12-1993

Evaluation and Implementation of Asynchronous Adders

Poornima Y. Shankarreddy
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
https://scholarworks.wmich.edu/masters_theses/824

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
EVALUATION AND IMPLEMENTATION OF ASYNCHRONOUS ADDERS

by

Poornima Y. Shankarreddy

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

Western Michigan University
Kalamazoo, Michigan
December 1993
EVALUATION AND IMPLEMENTATION OF ASYNCHRONOUS ADDERS

Poornima Y. Shankarreddy, M.S.
Western Michigan University, 1993

The need for high speed addition is forcing digital system designers to trade off space. If speed is used as a criterion to judge adders, then the Ripple Carry Adder (RCA) and the Carry Look Ahead Adders (CLAAD) will rank last and first respectively. On the other hand, if space is used, then the order of these two is reversed. The rest of the adders rank between these two. This paper evaluates RCA, CLAAD, and some of the other well documented adders. Both space and speed are used to judge the merits of each of these, and whether or not they qualify as asynchronous adders. This process led to the design of a new, high speed asynchronous adder. Both probability theory and simulation are used to demonstrate the superiority of new design.
ACKNOWLEDGMENTS

I would like to express my deep appreciation to my advisor, Dr. Abuelyaman, for his excellent guidance and support throughout this study.

This thesis is dedicated to my wonderful mother, B. Yeshodha Shankarreddy, whose support made this endeavor worthwhile.

Poornima Y. Shankarreddy
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Evaluation and implementation of asynchronous adders

Shankarreddy, Poornima Yeshoda, M.S.
Western Michigan University, 1993
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS......................................................... ii

LIST OF TABLES........................................................... iv

LIST OF FIGURES.......................................................... v

CHAPTERS

I. INTRODUCTION.......................................................... 1

II. DESIGN OF AN ASAD.................................................. 3

III. IMPLEMENTATION OF ASAD USING AHPL......................... 17

IV. ANALYZING THE ASAD USING A PROBABILISTIC APPROACH........ 22

V. DESIGN OF A HSCSA................................................... 29

HSCSA Design............................................................... 29

Complexity of CSA and CSCA.............................................. 30

VI. INTERFACING THE MPM, MM AND HSCSA.......................... 35

Simulation Results....................................................... 36

VII. CONCLUSION........................................................... 38

APPENDICES

A. An AHPL Sequence for an 8-Bit ASAD......................... 42

B. An AHPL Sequence for an 8-Bit Microprocessor............. 45

C. An AHPL Sequence for 4-Bits HSCSA.......................... 49

D. An AHPL Sequence for 16-Bits HSCSA......................... 52

REFERENCES............................................................... 59
LIST OF TABLES

1. Comparison of 4-Bit Adders................................. 8
2. I/O Relations of the Lower Block............................ 11
3. Number of Logic Levels in a 4-Bit ASAD................... 13
4. Number of Logic Levels in an 8-Bit ASAD................. 15
5. Probabilities for Different Logic Levels in a 4-Bit Adder........................................ 25
6. Probabilities for Different Logic Levels in an 8-Bit Adder........................................ 27
7. Comparison of the Number of Gates Required by HSCSA and CSCA........................................ 34
8. Comparison of Simulation Results.............................. 37
9. Number of Gates and Logic Levels for Different Adders........................................ 39
10. Space-Speed Product for Different Adders................... 39
11. Normalized Space-Speed Product.............................. 40
# LIST OF FIGURES

1. Addition Using CSA ................................ 5  
2. Ripple Carry Look Ahead Adder .................... 9  
3. Typical Cell of an ASAD ........................... 10  
4. Layout of a 4-Bit ASAD ............................. 12  
5. Handshaking Logic for a 4-Bit ASAD ............... 15  
6. Block Diagram of an 8-Bit ASAD .................... 16  
7. Addition Using CSA ................................ 18  
8. Addition Using CSA for Different Operands ....... 18  
9. State of the Cells to Complete Addition in 2 Logic Levels................................. 23  
10. Number of Gates Required at Level 1 to Compute (a) SUM (b) CARRY ...................... 32  
11. Plot of SSP vs Number of Bits .................... 40
CHAPTER I

INTRODUCTION

The design of efficient adders is a key factor in determining the speed of operation of a microprocessor because addition is very frequently used.

Several synchronous adders have been developed using different techniques [1-6]. One of the disadvantages among these adders is the number of gates and/or logic levels required for completion of addition. This paper will study these disadvantages. In the process, an asynchronous adder (ASAD) is introduced. ASAD is considered asynchronous because the time needed for completion of addition is a function of the addend and the augend. This is not to be confused with an unclocked system, for there is no clock in these systems. Asynchronous systems are clocked but not deterministic whereas synchronous systems are clocked and deterministic. Unclocked systems are neither clocked nor deterministic. ASAD is clocked locally and handshakes with the control unit. Such adders fit nicely in self timed systems.

In Chapter II the design of an asynchronous adder will be discussed. Simulation of the ASAD using Universal
Hardware Programming Language (UAPHL) will be presented in Chapter III. Probability theory will be used to analyze the different cases of 4 and 8 bit adders in Chapter IV. Modification of CSA to operate in an asynchronous mode will be discussed in Chapter V. In Chapter VI, an 8-bit microprocessor module will be introduced. An average frequency of addition operations, required by the microprocessor, will also be computed. These addition operations are performed by the built-in, synchronous adder. The time required for each of the addition operations will be compared against the time needed by the HSCSA for the same operands. Furthermore, the memory module (MM), HSCSA and the microprocessor module (MPM) written and simulated using UAPHL will be linked and tested. Finally, a conclusion and future improvements will be presented in Chapter VII.
CHAPTER II

DESIGN OF AN ASAD

Adders play a major role in determining the operating speed of microprocessors. This fact led to the design of high speed adders, a number of which have been developed using different techniques [1-6].

The most widely used adders are the ripple carry adder (RCA) [1], and the carry look ahead adder (CLAAD) [2]. The former is inexpensive and slow, while the latter is fast and expensive. Another adder of interest is the Conditional Sum Adder (CSA) [3]. In a N-bit CSA, there are N full adders forming the leaf nodes of a binary tree. The outputs of a leaf node are the sum and carry-out. The root node represents the overall sum. Outputs from the intermediate nodes represent either a convergent sum or a value which leads to it [4]. It has been shown by Gosling [5] that the CSA is superior to other high speed adders in several quantitative respects. The number of gates, however, is the main disadvantage of the CSA. In a CSA, two sum and carry-output combinations are evaluated for every pair of bits. One combination corresponds to 0 carry-in, whereas the other corresponds to the carry-in being 1. To
trate how CSA operates, let us consider the following example. The Boolean equations 1-15 are used for evaluating sum and carry values. The augend $A_i$ is added to the addend $B_i$ where $i$ represents the bit index, which sequences from 1 to $M$, and $M$ is the number of bits. $S_{j,i}$ corresponds to the sum and $C_{j,i}$ the carry, where $j$ is the level number. The superscript of 0 or 1 implies whether a carry-in of 0 or 1 is assumed. The Boolean equations used to determine sum and carry at level 1 are as follows:

\[
\begin{align*}
S^0_{1,i} &= \bar{A}_i B_i + A_i \bar{B}_i & 1 \\
S^1_{1,i} &= A_i B_i + \bar{A}_i \bar{B}_i & 2 \\
C^0_{1,i} &= A_i \bar{B}_i & 3 \\
C^1_{1,i} &= A_i + B_i & 4
\end{align*}
\]

The second level equations are determined from the first level pairs from right to left. The Boolean equations for the sum and carry are as follows:

\[
\begin{align*}
S^0_{2,K} &= S^1_{1,K} C^0_{1,K-1} + S^0_{1,K} \bar{C}^0_{1,K-1} & 5 \\
S^1_{2,K} &= S^1_{1,K} C^1_{1,K-1} + S^0_{1,K} \bar{C}^1_{1,K-1} & 6 \\
S^0_{2,K-1} &= S^0_{1,K-1} & 7 \\
S^1_{2,K-1} &= S^1_{1,K-1} & 8 \\
C^0_{2,K} &= C^1_{1,K} C^0_{1,K-1} + C^0_{1,K} \bar{C}^0_{1,K-1} & 9 \\
C^1_{2,K} &= C^1_{1,K} C^1_{1,K-1} + C^0_{1,K} \bar{C}^1_{1,K-1} & 10
\end{align*}
\]

where $K = 2^{j-1}$, $2 \times 2^{j-1}$, $3 \times 2^{j-1}$ ...... $M$

The third level equations are determined from second level quartets again from right to left.

\[
S^a_{3,K} = S^1_{2,K} C^a_{2,K-2} + S^0_{2,K} \bar{C}^a_{2,K-2}
\]
\begin{align*}
S^a_{3,k-1} &= S^1_{2,k-1}C^a_{2,k-2} + S^0_{2,k-1}C^a_{2,k-2} \\
S^a_{3,k-2} &= S^a_{2,k-2} \\
S^a_{3,k-3} &= S^a_{2,k-3} \\
C^a_{3,k} &= C^1_{2,k}C^a_{2,k-2} + C^0_{2,k}C^a_{2,k-2}
\end{align*}

where the superscript "a" is equal to 0 or 1.

The first (second) line for each level corresponds to the 0 (1) carry-in. The first level nodes are the leafs and the third level node is the root.

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
\end{array}
\]

Figure 1. Addition Using CSA.

Two sets of results from level 1 are merged as shown in level 2 (refer to Figure 1). The carry into the least significant cell (cell 0) is assumed 0, hence the first line is taken as the output of cell 0. The carry-out on this line is 0, therefore the first line of the second least significant cell (cell 1) is considered (i.e., 10, the 11 which corresponds to the carry-in being a 1 is discarded). To merge the outputs of the second most significant cell (cell 2) and the most significant cell (cell 3),
one has to know the carry-in from cell 1. Since the carry-out of cell 1, which is also the carry into cell 2 is unknown at this level, both lines for cells 2 and 3 will be considered. Values obtained at level 2 are merged to form level 3. Since the carry-out of cell 1 is now known to be 1 (see level 2 column 4), one has to take the second line as the output of cell 2, thereby discarding the first line. Similarly, the second line is taken as output of cell 3.

Martin and Hunagel [6] improved the CSA by observing early completion of sum and called it a Conditional Sum early Completion Adder (CSCA). CSCA design is based on the computation of "conditional" sums, carries, and column completion detection logic. The number of logic levels in CSCA is reduced, compared to that of CSA, at the expense of more gates. The equations for evaluating sum and carry are as shown below [3]. For each stage in a CSCA $S_{j,i}$ is the sum that is incomplete with no carry-in value and is the completed sum value when the column sum is complete. $S^*_{j,i}$ is the sum carry-in value that is incomplete and is the completed sum value when the column sum is complete. $C_{j,i}$ is the level no-carry-out value and $C^*_{j,i}$ is the level carry-out value. The addend $A_i$ is being added to the augend $B_i$ to provide the sum at level 1, where "i" is the bit index.

\[
S_{j,i} = \overline{A}_iB_i + A_i\overline{B}_i \quad 16
\]

\[
S^*_{j,i} = A_iB_i + \overline{A}_i\overline{B}_i \quad 17
\]

\[
C_{j,i} = \overline{A}_i\overline{B}_i \quad 18
\]
The general level equations which are applied to every column position after the first level are shown below:

\[
\begin{align*}
C_{1,i}^* &= A_iB_i & \text{(19)} \\
\bar{C}_{i,i} &= A_i + B_i & \text{(20)} \\
\bar{C}_{1,i}^* &= \bar{A}_i + \bar{B}_i & \text{(21)}
\end{align*}
\]

\[
\begin{align*}
S_{j+1,i} &= C_{j,i-nj}^*S_{j,i} + \bar{C}_{j,i-nj}S_{j,i} + S_{j,i}S_{j,i} & \text{(22)} \\
S_{j+1,i}^* &= C_{j,i-nj}S_{j,i} + \bar{C}_{j,i-nj}^*S_{j,i} + S_{j,i}S_{j,i} & \text{(23)} \\
C_{j+1,i} &= C_{j,i} + C_{j,i-nj}\bar{C}_{j,i} & \text{(24)} \\
C_{j+1,i}^* &= C_{j,i}^* + C_{j,i-nj}\bar{C}_{j,i} & \text{(25)} \\
\bar{C}_{j+1,i} &= \bar{C}_{j,i} (\bar{C}_{j,i-nj} + C_{j,i}) & \text{(26)} \\
\bar{C}_{j+1,i}^* &= \bar{C}_{j,i}^* (\bar{C}_{j,i-nj}^* + C_{j,i}) & \text{(27)}
\end{align*}
\]

where \( nj = 2^{j-1}, j = 1,2, \ldots \), is the check bit offset for each level of \( j \), which is also the carry completion test for each bit. Detection logic is required at each level to check for completion of addition. Completion is true when \( S_{j,i} \) is equal to \( S_{j,i}^* \) for all values of the bit index for a particular level of \( j \).

A comparison between the number of gates and logic levels for 4-bit versions of six addition techniques is given in Table 1. The number of gates and logic levels for both CSA and CSCA is computed from equations 1-27.

Theoretically speaking, the CSCA makes a good asynchronous adder, but the fact that it requires a large number of gates makes it less favorable. On the other hand, the CSA is well suited for the synchronous mode of operation. Hence modification is required to operate it in an
asynchronous mode. This will be dealt with in Chapter V.

### Table 1

**Comparison of 4-Bit Adders**

<table>
<thead>
<tr>
<th>Adders</th>
<th>Number of gates</th>
<th>Number of logic levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCA</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>CLAAD</td>
<td>104</td>
<td>2</td>
</tr>
<tr>
<td>RCLAD</td>
<td>32</td>
<td>4</td>
</tr>
<tr>
<td>CSA</td>
<td>80</td>
<td>6</td>
</tr>
<tr>
<td>CSCA</td>
<td>184</td>
<td>*5</td>
</tr>
<tr>
<td>ASAD</td>
<td>19</td>
<td>*3.4</td>
</tr>
</tbody>
</table>

* Average number of logic levels

The CLAAD requires only 2 logic levels, hence it is a good candidate for asynchronous addition. Another candidate is the RCA which qualifies because it requires the fewest number of gates.

If the advantages of the RCA and the CLAAD are combined to build a 4-bit adder, RCLAD, as shown in Figure 2, a total of 32 gates arranged in 4 logic levels will suffice [7]. While the reduction in the number of logic levels compared to a 4-bit RCA is significant (almost 50%), the number of gates remains relatively high. Though this number is less than that of the equivalent CLAAD [7], a further improvement will be significant.

Without loss of generality, we will first design a 4-bit Asynchronous Adder (ASAD), then generalize the technique to any number of bits.
Next we will demonstrate that the 4-bit asynchronous adder (ASAD) possesses the advantages of the RCA and the CLAAD. It will be shown that the ASAD uses 20 gates arranged in 8 logic levels. However, the latter number is necessary for only 12.5% of the addition cases. The remaining 87.5% cases require a maximum of 4 logic levels. On the average, the ASAD will be shown to require only 3.4 logic levels. This fact motivated the design of ASAD. The speed of ASAD is significant only when it is clocked at high frequency.

Before we proceed with the design of the ASAD, the following symbols will be defined:

**Definition 1**

- $S^H_i$: the sum from half adder block $i$
- $S^A_i$: the sum from the ASAD cell $i$
- $C^H_{i+1}$: the carry out from half adder block $i$
- $C^A_{i+1}$: the carry out from the ASAD cell $i$
C^A_i : the carry in from the ASAD cell i-1.

Figure 3 shows the layout of a typical ASAD cell. This cell consists of two blocks; the upper is a simple half adder while the lower represents the necessary logic for making the cell a full adder. The relationship between the inputs and outputs of the lower block is shown in Table 2.

Next, we redefine the well known carry generate and carry propagate terms in a more convenient way. We will also define an idle state.
Table 2

I/O Relations of the Lower Block

<table>
<thead>
<tr>
<th>ROW No.</th>
<th>$C^H_{i+1}$</th>
<th>$S^H_i$</th>
<th>$C^A_i$</th>
<th>$S^A_i$</th>
<th>$C^{A}_{i+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Definition 2

A cell "$i$" is in the Carry Propagate State (CPS) if

$$S^H_i = 1 \text{ and } C^H_{i+1} = 0.$$  

Definition 3

A cell "$i$" is in the Carry Generate State (CGS) if

$$C^H_{i+1} = 1.$$  

Definition 4

A cell "$i$" is in the IDle State (IDS) if it is neither in the CPS nor in the CGS state.

From the above definitions and Table 2 it follows that cell $i$ will be in the IDS state for rows 0 and 1. Further, it will be in the CPS state for rows 2 and 3; otherwise, Cell 2 will be in the CGS state. The complete layout of the ASAD is shown in Figure 4. From this figure, the number of gates is 20 as stated earlier.

All the cells of the ASAD will receive inputs simultaneously. Each cell's output will be ready after 2 logic
level delays, except of course in cases of carry-in from the next least significant stage.

Figure 4. Layout of a 4-Bit ASAD.

Table 3 relates the outputs of upper and lower blocks to the optimal number of logic levels. Outputs $S_0^A$, $C_1^A$, and $S_1^A$ are not included because they converge after 2 logic levels (refer to Figure 4). $C_4^A$ is also omitted because its value will be 1 after 1 logic level delay following 1 for $C_4^H$. The only other situation in which $C_4^A$ will be 1 is shown in rows 11 through 15 of the second column. From these rows, it follows that:

$$C_4^A = S_3^H C_3^H + S_3^H S_2^H C_2^A$$  \[28\]

From rows 11 to 15 (column 5) it could be seen that $C_3^H$ could be replaced by $C_3^A$. Therefore equation 28 could be rewritten as follows:
Table 3
Number of Logic Levels in a 4-Bit ASAD

<table>
<thead>
<tr>
<th>ROW No.</th>
<th>$S_3^n(S_3^*)$</th>
<th>$C_3^n(C_3^*)$</th>
<th>$S_2^n(S_2^*)$</th>
<th>$C_2^*$</th>
<th>NUMBER OF LEVELS</th>
<th>STATE OF CELL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0(0)</td>
<td>0(0)</td>
<td>0(0)</td>
<td>0(0)</td>
<td>0</td>
<td>IDS</td>
</tr>
<tr>
<td>1</td>
<td>0(0)</td>
<td>0(0)</td>
<td>0(1)</td>
<td>0(1)</td>
<td>1</td>
<td>IDS</td>
</tr>
<tr>
<td>2</td>
<td>0(0)</td>
<td>0(0)</td>
<td>1(1)</td>
<td>0(0)</td>
<td>2</td>
<td>CPS</td>
</tr>
<tr>
<td>3</td>
<td>0(1)</td>
<td>0(1)</td>
<td>1(0)</td>
<td>1(0)</td>
<td>2</td>
<td>CPS</td>
</tr>
<tr>
<td>4</td>
<td>0(1)</td>
<td>1(1)</td>
<td>0(0)</td>
<td>0(0)</td>
<td>2</td>
<td>CPS</td>
</tr>
<tr>
<td>5</td>
<td>0(1)</td>
<td>1(1)</td>
<td>0(1)</td>
<td>0(1)</td>
<td>4</td>
<td>CGS</td>
</tr>
<tr>
<td>6</td>
<td>0(1)</td>
<td>1(1)</td>
<td>1(1)</td>
<td>0(1)</td>
<td>4</td>
<td>CGS</td>
</tr>
<tr>
<td>7</td>
<td>0(1)</td>
<td>1(1)</td>
<td>1(0)</td>
<td>1(0)</td>
<td>4</td>
<td>CGS</td>
</tr>
<tr>
<td>8</td>
<td>1(1)</td>
<td>0(0)</td>
<td>0(0)</td>
<td>0(0)</td>
<td>2</td>
<td>IDS</td>
</tr>
<tr>
<td>9</td>
<td>1(1)</td>
<td>0(0)</td>
<td>0(1)</td>
<td>0(1)</td>
<td>4</td>
<td>IDS</td>
</tr>
<tr>
<td>10</td>
<td>1(1)</td>
<td>0(0)</td>
<td>1(1)</td>
<td>1(1)</td>
<td>2</td>
<td>CPS</td>
</tr>
<tr>
<td>11</td>
<td>1(0)</td>
<td>0(0)</td>
<td>1(0)</td>
<td>1(0)</td>
<td>8</td>
<td>CPS</td>
</tr>
<tr>
<td>12</td>
<td>1(0)</td>
<td>1(1)</td>
<td>0(0)</td>
<td>0(0)</td>
<td>4</td>
<td>CPS</td>
</tr>
<tr>
<td>13</td>
<td>1(0)</td>
<td>1(1)</td>
<td>1(0)</td>
<td>1(0)</td>
<td>4</td>
<td>CGS</td>
</tr>
<tr>
<td>14</td>
<td>1(0)</td>
<td>1(1)</td>
<td>1(0)</td>
<td>1(0)</td>
<td>4</td>
<td>CGS</td>
</tr>
<tr>
<td>15</td>
<td>1(0)</td>
<td>1(1)</td>
<td>1(0)</td>
<td>1(0)</td>
<td>4</td>
<td>CGS</td>
</tr>
</tbody>
</table>

$c^* = S_3^n C_3^* + S_3^n S_2^n C_2^*$

From this equation and Figure 4, it follows that $c_4^*$ requires a maximum of 8 logic levels. This situation represents the other extreme case, therefore, $C_4^n (C_4^*)$ is also not included.

Listed under $S_3^n$, $C_3^n$, $S_2^n$ and $C_2^*$ are all the possible combinations of 4 variables. The values relating to cell 2 are shown in columns 4 and 5. These columns indicate that cell 2 will be in the IDS state for each of the rows 0,1,8 and 9. For rows 2,3,10 and 11 the cell will be in the CPS state, otherwise it will be in the CGS state.

Except for the situation when cell 2 is in the CPS, cell 3 is independent of cells 1 and 0. Hence, the 4-bit
ASAD will be logically reconfigured into two independent 2-input adders each requiring a maximum of 4 logic levels. Consequently, only 4 logic levels are needed for the output of the ASAD to converge.

Rows 2 and 10 indicate a CPS state for cell 2 while its carry-in is 0. In this case, cell 3 is also independent of cells 1 and 0, and the sum will converge after only 2 logic levels. The only case in which cell 3 is affected by cell 1 and/or cell 0 is shown in rows 3 and 11. The solid line in Figure 4 reflects that 8 propagation delays are needed for cases relating to rows 3 and 11. Hence they are treated as special cases which take place when:

\[ C_3^n \times S_2^n \times C_2^A = 1. \]

Figure 5 shows the circuit needed to isolate these cases. The output of this circuit could be viewed as a hand-shaking mechanism between the control unit and the adder.

From Table 3, it follows that 14 out of the 16 cases indicate that addition can be completed after 4 logic level delays (i.e. 87.5% of the time). Therefore, the 4-bit ASAD completes addition in approximately half the time needed by the equivalent RCA. The average number of logic levels required will be:

\[
= 0.125 \times 8 + 0.875 \times 3 \\
= 3.63
\]
Figure 5. Handshaking Logic for a 4-Bit ASAD.

Table 4
Number of Logic Levels in an 8-Bit ASAD

<table>
<thead>
<tr>
<th>HSL2</th>
<th>LSC</th>
<th>HSL1</th>
<th>NUMBER OF LOGIC LEVELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8-16</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>8-12</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8-12</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>8-16</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>8-12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8-12</td>
</tr>
</tbody>
</table>

In Figure 6, we show how an 8-bit ASAD may be constructed using two 4-bit ASAD's. The two handshaking lines (HSL1 and HSL2) along with the carry-out from the least significant 4-bit ASAD (LSC) will now be used to detect early completion of addition process. Table 4 shows the number of logic levels required for every combination of
HSL1, HSL2 and LSC.

Analysis of more than 8-bit is complicated, hence another approach will be discussed in Chapter III.
CHAPTER III

IMPLEMENTATION OF ASAD USING AHPL

Analysis of more than 8-bit is now possible, thanks to the availability of Computer Aided Design (CAD) tools. In this chapter, an 8-BIT ASAD will be designed using one of the known CAD tools, the AHPL, a Register Transfer Language (RTL), invented by Hill [8]. One of the advantages of AHPL is that there is a 1 to 1 correspondence between the AHPL sequence and the hardware needed. This fact makes the design, functional simulation and testing of ASAD extremely simple. Furthermore, AHPL offers a modular design environment in which extension of the adder to any number of bits is fairly simple. First, some of the concepts used hereafter will be restated. Other concepts will also be introduced. The AHPL sequence for the adder module will then be presented.

Definition 5

A half adder cell is in the Carry Propagate State, CPS, if one of its inputs is the complement of the other.

Definition 6

A half adder cell is in the Carry Generate State, CGS, if both of its inputs are logical 1's.
Definition 7

A half adder cell is in the Idle State, IDS, if it is neither in the CGS nor in the CPS.

Let us consider the following examples (Figures 7 and 8):

\[
\begin{array}{cccccc}
0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 \\
\end{array}
\]  

Figure 7. Addition Using CSA.

In Figure 7, all the carry-outs of level 1 are zero's. For this situation to occur, every cell has to be in either the IDS or CPS. Therefore, addition will complete in 2 logic levels.

\[
\begin{array}{cccccc}
1 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

Figure 8. Addition Using CSA for Different Operands.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The other case of early completion is when the carry-out is independent of the carry-in for every bit (refer to level in Figure 8). This situation occurs when every cell is in either the IDS or CGS and addition will complete in at most 3 logic levels.

In general, early completion is detected at level 1 by checking that \( C_{0,i}^0 = 0 \) for all "i", and at any level thereafter by checking that \( C_{0,i}^1 = C_{1,i}^1 \) for all "i".

The following algorithm manifests all the early completion cases.

**Definition 8**

A carry is said to be first generation carry (FGC), second generation carry (SGC), etc. if it ripples through 1, 2, etc. cells respectively. The algorithm for an 8-bit ASAD is outlined as follows:

1. \( \text{SUM} = \text{ADDEND} + \text{AUGEND}; \)
   \( \text{CARRY} = \text{ADDEND} \times \text{AUGEND}; \)
   \( \rightarrow \text{NO CARRY GENERATED}/\text{END} \)

2. \( \text{SUM} = \text{SUM} + \text{FGC}; \)
   \( \text{UPDATE CARRY}; \)
   \( \rightarrow \text{NO SGC}/\text{END} \)

3. \( \text{SUM} = \text{SUM} + \text{SGC} \)
   \( \text{UPDATE CARRY}; \)
   \( \rightarrow \text{NO TGC}/\text{END} \)

4. \( \text{SUM} = \text{SUM} + \text{TGC} \)
   \( \text{UPDATE CARRY}; \)
   \( \rightarrow \text{NO FoGC}/\text{END} \)

5. \( \text{SUM} = \text{SUM} + \text{FOGC} \)
   \( \text{UPDATE CARRY}; \)
   \( \rightarrow \text{NO FiGC}/\text{END} \)

6. \( \text{SUM} = \text{SUM} + \text{FiGC} \)


UPDATE CARRY;
-->(NO SiGC)/END

7. SUM = SUM + SiGC
   UPDATE CARRY;
   -->(NO SeGC)/END

8. SUM = SUM + SeGC
   UPDATE CARRY;

9. END.

The first step which is derived from Figure 7, represents the case in which all the cells are in either: (a) the IDS state, (b) the CPS state, or (c) a combination of IDS and CPS states.

If execution of the algorithm completes at step 2, which is derived from Figure 8, then there is at least one FGC and no SGC, TGC, etc. When execution finishes at the end of step 3, then there is at least one SGC and no TGC and so on. Each step requires 2 logic levels. Completion of addition in step 1 would be the best case. The average case completes at step 4. Step 8 represents the worst case. The algorithm is favorable if more than 50% of the addition process converges at the end of step 4. The complete AHPL sequence for the algorithm is given in Appendix A.

The number of gates needed to realize a 4-bit ASAD using AHPL is about 70, for 8-bit it doubles and so on. In contrast the number of gates needed for a 4-bit CLAAD is 104 and it grows exponentially with the number of bits.
The CSCA has the advantage of detecting early completion of addition at the expense of more gates (104 gates for a 4-bit adder).

In the next chapter we will discuss the probability that addition completes in step $i$ ($1 \leq i \leq N$), where $N$ is the total number of AHPL steps.
CHAPTER IV

ANALYZING THE ASAD USING A PROBABILISTIC APPROACH

In a CSA with $N = 2^k$ bits, there will be $K$ stages. If the execution of every stage requires 1 clock period, then $\log_2 N$ is the time needed for completion of addition. This requires the CSA to have $2N$ adders at stage 1 for simultaneous computation of the carry and no carry values. Otherwise, the time needed at stage 1 will be doubled. Furthermore every CSA level requires at least 2 logic levels.

Martin gave a probabilistic $\log_2 (\log_2 N)$ convergence time for CSCA. For small values of $N$, we find the convergence time to be $\log_2 \log_2 N + 1$. This is because any tree with $2^k$ nodes will have $K+1$ stages. The worst case for the CSCA is equal to the CSA convergence time.

For the ASAD, we will first give the probability that addition completes at certain steps of the AHPL, then find an overall average for the convergence time.

The approach we used in Chapter II to determine the average number of logic levels for addition worked nicely for 4-bits. To analyze 8-bit adders, one has to work with $2^8$ different combinations. This approach is, however, cumbersome for more than 8 bits.

22
To determine the average number of levels, we will take advantage of cell states as given in definitions 5, 6 and 7. Consequently, only \(2N\) comparisons will be needed for an \(N\) bit adder as opposed to \(2^N\) as in Chapter II.

To illustrate this approach, let us consider timing analysis for a 4-bit ASAD. The 4 blocks in Figure 9 represent the state of each of the 4 cells. Block \(i\) represents the state of cell \(i\), \(1 \leq i \leq 4\), which may be in the IDS, the CGS or the CPS.

<table>
<thead>
<tr>
<th>Block 3</th>
<th>Block 2</th>
<th>Block 1</th>
<th>Block 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>IDS/CPS</td>
<td>IDS/CPS</td>
<td>IDS/CPS</td>
</tr>
</tbody>
</table>

Figure 9. State of the Cells to Complete Addition in 2 Logic Levels.

If addition is to complete at step 1 (refer to Appendix A), then none of the three least significant cells should produce a carry which affects its next most significant neighbor. This corresponds to the situation in which addition completes after 2 logic levels. To satisfy this requirement, the states of the ASAD cells have to be matching those shown inside the blocks in Figure 9. The "X" implies that the state of the cell is insignificant.

To find the probability that the cells will be configured as in Figure 9, one has to analyze each cell separately. In Figure 8, the state of cell 3 is an "X," hence
it does not affect the required probability. Each of the other 3 cells have to be in the IDS or the CPS state. The probability that a cell will be in the IDS is 1/4 (input to a cell would be \{00, 01, 10, or 11\}). Similarly, the probability that a cell will be in the CPS is 1/2. Therefore the probability that a cell is either in the CPS or IDS is 1/2 + 1/4 = 3/4. Consequently, the probability that each of the 3 cells will be in either the IDS or the CPS is \((3/4)^3 = 27/64\) or 108/256.

Since each AHPL step corresponds to 2 logic levels and addition may or may not use both levels, a 4-bit ASAD may finish after 2, 3, 4, 5, 6, 7 or 8 logic levels.

Table 5 relates the logical configuration of the cells to the number of logic levels and their respective probabilities.

Probabilities for the different levels were obtained using the same approach as discussed in the previous paragraph.

One can obtain the average number of logic levels required for completing addition in a 4-bit adder as follows:

\[
2 \times \frac{108}{256} + 3 \times \frac{56}{256} + 4 \times \frac{28}{256} + 5 \times \frac{32}{256} + 6 \times \frac{16}{256} + 7 \times \frac{8}{256} + 8 \times \frac{8}{256}
\]

\[
= 3.4
\]

This number is more accurate than the 3.63 obtained in Chapter II.
Table 5
Probabilities for Different Logic Levels in a 4-Bit Adder

<table>
<thead>
<tr>
<th>Level</th>
<th>Probability</th>
<th>State of cells</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>108/25</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>8/256</td>
<td>IDS/CGS</td>
</tr>
<tr>
<td></td>
<td>12/256</td>
<td>IDS/CGS</td>
</tr>
<tr>
<td>3</td>
<td>18/256</td>
<td>IDS/CGS</td>
</tr>
<tr>
<td></td>
<td>8/256</td>
<td>CPS</td>
</tr>
<tr>
<td></td>
<td>8/256</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>2/256</td>
<td>CPS</td>
</tr>
<tr>
<td>4</td>
<td>24/256</td>
<td>CPS</td>
</tr>
<tr>
<td></td>
<td>4/256</td>
<td>CPS</td>
</tr>
<tr>
<td>5</td>
<td>16/256</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>16/256</td>
<td>IDS/CGS</td>
</tr>
<tr>
<td>6</td>
<td>16/256</td>
<td>CPS</td>
</tr>
<tr>
<td>7</td>
<td>8/256</td>
<td>IDS/CGS</td>
</tr>
<tr>
<td>8</td>
<td>8/256</td>
<td>CPS</td>
</tr>
</tbody>
</table>

The above analysis could be extended to 8-bit adder (refer to Figure 5). For the 4-bit adder, a detailed analysis was necessary to determine the required number of logic levels. A similar approach will not be used for an 8-bit adder because it becomes more complicated as the number of bits increases. Therefore we will concentrate on the best, average, and worst cases. Using Figure 5, the
probability that HSL1, HSL2 and LSC are 1's is calculated as follows:

The probability that HSL1 is 1 \( \{ P(HSL1=1) \} \)

\[ \{ i.e \ ( C^h_3 * S^h_2 * C^a_2 ) = 1 \} \]

\[ P(HSL1=1) = P(C^h_3) * P(S^h_2) * P(C^a_2) \]
\[ = \frac{3}{4} * \frac{1}{2} * \frac{3}{8} \]
\[ = \frac{9}{64} \]

Consequently

\[ P(HSL1=0) = 1 - \frac{9}{64} \]
\[ = \frac{55}{64} \]

Similarly \( P(HSL2=1) \) is

\[ \{ i.e \ ( C^h_3 * S^h_2 * C^a_2 ) = 1 \} \]

\[ P(HSL2=1) = \frac{3}{4} * \frac{1}{2} * \frac{3}{8} \]
\[ = \frac{9}{64} \]

Therefore

\[ P(HSL2=0) = 1 - \frac{9}{64} \]
\[ = \frac{55}{64} \]

\( P(LSC=1) \) is computed observing that the events are independent (refer to Figure 3)

\[ P(C^a_4=1) = P(C^h_4 + S^h_3 * C^h_3 + S^3_2 * C^2_2 + S^2_2 * C^1_2) \]
\[ = P(C^h_4) + P(S^h_3) * P(C^h_3) + P(S^3_2) * P(S^2_2) \]
\[ * P(C^2_2) + P(S^2_2) * P(S^1_2) * P(C^1_2) \]
\[ = \frac{1}{4} + \frac{1}{2} * \frac{1}{4} + \frac{1}{2} * \frac{1}{2} * \frac{1}{4} \]
\[ = \frac{1}{4} + \frac{1}{2} * \frac{1}{2} * \frac{1}{2} * \frac{1}{4} \]

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
= 15/32 or 30/64

OR

\[ P(C^t_4 = 0) = 1 - 15/32 \text{ or } 34/64 \]

Table 6 shows probabilities for all possible combinations of HSL2, LSC and HSL1 and the corresponding number of logic levels.

Table 6

<table>
<thead>
<tr>
<th>HSL2</th>
<th>LSC</th>
<th>HSL1</th>
<th>No. of Logic Levels</th>
<th>Average No. logic levels</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>0.3923</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>12</td>
<td>0.0642</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>8-12</td>
<td>10</td>
<td>0.3461</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8-12</td>
<td>10</td>
<td>0.0566</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>6</td>
<td>0.0624</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>8-16</td>
<td>12</td>
<td>0.0155</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>8-12</td>
<td>10</td>
<td>0.0566</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8-12</td>
<td>10</td>
<td>0.0092</td>
</tr>
</tbody>
</table>

\[ P(\text{NUMBER OF LEVELS } \leq 4) = P(\text{HSL2=HSL1=LSC=0}) \]

\[ = P(\text{HSL2}=0) \times P(\text{HSL1}=0) \times P(\text{LSC}=0) \]

\[ = 55/64 \times 3/64 \times 55/64 \]

\[ = 0.3923 \]

Probabilities for the other levels are obtained using the same approach.

Therefore the average number of logic levels required to complete the addition in an 8-bit adder is:

\[ = 3 \times 0.3923 + 6 \times 0.0642 + 10 \times 0.0566 + 14 \times 0.0092 \]
= 7.3

In the next chapter the synchronous CSA will be modified to operate as an asynchronous CSA.
CHAPTER V

DESIGN OF A HSCSA

The fact that ASAD is asynchronous makes it useful for high speed addition of more than 32-bits. Due to the technology constraints, however, one cannot build the described ASAD. The clock period for an ASAD should be based on 2 logic level delays. If one assumes an average propagation delay of 10ns per gate, then a 2 logic level function requires 20ns. This means that a clock of 50MHz is required. Since 50MHz processors are not feasible at the time this manuscript is written; an alternative will be discussed. In this section we will develop an early completion detection logic for the CSA which will be called HSCSA (High Speed Conditional Sum Adder). It will be shown that HSCSA requires only half the number of gates of CSCA and has better convergence time when compared to CSCA. In the next section, the number of gates for 4, 8, 16, 32 and 64-bit adders will be calculated for CSCA and HSCSA.

HSCSA Design

HSCSA is basically a CSA with additional logic for the detection of early completion. The following proposition
gives the conditions which must be met for addition to complete.

**Proposition 1**

In a $2^n$ CSA, addition completes at level $j$, where $j > 1$, if and only if:

$$S^0_{1,p} = 0 \text{ where } p = 2^{j-1} \cdot r; r = 1, 2, 3, ..., (2^{n-j+1} - 1)$$

**Proof:**

$S^0_{1,i} = 0$ implies that the corresponding adder is in the CGS or IDS state. Therefore, $C^0_{1,i}$ is equal to $C^1_{1,i}$ and both carries will result in the same output. The circuit for testing early completion is extremely simple and requires a maximum of 10 gates, up to 64 bit adders.

This proposition excludes checking for completion of addition at level 1. The following condition must be met for addition to complete at level 1:

$$C^0_{1,i} = 0 \text{ for } i = 1, 2, 3 ... M$$

**Complexity of CSA and CSCA**

The number of gates for a 4-bit CSA is calculated using equations 1-15. For example, the number of gates required to compute $S^0_{1,0}$ and $S^1_{1,0}$ at level 1 for a carry-in of 0 (1) is 5 (3) {using equations 1 and 2}. Looking at equation 2, it is obvious that $C^0_{1,0}$ is available. To compute $C^1_{1,0}$ (equation 4), only a single gate is required. Therefore, the total number of gates at level 1 is 9 for 1-bit, and 36 for 4-bits.
At level 2, cell 0 which is the Least Significant Cell (LSGC) is merged with cell 1, the Next Least Significant Cell (NLSC), and cell 3, which is the Most Significant Cell (MSC) is merged with cell 2 the Next Most Significant Cell (NMSC). The selection between $S_{0}^{1,0}$ and $S_{1}^{1,0}$ is made depending on the carry-in. This requires a maximum of 4 gates to realize (see RHS of Figure 10a). Similarly $C_{0}^{1,0}$ or $C_{1,0}$ is selected using the carry-in, which in turn requires 3 gates to realize (see RHS of Figure 10b). The Next Least Significant (NLS) sum and carry are selected using $C_{0}^{1,0}$ or $C_{1,0}$. The required logic is shown in the RHS of Figure 10a with the following transformation:

\[
\begin{align*}
C^{0} & \rightarrow C_{1,0}^{0}/C_{1,0}^{1}, \quad S_{1,0}^{1} \rightarrow S_{1,1}^{1}, \quad S_{0,0}^{1} \rightarrow S_{0,1}^{1}, \\
C_{1,0}^{0} & \rightarrow C_{1,1}^{0}, \quad C_{1,0}^{1} \rightarrow C_{1,1}^{1}.
\end{align*}
\]

Since the carry-in to cell 2 is unknown at this level, no selection can be made between $S_{2,1}$ and $S_{1,2}$. Hence, both are considered. The Most Significant Sum (MSS) is selected using $C_{0,1}$ and/or $C_{1,2}$. In Figure 10, the following transformation should take place:

\[
\begin{align*}
C^{0} & \rightarrow C_{1,2}^{0}, \quad C^{1} \rightarrow C_{1,2}^{1}, \quad S_{1,0}^{1} \rightarrow S_{1,3}^{1}, \\
S_{0,0}^{1} & \rightarrow S_{0,3}^{1}, \quad C_{0,0}^{0} \rightarrow C_{0,3}^{0}, \quad C_{1,0}^{1} \rightarrow C_{1,3}^{1}.
\end{align*}
\]

Therefore, a maximum of 27 gates are required at level 2 of a 4-bit adder.

Level 3 consists of 2 cells, which will be merged to obtain the final result. The carry-out from cell 1 is used to select $S_{2,2}^{0}$ or $S_{2,2}^{1}$. The same carry is used to select
$S^0_{2,3}$ or $S^1_{2,3}$ and $C^0_{2,4}$ or $C^1_{2,4}$. In RHS of Figure 10a, the following should be transformed:

\[
\begin{align*}
C^0 & \rightarrow C^0_{2,1}, \quad S^1_{1,0} \rightarrow S^1_{2,2}, \quad S^0_{1,0} \rightarrow S^0_{2,2}, \\
C^0_{1,0} & \rightarrow C^0_{1,3}, \quad C^1_{1,0} \rightarrow C^1_{1,3}
\end{align*}
\]

![Diagram](image)

**Figure 10. Number of Gates Required at Level 1 to Compute (a) SUM and (b) CARRY.**

To select the MSS replace $C^0$ by $C^0_{2,1}$, $S^1_{1,0}$ by $S^1_{2,3}$, $S^0_{1,0}$ by $S^0_{2,3}$. A maximum of 19 gates are required at level 3 for a 4-bit adder. Therefore, a 4-bit CSA, which converges in 3 levels, requires a total of 73 gates.

Using the above approach, the approximate number of gates required to construct 8, 16, 32 and 64-bit CSA's are 161, 341, 695 and 1407.

The convergence formula for an N-bit CSA is:
The number of gates required for a 4-bit CSCA is calculated as follows:

At every level in a CSCA, $S$, $S^*$ and $C$, $C^*$ are computed. At level 1, the number of gates required to compute $S_{i,i}$, $S^*_{i,i}$, and $C_{i,i}$, $C^*_{i,i}$, where $i = 0$ to 3, is 40 (using equations 16-19). The number of gates required to compute $S_{2,i}$, $S^*_{2,i}$ and $C_{2,i}$, $C^*_{2,i}$ is 64 (using equations 20-24). Therefore, 104 gates are required for 4-bit CSCA which converges in 2 levels.

Using the above approach, the approximate number of gates required for 8, 16, 32 and 64-bit CSCA adders are 336, 928, 2368, 5760.

The convergence formula for an $N$-bit CSCA is

$$C_1 \log_2 \log_2 N + C_2$$

where $C_1$ is the level switching time and $C_2$ is the completion detection time.

A comparison between the number of gates for HSCSA and CSCA is shown in Table 7.

In the next chapter HSCSA will be interfaced with MPM and MM. It will be proved that there is considerable time saving when compared to synchronous systems.
Table 7

Comparison of the Number of Gates Required by HSCSA and CSA

<table>
<thead>
<tr>
<th>NUMBER OF BITS</th>
<th>HSCSA</th>
<th>CSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>74</td>
<td>104</td>
</tr>
<tr>
<td>8</td>
<td>162</td>
<td>336</td>
</tr>
<tr>
<td>16</td>
<td>343</td>
<td>928</td>
</tr>
<tr>
<td>32</td>
<td>700</td>
<td>2368</td>
</tr>
<tr>
<td>64</td>
<td>1417</td>
<td>5760</td>
</tr>
</tbody>
</table>
CHAPTER VI

INTERFACING THE MPM, MM AND HSCSA

The designed HSCSA fits nicely in self timed systems. In a self timed system each functional element processes data using its own clock. All the functional elements communicate through handshaking lines. The advantage of a self timed system is the time saving when compared to synchronous systems. The important question is, whether the time saving offsets degradation in performance due to the handshaking lines. Also asynchronous systems are still limited by their slowest function (serial I/O, disk I/O). Due to the existence of several clocks, Electromagnetic Interference (EMI) is a major problem though certain EM noisy functional units in the circuit may be withdrawn from or modified in the circuit with little impact on the other sections. Extensive research has been done to overcome these problems at several leading institutions. It has been proved that an asynchronous system contributes a considerable time saving. Therefore, we will assume that the environment is set up to deal with asynchronous signals. In the next section, the MPM, MM and HSCSA will be simulated using UAHPL and results will indicate considerable time saving.
A general purpose Microprocessor Module (MPM) which is "operation bound" will be simulated using UAHPL. Randomly generated data will be used to test the MPM. An average frequency of addition operations, required by the microprocessor, will be computed. These addition operations are performed using the built-in synchronous adder. Furthermore, the MM (which has slower speed when compared to MPM and HSCSA), MPM (fastest speed), and HSCSA (slower than the speed of MPM) will be linked and tested. For the same operands, the time required by the built-in synchronous adder and the HSCSA will be compared.

Simulation Results

An AHPL module for 16-bit HSCSA has been written and simulated using randomly generated numbers. The same data have been used in testing 16-bit CSCA module. Listed in Table 8 is a comparison of convergence times. From this table, one finds that the average numbers of stages to be 3.19 and 3.55 for the HSCSA and CSCA respectively. It is, therefore, fairly obvious that HSCSA not only enjoys space advantage, as shown in Table 7, but also speed advantage. The AHPL sequence for both modules is given in Appendix D.

A complete AHPL sequence for the MPM is given in Appendix B. The AHPL sequence for HSCSA, is given in Appendix C. The MPM was tested for different programs using the built-in synchronous adder and HSCSA.
Table 8
Comparison of Simulation Results

<table>
<thead>
<tr>
<th>ADDERS</th>
<th>NUMBER OF LEVELS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>HSCSA</td>
<td>26%</td>
</tr>
<tr>
<td>CSCA</td>
<td>19%</td>
</tr>
</tbody>
</table>

The percentages of data transfer, data manipulation, and control instructions were found to be 37, 26, and 37 respectively. Out of the data manipulation, and control instructions, we found the percentage of addition instruction to be 45. It was also found that there was a considerable amount of time saving when the HSCSA was interfaced to the MPM. This is due to the fact that the convergence time for an HSCSA is $\log_2 \log_2 N + 1$, where $N$ is the number of bits. To demonstrate this fact, let us consider the following the example: For a 64-bit addition, the maximum number of logic levels required by a synchronous adder (RCA) is 64, while the number of logic levels required by an HSCSA is 4 (derived from the convergence formula). On the other hand, the number of gates required is approximately 1.4k, which is much less than 20M gates required by the 64-bit CLAAD.
CHAPTER VII

CONCLUSION

Several adders have been evaluated in this paper. The aim of the study was to come up with a fast adder suitable for use in self timed systems. First, a modified RCA, called ASAD, was evaluated. The superiority of the 4-bit was verified using probabilistic method. The AHPL CAD tools are used to design 8-bits ASAD and probability theory was also used for evaluation. Due to technology constraints, however, ASAD, as designed in Chapter II, cannot be physically realizable. This is due to the fact that ASAD requires above 50MHz clock frequency. Consequently, another alternative had to be sought.

A careful review of all the adders has resulted in the selection of CSA as the best candidate for high speed addition. A very efficient early completion detection logic has been designed for CSA. The resulting adder was given the name HSCSA. A comparison of the number of gates and logic levels among these adders is given in Table 9.

Normalized Space-Speed Product (SSP) values are listed in Table 11. The smallest of the SSP's in Table 10 was used as normalizing factor.
Table 9
Number of Gates and Logic Levels for Different Adders

<table>
<thead>
<tr>
<th>ADDER</th>
<th>4-BITS</th>
<th>8-BITS</th>
<th>16-BITS</th>
<th>32-BITS</th>
<th>64-BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCA</td>
<td>20/4</td>
<td>40/8</td>
<td>80/16</td>
<td>160/32</td>
<td>320/64</td>
</tr>
<tr>
<td>CLAAD</td>
<td>156/1</td>
<td>2976/1</td>
<td>635026/1</td>
<td>*/1</td>
<td>*/1</td>
</tr>
<tr>
<td>CSA</td>
<td>73/3</td>
<td>161/4</td>
<td>341/5</td>
<td>695/6</td>
<td>1407/7</td>
</tr>
<tr>
<td>CSCA</td>
<td>104/2</td>
<td>336/3</td>
<td>928/3</td>
<td>2368/4</td>
<td>5760/4</td>
</tr>
<tr>
<td>HSCSA</td>
<td>74/2</td>
<td>337/3</td>
<td>343/3</td>
<td>700/3</td>
<td>1417/3</td>
</tr>
</tbody>
</table>

Table 10
Space-Speed Product for Different Adders

<table>
<thead>
<tr>
<th>ADDER</th>
<th>4-BITS</th>
<th>8-BITS</th>
<th>16-BITS</th>
<th>32-BITS</th>
<th>64-BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCA</td>
<td>80</td>
<td>240</td>
<td>1280</td>
<td>5120</td>
<td>20480</td>
</tr>
<tr>
<td>CLAAD</td>
<td>156</td>
<td>2976</td>
<td>635026</td>
<td>*/</td>
<td>*/</td>
</tr>
<tr>
<td>CSA</td>
<td>219</td>
<td>644</td>
<td>1705</td>
<td>4170</td>
<td>9849</td>
</tr>
<tr>
<td>CSCA</td>
<td>208</td>
<td>1008</td>
<td>2784</td>
<td>9472</td>
<td>23040</td>
</tr>
<tr>
<td>HSCSA</td>
<td>148</td>
<td>1011</td>
<td>1029</td>
<td>2100</td>
<td>4251</td>
</tr>
</tbody>
</table>

Figure 11 relates SSP and the number of bits for the 5 adders. From this figure, one can say that HSCSA is superior to all adders, except for less than 12-bits, where it is the second best.
Table 11

Normalized Space-Speed Product

<table>
<thead>
<tr>
<th>ADDER</th>
<th>4-BITS</th>
<th>8-BITS</th>
<th>16-BITS</th>
<th>32-BITS</th>
<th>64-BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCA</td>
<td>1</td>
<td>3</td>
<td>16</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td>CLAAD</td>
<td>1.95</td>
<td>37.2</td>
<td>7937</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>CSA</td>
<td>2.73</td>
<td>8.05</td>
<td>21.31</td>
<td>52.12</td>
<td>123</td>
</tr>
<tr>
<td>CSCA</td>
<td>2.6</td>
<td>12.6</td>
<td>34.8</td>
<td>118.4</td>
<td>288</td>
</tr>
<tr>
<td>HSCSA</td>
<td>1.85</td>
<td>12.6</td>
<td>12.86</td>
<td>26.25</td>
<td>53</td>
</tr>
</tbody>
</table>

* Extremely large number

Figure 11. Plot of SSP vs Number of Bits.

Figure 11 shows the plot of number of bits versus SSP. From the graph, it is obvious that HSCSA is the most
favorable and enjoys a considerable advantage in both space and speed for more than 12-bits.

Future improvements of this work includes the design of a fast multiplier using HSCSA. Also due to the fact that synchronous systems are widely used and will remain for the years to come, HSCSA can be modified to operate in synchronous mode. Such a modification could be achieved by having the control unit examine the operands and determine the number of clock cycles required as discussed in Proposition 1. In this case, the adder should be equipped with a counter which would be set by the control unit.
Appendix A

An AHPL Sequence for an 8-Bit ASAD
AHPL MODULE : AHK

MEMORY : SUM[8]; C[8]; STEP[8]; CSUM[9].

EXINPUTS : A[8]; B[8].

OUTPUTS : C1SUM[9].

1. $STEP <= 8$:
   $SUM <= A \oplus B; C <= A \& B.$

2. $STEP <= 8$:
   $SUM[0:6] <= SUM[0:6] \oplus C[1:7];$
   $C[0] <= C[0] \oplus SUM[0] \& C[1];$

3. $STEP <= 8$:
   $SUM[0:5] <= SUM[0:5] \oplus C[1:6];$
   $C[0] <= C[0] \oplus SUM[0] \& C[1];$

4. $STEP <= 8$:
   $SUM[0:4] <= SUM[0:4] \oplus C[1:5];$
   $C[0] <= C[0] \oplus SUM[0] \& C[1];$

5. $STEP <= 8$:
   $SUM[0:3] <= SUM[0:3] \oplus C[1:4];$
   $C[0] <= C[0] \oplus SUM[0] \& C[1];$

6. $STEP <= 8$:
   $SUM[0:2] <= SUM[0:2] \oplus C[1:3];$
   $C[0] <= C[0] \oplus SUM[0] \& C[1];$

7. $STEP <= 8$:
   $SUM[0:1] <= SUM[0:1] \oplus C[1:2];$
\[ C[0] \leq C[0] + \text{SUM}[0] \land C[1] ; \]
\[ C[1] \leq \text{SUM}[1] \land C[2] ; \]
\[ \Rightarrow (\text{^C}[1] \land \text{^C}[2]) / (9) . \]

8 \text{STEP} \leq 8; \text{SUM}[0] \leq \text{SUM}[0] \land C[1] ;
\[ C[0] \leq C[0] + \text{SUM}[0] \land C[1] . \]

9 \text{STEP} \leq 8; \text{CSUM} \leq C[0], \text{SUM} ;
\[ \text{C1SUM} = \text{CSUM} ; \]
\[ \text{SUM} \leq \{0,0,0,0,0,0,0,0\} ; \]
\[ C \leq \{0,0,0,0,0,0,0,0\}; \Rightarrow (1) . \]

\text{ENDSEQUENCE}

\text{CONTROLRESET}(1) .

\text{END} .
Appendix B

An AHPL Sequence for an 8-Bit Microprocessor
AHPLMODULE : CDBMICRO.

MEMORY : MEM<256>[8]; MA[8]; PC[8]; AC[8]; MD[8]; IR[8];
         IX[8]; STEP[8]; CFF; ZFF; NFF; VFF.

BUSES : ABUS[8]; BBUS[8]; ADBUS[8]; DBUS[8]; OBUS[8];
         CIN.

CLUNITS : INC[8](PC); DEC[8](MD); ADD[9](ABUS;BBUS;CIN);
         DCD[256](ADBUS); BUSFN[8](MEM;DCD(ADBUS)).

1 MA <= PC; STEP <= 8$1.

2 ADBUS = MA; MD <= DBUS; DBUS = BUSFN(MEM;DCD(ADBUS));
   PC <= INC(PC); STEP <= 8$2.

3 IR <= MD; STEP <= 8$3.

4 STEP <= 8$4;
   => (/IR[0:2])/(16).

   STEP <= 8$5;
   => (^IR[3])/(7).

6 ADBUS = MA; MD <= DBUS; DBUS = BUSFN(MEM;DCD(ADBUS));
   STEP <= 8$6.

7 ABUS = (MD ! MA) * (^IR[3], ^IR[3]);
   BBUS = (IX ! 8$0) * (^(^IR[0] & IR[1] & ^IR[2]),
   (^IR[0] & ^IR[1] & ^IR[2])); "DON'T INDEX THE LDX
   INSTR."
   CIN = 1$0;
   OBUS = ADD[1:8](ABUS; BBUS; CIN); MA <= OBUS;
   STEP <= 8$7.

8 STEP <= 8$8;
=> (^
/IR[0:1]))/(10).

9  PC * (^AC[0] & (+/AC[1:7])) <= MA; STEP <= 8$9; "JPA"
=> (1).

10 STEP <= 8$10;

11 MD <= AC; STEP <= 8$11.

12 ADBUS = MA; DBUS = MD; MEM * DCD(ADBUS) <= DBUS;
STEP <= 8$12;
=> (1).

13 ADBUS = MA; DBUS = BUSFN(MEM; DCD(ADBUS)); MD <= DBUS;
STEP <= 8$13;
=> (^IR[1])/(15).

14 IX <= MD; STEP <= 8$14; "LDX"
=> (1).

15 ABUS = MD; BBUS = AC; CIN = CFF;
OBUS = (ADD[1:8](ABUS;BBUS;CIN) ! (ABUS & BBUS) !
"ADC,"AND"
(ABUS @ BBUS) ! ABUS) * "XOR, MVT"
((^+/IR[0:2])), (IR[0] & ^IR[1] & ^IR[2]),
IR[2]));
AC <= OBUS;
ZFF <= (^+/OBUS)); NFF <= OBUS[0]; STEP <= 8$15;
VFF * (^+/IR[0:2])) <= (ABUS[0] & BBUS[0] & ^ADD[1](ABUS;BBUS;CIN)) + (^ABUS[0] & ^BBUS[0] &
ADD[1](ABUS;BBUS;CIN));
16 STEP <= 8$16; "NOP"

=> (^IR[3])/(18). "2-CYCLE INSTRUCTION, 60 TO CYCLE 2"

CFF * IR[4] <= (AC[0] ! AC[7]) * (^IR[5], IR[5]);
STEP <= 8$17.

18 STEP <= 8$18; "NOP"

=> (^(+/IR[6:7]))/(1).

19 STEP <= 8$19;

=> (IR[6])/(21).

20 AC <= 8$0; STEP <= 8$20;

=> (1).

21 CFF <= IR[7]; STEP <= 8$21;

=> (1).

ENDSEQUENCE
CONTROLRESET(1).
END.
Appendix C

An AHPL Sequence for 4-Bits HSCSA
AHPLMODULE : HSCSCA.
MEMORY : S[4]; SS[4]; C[4]; CS[4]; STEP[4].
OUTPUTS : S2[5].

1 STEP<=4$1; S<=A&B; SS<=^((A&B)); C<=^A&B; CS<=A&B.

2 STEP<=4$2; S[0]<= (CS[1]&SS[0])+(^CS[1]&S[0])+S[0]&SS[0]);
    S[1]<= (CS[2]&SS[1])+(^CS[2]&S[1])+S[1]&SS[1]);
    S[3]<=S[3];
    SS[0]<= (C[1]&S[0])+^C[1]&SS[0])+(S[0]&SS[0]);
    SS[1]<= (C[2]&S[1])+^C[2]&SS[1])+(S[1]&SS[1]);
    C[0]<= C[0]+(C[1]&(^CS[0]));
    CS[0]<= CS[0]+(CS[1]&(^C[0]));

3 STEP<=4$3;
    S[0]<= (CS[2]&SS[0])+(^CS[2]&S[0])+S[0]&SS[0]);
    S[1]<= (CS[3]&SS[1])+(^CS[3]&S[1])+S[1]&SS[1]);
SS[0] <= (C[2] & S[0]) + (C[2] & SS[0]) + (S[0] & SS[0]);
C[0] <= C[0] + (C[2] & (CS[0]));
CS[0] <= CS[0] + (CS[2] & (^C[0]));
CS[2] <= CS[2];

4 STEP <= 4$4; S2 = CS[0], S.
5 STEP <= 4$5; S <= \0,0,0,0\; => (1).
END SEQUENCE
CONTROL RESET (1).
END.
Appendix D

An AHPL Sequence for 16-Bits HSCSA
**AHPLMODULE** : HSCSA.

**MEMORY** : S0[16]; S1[16]; C0[16]; C1[16]; STEP[5]; S3[17].

**EXINPUTS** : A[16]; B[16].

**OUTPUTS** : S2[17].

1. \( \text{STEP} \leq 5\$1; S0 \leq A \& B; S1 \leq (A \& (A \& B)); C0 \leq A \& B; C1 \leq A + B. \)

2. \( \text{STEP} \leq 5\$2; S0[0] \leq (S1[0] \& C0[1]) + (S0[0] \& (\neg C0[1])); S1[0] \leq (S1[0] \& C1[1]) + (S0[0] \& (\neg C1[1])); C0[0] \leq (C1[0] \& C0[1]) + (C0[0] \& (\neg C0[1])); C1[0] \leq (C1[0] \& C1[1]) + (C0[0] \& (\neg C1[1])); S0[2] \leq (S2[2] \& C0[3]) + (S0[2] \& (\neg C0[3])); S1[2] \leq (S2[2] \& C1[3]) + (S0[2] \& (\neg C1[3])); C0[2] \leq (C1[2] \& C0[3]) + (C0[2] \& (\neg C0[3])); C1[2] \leq (C1[2] \& C1[3]) + (C0[2] \& (\neg C1[3])); S0[4] \leq (S1[4] \& C0[5]) + (S0[4] \& (\neg C0[5])); S1[4] \leq (S1[4] \& C1[5]) + (S0[4] \& (\neg C1[5])); C0[4] \leq (C1[4] \& C0[5]) + (C0[4] \& (\neg C0[5])); C1[4] \leq (C1[4] \& C1[5]) + (C0[4] \& (\neg C1[5])); S0[6] \leq (S1[6] \& C0[7]) + (S0[6] \& (\neg C0[7])); S1[6] \leq (S1[6] \& C1[7]) + (S0[6] \& (\neg C1[7])); C0[6] \leq (C1[6] \& C0[7]) + (C0[6] \& (\neg C0[7])); C1[6] \leq (C1[6] \& C1[7]) + (C0[6] \& (\neg C1[7])); S0[8] \leq (S1[8] \& C0[9]) + (S0[8] \& (\neg C0[9])); S1[8] \leq (S1[8] \& C1[9]) + (S0[8] \& (\neg C1[9])); C0[8] \leq (C1[8] \& C0[9]) + (C0[8] \& (\neg C0[9])); C1[8] \leq (C1[8] \& C1[9]) + (C0[8] \& (\neg C1[9]));
\[ S_0[10] \leq (S_1[10] \& C_0[11]) + (S_0[10] \& (\neg C_0[11])); \]
\[ S_1[10] \leq (S_1[10] \& C_1[11]) + (S_0[10] \& (\neg C_1[11])); \]
\[ C_1[10] \leq (C_1[10] \& C_0[11]) + (C_0[10] \& (\neg C_0[11])); \]
\[ C_1[10] \leq (C_1[10] \& C_1[11]) + (C_0[10] \& (\neg C_1[11])); \]
\[ S_0[12] \leq (S_1[12] \& C_0[13]) + (S_0[12] \& (\neg C_0[13])); \]
\[ S_1[12] \leq (S_1[12] \& C_1[13]) + (S_0[12] \& (\neg C_1[13])); \]
\[ C_0[12] \leq (C_1[12] \& C_0[13]) + (C_0[12] \& (\neg C_0[13])); \]
\[ C_1[12] \leq (C_1[12] \& C_1[13]) + (C_0[12] \& (\neg C_1[13])); \]
\[ S_0[14] \leq (S_1[14] \& C_0[15]) + (S_0[14] \& (\neg C_0[15])); \]
\[ S_1[14] \leq (S_1[14] \& C_1[15]) + (S_0[14] \& (\neg C_1[15])); \]
\[ C_0[14] \leq (C_1[14] \& C_0[15]) + (C_0[14] \& (\neg C_1[15])); \]
\[ C_1[14] \leq (C_1[14] \& C_1[15]) + (C_0[14] \& (\neg C_1[15])); \]


3 \text{ STEP} \leq 5; S_0[0] \leq (S_0[0] \& S_0[0] \& S_0[1]) \& ((\neg C_0[2] \& C_1[2]),
\quad (\neg C_0[2] \& C_1[2]), (C_0[2] \& C_1[2]));
\[ S_0[1] \leq (S_0[1] \& S_0[1] \& S_1[1]) \& ((\neg C_0[2] \& C_1[2]),
\quad (\neg C_0[2] \& C_1[2]), (C_0[2] \& C_1[2]));
\[ S_1[0] \leq (S_0[0] \& S_1[0] \& S_1[0]) \& ((\neg C_0[2] \& C_1[2]),
\quad (\neg C_0[2] \& C_1[2]), (C_0[2] \& C_1[2]));
\quad (\neg C_0[2] \& C_1[2]), (C_0[2] \& C_1[2]));
\[ C_0[0] \leq (C_0[0] \& C_0[0] \& C_1[0]) \& ((\neg C_0[2] \& C_1[2]),
\quad (\neg C_0[2] \& C_1[2]), (C_0[2] \& C_1[2]));
\[ C_1[0] \leq (C_0[0] \& C_1[0] \& C_1[0]) \& ((\neg C_0[2] \& C_1[2]),
\quad (\neg C_0[2] \& C_1[2]), (C_0[2] \& C_1[2]));
\quad (\neg C_0[6] \& C_1[6]), (C_0[6] \& C_1[6])); \]
\((^C0[6]&C1[6]),(C0[6]&C1[6])\);
S0[5]<= (S0[5]!S0[5]!S1[5])*(^C0[6]&C1[6]),
\((^C0[6]&C1[6]),(C0[6]&C1[6])\);
S1[4]<= (S0[4]!S1[4]!S1[4])*(^C0[6]&C1[6]),
\((^C0[6]&C1[6]),(C0[6]&C1[6])\);
S1[5]<= (S0[5]!S1[5]!S1[5])*(^C0[6]&C1[6]),
\((^C0[6]&C1[6]),(C0[6]&C1[6])\);
C0[4]<= (C0[4]!C0[4]!C1[4])*(^C0[6]&C1[6]),
\((^C0[6]&C1[6]),(C0[6]&C1[6])\);
C1[4]<= (C0[4]!C1[4]!C1[4])*(^C0[6]&C1[6]),
\((^C0[6]&C1[6]),(C0[6]&C1[6])\);
S0[8]<= (S0[8]!S0[8]!S1[8])*(^C0[10]&C1[10]),
\((^C0[10]&C1[10]),(C0[10]&C1[10])\);
S0[9]<= (S0[9]!S0[9]!S1[9])*(^C0[10]&C1[10]),
\((^C0[10]&C1[10]),(C0[10]&C1[10])\);
S1[8]<= (S0[8]!S1[8]!S1[8])*(^C0[10]&C1[10]),
\((^C0[10]&C1[10]),(C0[10]&C1[10])\);
S1[9]<= (S0[9]!S1[9]!S1[9])*(^C0[10]&C1[10]),
\((^C0[10]&C1[10]),(C0[10]&C1[10])\);
C0[8]<= (C0[8]!C0[8]!C1[8])*(^C0[10]&C1[10]),
\((^C0[10]&C1[10]),(C0[10]&C1[10])\);
C1[8]<= (C0[8]!C1[8]!C1[8])*(^C0[10]&C1[10]),
\((^C0[10]&C1[10]),(C0[10]&C1[10])\);
S0[12]<= (S0[12]!S1[12])*(^C0[14],C0[14])
S0[13]<= (S0[13]!S1[13])*(^C0[14],C0[14])
C0[12]<= (C0[12]!C1[12])*(^C0[14],C0[14])
\( \Rightarrow (\{S0[2]+S0[6]+S0[10]\})/(9) \).

4 \( \text{STEP}<5 \); 

\[
S0[0] = (S0[0]!S0[0]!S1[0])*(^C0[4]&^C1[4]), 
(^C0[4]&C1[4]), (C0[4]&C1[4]));
\]

\[
S0[1] = (S0[1]!S0[1]!S1[1])*(^C0[4]&^C1[4]), 
(^C0[4]&C1[4]), (C0[4]&C1[4]));
\]

\[
S1[0] = (S0[0]!S1[0]!S1[0])*(^C0[4]&^C1[4]), 
(^C0[4]&C1[4]), (C0[4]&C1[4]));
\]

\[
S1[1] = (S0[1]!S1[1]!S1[1])*(^C0[4]&^C1[4]), 
(^C0[4]&C1[4]), (C0[4]&C1[4]));
\]

\[
S0[2] = (S0[2]!S0[2]!S1[2])*(^C0[4]&^C1[4]), 
(^C0[4]&C1[4]), (C0[4]&C1[4]));
\]

\[
S0[3] = (S0[3]!S0[3]!S1[3])*(^C0[4]&^C1[4]), 
(^C0[4]&C1[4]), (C0[4]&C1[4]));
\]

\[
S1[2] = (S0[2]!S1[2]!S1[2])*(^C0[4]&^C1[4]), 
(^C0[4]&C1[4]), (C0[4]&C1[4]));
\]

\[
S1[3] = (S0[3]!S1[3]!S1[3])*(^C0[4]&^C1[4]), 
(^C0[4]&C1[4]), (C0[4]&C1[4]));
\]

\[
C0[0] = (C0[0]!C0[0]!C1[0])*(^C0[4]&^C1[4]), 
(^C0[4]&C1[4]), (C0[4]&C1[4]));
\]

\[
C1[0] = (C0[0]!C1[0]!C1[0])*(^C0[4]&^C1[4]), 
(^C0[4]&C1[4]), (C0[4]&C1[4]));
\]

\[
S0[8] = (S0[8]!S1[8])*(^C0[12],C0[12]);
\]

\[
S0[9] = (S0[9]!S1[9])*(^C0[12],C0[12]);
\]

\[
S0[10] = (S0[10]!S1[10])*(^C0[12],C0[12]);
\]

\[
S0[11] = (S0[11]!S1[11])*(^C0[12],C0[12]);
\]

\[
C0[8] = (C0[8]!C1[8])*(^C0[12],C0[12]).
\]
5 \text{STEP}<=5; \quad \text{S0[0]}=(\text{S0[0]}!\text{S1[0]})*(\text{C0[8]},\text{C0[8]});
\text{S0[1]}=(\text{S0[1]}!\text{S1[1]})*(\text{C0[8]},\text{C0[8]});
\text{S0[2]}=(\text{S0[2]}!\text{S1[2]})*(\text{C0[8]},\text{C0[8]});
\text{S0[3]}=(\text{S0[3]}!\text{S1[3]})*(\text{C0[8]},\text{C0[8]});
\text{S0[4]}=(\text{S0[4]}!\text{S1[4]})*(\text{C0[8]},\text{C0[8]});
\text{S0[5]}=(\text{S0[5]}!\text{S1[5]})*(\text{C0[8]},\text{C0[8]});
\text{S0[6]}=(\text{S0[6]}!\text{S1[6]})*(\text{C0[8]},\text{C0[8]});
\text{S0[7]}=(\text{S0[7]}!\text{S1[7]})*(\text{C0[8]},\text{C0[8]});
\text{C0[0]}=(\text{C1[0]}*\text{C0[8]}).

6 \text{STEP}<=5; \quad \text{S2=0},\text{S0};\text{S0}<=0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;=>(1).

7 \text{STEP}<=5; \quad \text{S0[0]}=(\text{S1[0]}!\text{S0[0]})*(\text{C0[2]},\text{C0[2]});
\text{S0[1]}=(\text{S1[1]}!\text{S0[1]})*(\text{C0[2]},\text{C0[2]});
\text{S0[2]}=(\text{S1[2]}!\text{S0[2]})*(\text{C0[4]},\text{C0[4]});
\text{S0[3]}=(\text{S1[3]}!\text{S0[3]})*(\text{C0[4]},\text{C0[4]});
\text{S0[4]}=(\text{S1[4]}!\text{S0[4]})*(\text{C0[6]},\text{C0[6]});
\text{S0[5]}=(\text{S1[5]}!\text{S0[5]})*(\text{C0[6]},\text{C0[6]});
\text{S0[6]}=(\text{S1[6]}!\text{S0[6]})*(\text{C0[8]},\text{C0[8]});
\text{S0[7]}=(\text{S1[7]}!\text{S0[7]})*(\text{C0[8]},\text{C0[8]});
\text{S0[8]}=(\text{S1[8]}!\text{S0[8]})*(\text{C0[10]},\text{C0[10]});
\text{S0[9]}=(\text{S1[9]}!\text{S0[9]})*(\text{C0[10]},\text{C0[10]});
\text{S0[10]}=(\text{S1[10]}!\text{S0[10]})*(\text{C0[12]},\text{C0[12]});
\text{S0[11]}=(\text{S1[11]}!\text{S0[11]})*(\text{C0[12]},\text{C0[12]});
\text{S0[12]}=(\text{S1[12]}!\text{S0[12]})*(\text{C0[14]},\text{C0[14]});
\text{S0[13]}=(\text{S1[13]}!\text{S1[13]})*(\text{C0[14]},\text{C0[14]});=>(1).

8 \text{STEP}<=5; \quad \text{S0[0]}=(\text{S1[0]}!\text{S0[0]})*(\text{C0[4]},\text{C0[4]});

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
S0[1]<=(S1[1]!S0[1])*(C0[4],^C0[4]);
S0[2]<=(S1[2]!S0[2])*(C0[4],^C0[4]);
S0[3]<=(S1[3]!S0[3])*(C0[4],^C0[4]);
S0[5]<=(S1[5]!S0[5])*(C0[8],^C0[8]);
S0[6]<=(S1[6]!S0[6])*(C0[8],^C0[8]);
S0[7]<=(S1[7]!S0[7])*(C0[8],^C0[8]);
S0[9]<=(S1[9]!S0[9])*(C0[12],^C0[12]);
S0[10]<=(S1[10]!S0[10])*(C0[12],^C0[12]);
S0[11]<=(S1[11]!S0[11])*(C0[12],^C0[12]);=>(1).

ENDSEQUENCE

CONTROLRESET(1).

END.
REFERENCES


