTY))
{
    printf("Must have memory in slot 0 and 1 - cannot run test.\n");
    set_run_mode(FALSE);
    return;
}
} /* program_from_file */

/*****************************/
/* Errors handled by library text_error() function */
/*****************************/
void error(va_list)

va_dcl
/*****************************/
{
    va_list args;
    va_start(args);
    text_error(args);
} /* error */
Appendix I

Occam Bit Serial Adder Control Code
#USE "fsraddr.lib"

SEQ
Reset(Mode)
BitSerialAdder(Mode, LData.in, LData.out, HData.in, HData.out, Address)
SetBaseAddress(#280000, Mode, Address)
WriteWord(INT #0000, Mode, LData.out, HData.out)
WriteWord(INT #0000, Mode, LData.out, HData.out)
WriteWord(INT #0000, Mode, LData.out, HData.out)
WriteWord(INT #5191, Mode, LData.out, HData.out)
WriteWord(INT #0D5E, Mode, LData.out, HData.out)
WriteWord(INT #3765, Mode, LData.out, HData.out)
WriteWord(INT #0022, Mode, LData.out, HData.out)
WriteWord(INT #0000, Mode, LData.out, HData.out)
WriteWord(INT #0000, Mode, LData.out, HData.out)
SetBaseAddress(#200000, Mode, Address)
ToggleRunMode(Mode)
clock ? time
main.disp.channel 1 0;0;time
-- data received during local transfers is
-- displayed from the array
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
LocalTransfer(Mode)
clock ? time
main.disp.channel 1 0;0;time
ToggleRunMode(Mode)
Terminate(Mode)
Appendix J

PC Integer Addition Timing Program
Program to estimate the time required by this PC to add 32 bit numbers. These values will be compared to those of the CHS2x4 running a bit serial adder configuration.

RKrans
8/1/94

#include <time.h>
#include <sys/timeb.h>
#include <stdio.h>

void main()
{
    int i, j, sum, loop;
    char *start_time, *end_time;
    struct timeb tstruct;
    long data[4096];

    /**************************************/
    /* create 16k of random 8 bit numbers */
    for (i=0; i<4096; i++){
        data[i] = rand();
    } /* for */
    sum = O;

    printf("\nstart\tend\n");
    for (loop=O;loop<10;loop++){
    /**************************************/
    /* record the start time */
    ftime( &tstruct);
    start_time = ctime( &(tstruct.time));
    printf("\n%.2s %hu", &start_time[17],tstruct.millitm);
    /**************************************/
    /* add the 16k of numbers */
    for (j=0; j<4096; j++){
        for (i=0; i<4096; i++){
            sum = sum + data[i];
        }
    }
    /**************************************/
    /* record the end time */
    ftime( &tstruct);
    end_time = ctime( &(tstruct.time));
    printf("\n%.2s %hu", &end_time[17],tstruct.millitm);
    } /* for loop */
} /* main */
Appendix K

PC Sorter Timing Program
sorttime.c

Program to estimate the time required by this PC to sort 8-bit numbers. These values will be compared to those of the CHS2x4 running a (massively) parallel sorter configuration.

RKrans
8/2/94

#include <time.h>
#include <sys/timeb.h>
#include <stdio.h>
define MAX 4096
#define ITERATIONS 10

void main()
{
  int i, j, loop;
  char *start_time, *end_time;
  struct timeb tstruct;
  unsigned char data[MAX], temp;

  /* create random 8 bit numbers */
  for (i=0; i<MAX; i++){
    data[i] = rand();
  } /* for */

  printf("\nstart\tend\n");
  for (loop=0; loop<ITERATIONS; loop++){
    /* record the start time */
    _ftime( &tstruct);
    start_time = ctime( &(tstruct.time));
    printf("\n%2s %hu", &start_time[17], tstruct.millitm);

    /* sort the numbers */
    for (j=MAX; j>0; j--)
      for (i=1; i<j; i++)
        if (data[i] > data[i-1]) {
          temp = data[i];
          data[i] = data[i-1];
          data[i-1] = temp;
        }

    /* record the end time */
  }
}
ftime(&tstruct);
end_time = ctime(&tstruct.time));
printf("\t%.2s %hu", &end_time[17], tstruct.millitm);

} /* for loop */
} /* main */
Appendix L

Stochastic Neural Network C Control Code
This implementation uses stochastic bit streams to implement weighted input multiplication with a single exclusive or gate.
program_from_file("stagel.cal");

for (i=0;i<20;i++){
    if ( i==0 ) n=1;
    else n = 0;
    printf("\nOutputting \%d to \%ld ", n, slot_1_add+i);
    out_byte(slot_1_add+i,(char) n);
}

/* Set board memory address counter */
set_counter_address(slot_0_add);
set_run_mode(TRUE);
for(i=0;i<20;i++){
    initiate_local_transfer();
}
set_run_mode(FALSE);

/* Copy data from output buffer back to data buffer */
/* using values 10-14 */
for ( i=9 ; i<13 ; i++){
    data[0][i-9] = in_byte(slot_0_add+i);
    printf("\nRead byte = \%d from \%ld ",i,data[0][i-9],slot_0_add+i);
}

program_from_file("stage2.cal");

for (i=0;i<20;i++){
    if ( i==0 ) n=1;
    else n = 0;
    printf("\nOutputting \%d to \%ld ", n, slot_1_add+i);
    out_byte(slot_1_add+i,(char) n);
}

/* Set board memory address counter */
set_counter_address(slot_0_add);
set_run_mode(TRUE);
for(i=0;i<20;i++){
    initiate_local_transfer();
}
set_run_mode(FALSE);
/* Copy data from output buffer back to data buffer */
/* using values 10-14 */

for (i=9; i<13; i++){
    data[1][i-9] = in_byte(slot_0_add+i);
    printf("\n\%d Read byte = \%d from \%ld ",i,data[1][i-9],slot_0_add+i);
}

/******************************************************/
/* */
/* obtain the input bit streams from stage3, using */
/* input 3 and the corresponding weights for input 3 */
/* */
/******************************************************/

program_from_file("stage3.cal");

for (i=0;i<20;i++){
    if (i==0) n=1;
    else n = 0;
    printf("\nOutputting \%d to \%ld ", n, slot_1_add+i);
    out_byte(slot_1_add+i, (char) n);
}

/* Set board memory address counter */

set_counter_address(slot_0_add);
set_run_mode(TRUE);
for (i=0;i<20;i++){
    initiate_local_transfer();
}
set_run_mode(FALSE);

/* Copy data from output buffer back to data buffer */
/* using values 10-14 */

for (i=9; i<13; i++){
    data[2][i-9] = in_byte(slot_0_add+i);
    printf("\n\%d Read byte = \%d from \%ld ",i,data[2][i-9],slot_0_add+i);
}

/******************************************************/
/* */
/* obtain the input bit streams from stage4, using */
/* input 4 and the corresponding weights for input 4 */
/* */
/******************************************************/

program_from_file("stage4.cal");

for (i=0;i<20;i++){
    if (i==0) n=1;
    else n = 0;
    printf("\nOutputting \%d to \%ld ", n, slot_1_add+i);
    out_byte(slot_1_add+i, (char) n);
/* Set board memory address counter */

set_counter_address(slot_0_add);
set_run_mode(TRUE);
for (i=0; i<20; i++) {
    initiate_local_transfer();
}
set_run_mode(FALSE);

/* Copy data from output buffer back to data buffer */
/* using values 10-14 */

for (i=9; i<13; i++) {
    data[3][i-9] = in_byte(slot_0_add+i);
    printf("\n%d Read byte = %d from %ld ", i, data[3][i-9], slot_0_add+i);
}

/***********************************************
/*
/* obtain the output bit streams from neur net, using*
/* the previously stored input and weight bit streams*
/*
***********************************************/

program_from_file("neur_net.cal");

/* flush counters */

for (i=0; i<4; i++) {
    n=32;
    printf("\nOutputting %d to %ld ", n, slot_1_add+i);
    out_byte(slot_1_add+i,(char) n);
}

/* preset counter threshold values */
/* when pin 5 is high, pins 1-4 are used as threshold inputs */

for (i=4; i<8; i++) {
    n=32+30;
    if (i==6) n=32;
    /* slight quirk - last threshold bit set with pin 5 low */
    if (i==7) n=0;
    printf("\nOutputting %d to %ld ", n, slot_1_add+i);
    out_byte(slot_1_add+i,(char) n);
}

/* store bit streams */

index = 8;
for (i=0; i<4; i++) {
    for (j=0; j<4; j++) {
        n=data[i][j];
        printf("\nOutputting %d to %ld ", n, slot_1_add+index);
        out_byte(slot_1_add+index,(char) n);
        index++;
    }
}
/* Set board memory address counter */

set_counter_address(slot_0_add);
set_run_mode(TRUE);
for (i=0; i<30; i++){
    initiate_local_transfer();
}
set_run_mode(FALSE);

/* display output data */

for (i=0; i<30; i++){
    n = in_byte(slot_0_add+i);
    printf("\n%d) Read byte = %d from %ld ",i,n,slot_0_add+i);
}

} /* neural_demo */

/*******************/
static void program_from_file(file_name)
    char *file_name;
/*******************/
{
    FILE *f;
    int i,b,d,j,ci,cj;
    char *full_name;

    full_name=find_file(file_name,".");
    if (full_name==NULL)
    {
        fprintf(stderr,"Cannot find input file \s in current directory.\n",file_name);
        exit(1);
    }
    f=fopen(full_name,"r");
    if (f==NULL) -
    {
        fprintf(stderr,"Cannot open input file for reading.\n");
        exit(1);
    }

    printf("Reading CAL File\n");
    for (;read_cal_chip(&ram,&ci,&cj,f));
    {
        if (config->cals[ci][cj]!=SLOT_CAL)
        {
            fprintf(stderr, ".CAL file attempts to program slot (%d,%d) which does not contain a CAL chip.\n",ci,cj);
            fprintf(stderr,"Fatal Error - Cannot Continue.\n");
            exit(1);
        }
        printf("Configuring Chip At (%d,%d).\n",ci,cj);
        program_chip(&ram,ci,cj);
    }
    fclose(f);
if ((config->mems[0]==MEM_EMPTY) || (config->mems[1]==MEM_EMPTY))
{
    printf("Must have memory in slot 0 and 1 - cannot run test.\n");
    set_run_mode(FALSE);
    return;
}
} /* program_from_file */

/*****************************************
/* Errors handled by library text_error() function */
/*VARARGS0*/
void error(va alist)
    va_dcl
/*******************************************/
{
    va_list args;
    va_start(args);
    text_error(args);
} /* error */
Appendix M

Stochastic Neural Network Example Output
Reading CAL File
Configuring Chip At (0,0).
Configuring Chip At (0,1).
Configuring Chip At (1,0).
Configuring Chip At (1,1).
Configuring Chip At (2,0).
Configuring Chip At (2,1).
Configuring Chip At (3,0).
Configuring Chip At (3,1).
Outputting 1 to 2621440
Outputting 0 to 2621441
Outputting 0 to 2621442
Outputting 0 to 2621443
Outputting 0 to 2621444
Outputting 0 to 2621445
Outputting 0 to 2621446
Outputting 0 to 2621447
Outputting 0 to 2621448
Outputting 0 to 2621449
Outputting 0 to 2621450
Outputting 0 to 2621451
Outputting 0 to 2621452
Outputting 0 to 2621453
Outputting 0 to 2621454
Outputting 0 to 2621455
Outputting 0 to 2621456
Outputting 0 to 2621457
Outputting 0 to 2621458
Outputting 0 to 2621459
9) Read byte = 28 from 2097161
10) Read byte = 29 from 2097162
11) Read byte = 12 from 2097163
12) Read byte = 0 from 2097164 Reading CAL File
Configuring Chip At (0,0).
Configuring Chip At (0,1).
Configuring Chip At (1,0).
Configuring Chip At (1,1).
Configuring Chip At (2,0).
Configuring Chip At (2,1).
Configuring Chip At (3,0).
Configuring Chip At (3,1).
Outputting 1 to 2621440
Outputting 0 to 2621441
Outputting 0 to 2621442
Outputting 0 to 2621443
Outputting 0 to 2621444
Outputting 0 to 2621445
Outputting 0 to 2621446
Outputting 0 to 2621447
Outputting 0 to 2621448
Outputting 0 to 2621449
Outputting 0 to 2621450
Outputting 0 to 2621451
Outputting 0 to 2621452
Outputting 0 to 2621453
Outputting 0 to 2621454
Outputting 0 to 2621455
Outputting 0 to 2621456
Outputting 0 to 2621457
Outputting 0 to 2621458
Outputting 0 to 2621459
9) Read byte = 31 from 2097161
10) Read byte = 1 from 2097162
11) Read byte = 29 from 2097163
12) Read byte = 31 from 2097164 Reading CAL File
Configuring Chip At (0,0).
Configuring Chip At (0,1).
Configuring Chip At (1,0).
Configuring Chip At (1,1).
Configuring Chip At (2,0).
Configuring Chip At (2,1).
Configuring Chip At (3,0).
Configuring Chip At (3,1).

Outputting 1 to 2621440
Outputting 0 to 2621441
Outputting 0 to 2621442
Outputting 0 to 2621443
Outputting 0 to 2621444
Outputting 0 to 2621445
Outputting 0 to 2621446
Outputting 0 to 2621447
Outputting 0 to 2621448
Outputting 0 to 2621449
Outputting 0 to 2621450
Outputting 0 to 2621451
Outputting 0 to 2621452
Outputting 0 to 2621453
Outputting 0 to 2621454
Outputting 0 to 2621455
Outputting 0 to 2621456
Outputting 0 to 2621457
Outputting 0 to 2621458
Outputting 0 to 2621459

9) Read byte = 29 from 2097161
10) Read byte = 29 from 2097162
11) Read byte = 21 from 2097163
12) Read byte = 29 from 2097164 Reading CAL File
Configuring Chip At (0,0).
Configuring Chip At (0,1).
Configuring Chip At (1,0).
Configuring Chip At (1,1).
Configuring Chip At (2,0).
Configuring Chip At (2,1).
Configuring Chip At (3,0).
Configuring Chip At (3,1).

Outputting 1 to 2621440
Outputting 0 to 2621441
Outputting 0 to 2621442
Outputting 0 to 2621443
Outputting 0 to 2621444
Outputting 0 to 2621445
Outputting 0 to 2621446
Outputting 0 to 2621447
Outputting 0 to 2621448
Outputting 0 to 2621449
Outputting 0 to 2621450
9) Read byte = 15 from 2097161
10) Read byte = 14 from 2097162
11) Read byte = 4 from 2097163
12) Read byte = 15 from 2097164
Reading CAL File
Configuring Chip At (0,0).
Configuring Chip At (0,1).
Configuring Chip At (1,0).
Configuring Chip At (1,1).
Configuring Chip At (2,0).
Configuring Chip At (2,1).
Configuring Chip At (3,0).
Configuring Chip At (3,1).

Outputting 32 to 2621440
Outputting 32 to 2621441
Outputting 32 to 2621442
Outputting 32 to 2621443
Outputting 62 to 2621444
Outputting 62 to 2621445
Outputting 32 to 2621446
Outputting 0 to 2621447
Outputting 28 to 2621448
Outputting 29 to 2621449
Outputting 12 to 2621450
Outputting 0 to 2621451
Outputting 31 to 2621452
Outputting 1 to 2621453
Outputting 29 to 2621454
Outputting 31 to 2621455
Outputting 29 to 2621456
Outputting 29 to 2621457
Outputting 21 to 2621458
Outputting 29 to 2621459
Outputting 15 to 2621460
Outputting 14 to 2621461
Outputting 4 to 2621462
Outputting 15 to 2621463
0) Read byte = 0 from 2097152
1) Read byte = 0 from 2097153
2) Read byte = 0 from 2097154
3) Read byte = 0 from 2097155
4) Read byte = 0 from 2097156
5) Read byte = 0 from 2097157
6) Read byte = 0 from 2097158
7) Read byte = 0 from 2097159
8) Read byte = 0 from 2097160
9) Read byte = 0 from 2097161
10) Read byte = 0 from 2097162
11) Read byte = 0 from 2097163
12) Read byte = 0 from 2097164
13) Read byte = 0 from 2097165
14) Read byte = 0 from 2097166
15) Read byte = 6 from 2097167
16) Read byte = 7 from 2097168
17) Read byte = 7 from 2097169
18) Read byte = 7 from 2097170
19) Read byte = 7 from 2097171
20) Read byte = 7 from 2097172
21) Read byte = 7 from 2097173
22) Read byte = 15 from 2097174
23) Read byte = 15 from 2097175
24) Read byte = 15 from 2097176
25) Read byte = 15 from 2097177
26) Read byte = 15 from 2097178
27) Read byte = 15 from 2097179
28) Read byte = 15 from 2097180
29) Read byte = 15 from 2097181
Appendix N

PC Neural Network Timing Comparison Program
neurtime.c

Program to estimate the time required by this PC to perform 16, 32-bit multiplications and then 12, 32-bit additions. These values will be compared to those of the CHS2x4 running a stochastic neural network configuration.

RKrans
9/20/94

#include <time.h>
#include <sys/time.h>
#include <stdio.h>

void main()
{
    int i, j, loop, loop2;
    char *start_time, *end_time;
    struct tm* tstruct;
    int sum[4], in[4], weight[4][4];

    /***************************************************************************/
    /* create some random number */
    /***************************************************************************/
    for (i=0; i<4; i++) {
        for (j=0; j<4; j++)
            weight[i][j] = rand();
        sum[i] = rand();
        in[i] = rand();
    } /* for */

    printf("\nstart\tend");
    for (loop=0; loop<10; loop++){

    /***************************************************************************/
    /* record the start time */
    /***************************************************************************/
    _ftime( &tstruct);
    start_time = ctime( &(tstruct.time));
    printf("\n%2s %hu", &start_time[17], tstruct.millitm);

    /***************************************************************************/
    /* multiply and sum the weights */
    /***************************************************************************/
    for (loop2=0; loop2<10000; loop2++){
        for (j=0; j<4; j++){
            for (i=0; i<4; i++){
                sum[j] = sum[j] + in[j]*weight[i][j];
            }
        } /* loop2 */

    /***************************************************************************/
    /* record the end time */
    /***************************************************************************/
    _ftime( &tstruct);
    end_time = ctime( &(tstruct.time));
    printf("\n%2s %hu", &end_time[17], tstruct.millitm);
} /* for loop */
} /* main */
Appendix O

Example CLARE .cfg File
BLOCK toggle
PORT IN clock 0 1 WEST
PORT IN clear 2 1 EAST
PORT OUT q1 2 0 SOUTH
ENDPORTS
CELL 0 0
  FUNCTION DLATCH
  ESOURCE SELF
  X1SOURCE NORTH
  X2SOURCE EAST
ENDCELL
CELL 1 0
  FUNCTION NOR
  ESOURCE WEST
  WSOURCE SELF
  X1SOURCE NORTH
  X2SOURCE EAST
ENDCELL
CELL 2 0
  FUNCTION DLATCH
  WSOURCE SELF
  NSOURCE SELF
  SSOURCE SELF
  X1SOURCE NORTH
  X2SOURCE WEST
ENDCELL
CELL 0 1
  FUNCTION 0 1
  WSOURCE EAST
  ESOURCE WEST
  SSOURCE SELF
  X1SOURCE EAST
  X2SOURCE WEST
ENDCELL
CELL 1 1
  WSOURCE EAST
  ESOURCE WEST
  SSOURCE EAST
ENDCELL
CELL 2 1
  FUNCTION X1ORX2BAR
  WSOURCE EAST
  ESOURCE SOUTH
  SSOURCE SELF
  X1SOURCE EAST
  X2SOURCE WEST
ENDCELL
ENDBLOCK
ENDFILE

[JPG91]
BIBLIOGRAPHY


