Design and Analysis of Asynchronous Adders

Khalid Khanfar

Western Michigan University

Follow this and additional works at: http://scholarworks.wmich.edu/masters_theses

Part of the Computer Sciences Commons

Recommended Citation

http://scholarworks.wmich.edu/masters_theses/991

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.
DESIGN AND ANALYSIS OF
ASYNCHRONOUS ADDERS

by

Khalid Khanfar

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 Computer Science

Western Michigan University
Kalamazoo, Michigan
December 1991
DESIGN AND ANALYSIS OF ASYNCHRONOUS ADDERS

Khalid Khanfar, M.S.
Western Michigan University, 1991

The aim of this study is to come up with a fast adder suitable for use in self-timed subsystems. 7 adders were evaluated.

The findings from this study indicated that a Conditional Sum Adder (CSA) is the best candidate for high speed addition. One of the two resulting adders was given the name High Speed Conditional Sum Adder (HSCSA).
ACKNOWLEDGMENTS

I would like to express my deepest respect and appreciation to my advisor, Dr. Eltayeb Abuelyaman, for his guidance and assistance in the production of this thesis. Also I would like to express my gratitude to Dr. Dionysios Kountanis and Dr. Ben Pinkowski, who directed me through the completion of this thesis; to my family for their support; to my nieces Huda and Khairya; and to all my friends, in particular Ghazi Altarifi, Nader Musleh, Ahmed Musleh, Maher Musleh, Nazmi Quash, Nile Alrushaid, Fuad Ayyad and Saleh Ayyad.

Khalid Khanfar
DEDICATION

This thesis is dedicated to my God; to the memory of my father, who spent his whole life in educating my brothers and me; and to my mother, who sacrifices every thing in order to provide for our needs night and day so that we would have success in life.

Khalid Khanfar
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.
Design and analysis of asynchronous adders

Khanfar, Khalid Ahmad, M.S.

Western Michigan University, 1991
TABLE OF CONTENTS

ACKNOWLEDGMENTS............................................................... ii
LIST OF TABLES........................................................................ v
LIST OF FIGURES....................................................................... vi

CHAPTER

I. INTRODUCTION ................................................................. 1
II. STUDY OF WHAT HAS BEEN DONE ..................................... 3
   Addition Using CSA .................................................. 4
   Addition Using CSCA ............................................... 7
III. DESIGN OF AN ASAD ........................................................ 9
   Design of an 4-bit ASAD ........................................... 9
   Improved CSA ......................................................... 12
   Addition Using Improved CSA ..................................... 14
IV. PROBABILISTIC ANALYSIS OF A 4-BIT ASAD ..................... 15
   Probability of Every Level in a 4-bit ASAD .................... 15
V. A 4-ASAD ANALYSIS .......................................................... 21
   Logic Level Analysis ............................................... 21
   AHPL Steps Analysis ............................................... 22
VI. DESIGN OF A HSCSA .......................................................... 26
   HSCSA Design ......................................................... 26
   Complexity of CSA and CSCA ....................................... 27
VII. CONCLUSION ..................................................................... 32
Table of Contents--Continued

APPENDICES ............................................ 33
A. A Complete AHPL Sequence for a 4 CSA .. 33
B. A Complete AHPL Sequence for a 4-CSCA . 36
C. A Complete AHPL Sequence for a 4-ASAD . 40
D. A Complete AHPL Sequence for a 8-ASAD . 43
E. A Complete AHPL Sequence for a 16-CSA . 46
F. A Complete AHPL Sequence for a 16-HSCSA 54

REFERENCES ............................................ 63
LIST OF TABLES

1. I/O Relations of the Lower Block of ASAD . . . . 10
2. Selection According to the States of the Cells . 12
3. Logic Levels and Their Probabilities . . . . . 15
4. Comparison of 4-bit Adders Logic Levels . . . . 20
5. Logical Levels Required for Adding Two Numbers Between 0 and 15 Using 4-ASAD . . . . . . . . . 22
6. AHPL Steps Required for Adding Two Numbers Between 0 and 15 Using 4-ASAD . . . . . . . . . 24
7. Logical Levels and Their Completion Probability . 25
8. Number of Gates Required by HSCSA and CSCA . . . 31
LIST OF FIGURES

1. Addition Using CSA .................................... 6
2. Block Diagram of an ASAD Cell ................................. 9
3. Layout of 4-bit ASAD ........................................... 11
4. Addition Using Improved CSA ................................... 14
5. Number of Gates Required at Level 1 to Compute the SUM and the CARRY in HSCSA ................. 29
CHAPTER I

INTRODUCTION

Clocking is one of the most important factors in determining the speed of a microprocessor. In order to achieve minimal delay, it is important to speed up the execution cycle. One way to speed up the execution time is to have modules communicate in an asynchronous mode. One of the main modules of a microprocessor is the adder. Several synchronous adders have been developed using different algorithms [1-6]. Each enjoys advantages and suffers from disadvantages. Our objective here is to study and minimize these disadvantages. Two main factors will be considered:

1. The number of logic levels required or speed.
2. The number of gates or space.

In Chapter II we will discuss what has been published. Some definitions and the description of an ASynchronous ADder (ASAD) will be stated in Chapter III. The probability that addition completes at a level will be stated in Chapter IV. Analysis of the logic levels and the ASAD module using A Hardware Programming Language (AHPL) will be stated in Chapter V. The design of a High Speed Conditional Sum Adder (HSCSA) will be discussed in Chapter VI.
Finally, the conclusion will be stated in Chapter VII.
CHAPTER II

STUDY OF WHAT HAS BEEN DONE

The most commonly known adders are the Ripple Carry Adder (RCA) [1], the Carry Look Ahead Adder (CLADR) [2], the Ripple Carry Look Ahead Adder (RCLAD) [7], the Conditional Sum Adder (CSA) [3], and the Conditional Sum early Completion Adder (CSCA) [6]. In RCA, a lot of time is wasted assuming a carry will propagate even if there is no carry. In this case, addition passes through all the levels sequentially. The CLADR is the fastest, but it is expensive, for it uses too many gates. RCLADR is formed by combining advantages of RCA and CLADR [7]. However, neither the speed nor the space gained is near optimal. The CSA is reasonable in speed but not in the number of gates. A CSA adder has $N$ full adders which may be viewed as leafs of a binary tree whose root node represents the final sum. The conditional sum method of performing addition requires that, for each pair of bits to be added, two results are generated. The first result, one sum and one carry bit per stage, is prepared as though the carry-in to a stage was a 0. The second result is prepared as though the carry-in to that stage was 1 and consists of one sum and one carry bit.
per stage. Next, these results are examined in pairs. The addition illustrated in Figure 1 uses the CSA. The augend $A_i$ is being added to the addend $B_i$, where $i$ is the bit index to provide a sum in at level 3. For the first level, the following Boolean equations are used:

$$S^0_{1,i} = A_i\overline{B_i} + \overline{A_i}B_i$$

$$S^1_{1,i} = A_iB_i + \overline{A_i}\overline{B_i}$$

$$C^0_{1,i} = A_iB_i$$

$$C^1_{1,i} = A_i + B_i$$

The second level results are determined from pairs, from right to left, of the first level stages. The sum bits for the stage pairs can now be calculated as follows:

$$S^0_{2,k} = S^1_{1,k}C^0_{1,k-1} + S^0_{1,k}C^0_{1,k-1}$$

$$S^1_{2,k} = S^1_{1,k}C^1_{1,k-1} + S^0_{1,k}C^1_{1,k-1}$$

$$S^0_{2,k-1} = S^0_{1,k-1}$$

$$S^1_{2,k-1} = S^1_{1,k-1}$$

where $K = 2^{i-1}, 2 \times 2^{i-1}, 4 \times 2^{i-1}, \ldots, M$

where $M$ is the number of bits and $j$ is the level number. The carry for the stage pair is formulated as follows:

$$C^0_{2,k} = C^1_{1,k}C^0_{1,k-1} + C^0_{1,k}C^0_{1,k-1}$$
The sum bits for the stages at level 3 is calculated as follows:

\[ C_{2,k}^1 = C_{1,k}^1 C_{1,k-1}^1 + C_{1,k}^0 \bar{C}_{1,k-1}^1 \]  

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

The carry-in of the least significant cell is always assumed to be zero. So, for the first cell we choose the first line to be merged with its sum to form the first cell of the next level. Then we choose the second line of the most two significant cells for the next level; recursively, we find the total sum. As we note here, we have to go through a fixed number of levels in any addition operation regardless of the state of the cells. Hence, this technique will be improved in Chapter III. The complete AHPL sequence for 4-bit CSA is given in Appendix A.
Figure 1. Addition Using CSA.

CSCA is an improvement over CSA introduced by Mortin [6]. Four different values are generated in each level and are used to determine the values of the next level. These values are:

\[
\begin{array}{cc}
C & S \\
C^* & S^*
\end{array}
\]
The Boolean equations for generating $S$, $S^*$ and $C$, $C^*$ are given as follows:

For the first level, these values are generated by using the following equations.

- $S_{1,i} = A_iB_i + \overline{A_i}B_i$  
- $S^*_{1,i} = A_iB_i + \overline{A_i}B_i$  
- $C_{1,i} = \overline{A_i}B_i$  
- $C^*_{1,i} = A_iB_i$  
- $\overline{C_{1,i}} = A_i + B_i$  
- $\overline{C^*_{1,i}} = \overline{A_i} + \overline{B_i}$

For the second and the further levels, these values are generated by using the following equations.

- $S_{j+1,i} = C^*_{j,i-n_j}S_{j,i} + \overline{C^*_{j,i-n_j}}S_{j,i} + S_{j,i}S^*_{j,i}$  
- $S^*_{j+1,i} = C_{j,i-n_j}S^*_{j,i} + \overline{C_{j,i-n_j}}S^*_{j,i} + S_{j,i}S^*_{j,i}$  
- $C_{j+1,i} = C_{j,i} + C_{j,i-n_j}\overline{C^*_{j,i}}$  
- $C^*_{j+1,i} = C^*_{j,i} + C^*_{j,i-n_j}\overline{C_{j,i}}$  
- $\overline{C_{j+1,i}} = \overline{C_{j,i}}(\overline{C_{j,i-n_j}} + C^*_{j,i})$  
- $\overline{C^*_{j+1,i}} = \overline{C^*_{j,i}}(\overline{C^*_{j,i-n_j}} + C^*_{j,i})$  
  
  $n_j = 2^{j-1}, j=1,2,\ldots$
The variable \( n_j \) is the check bit offset for a particular level \((j)\) and is also required. The variable \( n_j \), increasing as two to the power of the processing level number, is the look-back index for the carry completion test for each bit. The complete AHPL sequence for 4-bit CSCA is given in Appendix B.

From the above review, one notices a delay problem in some adders and a gate count problem in others. To minimize these problems we introduce the ASynchronous ADder (ASAD).
CHAPTER III

DESIGN OF AN ASAD

Design of an 4-bit ASAD

ASAD is considered asynchronous because addition completes whenever there is no propagating carry. Communication between ASAD and the control unit is handled by handshaking.

The layout of cell i, a typical ASAD cell is shown in Figure 2. It consists of two blocks:

1. The upper, which represents a simple half-adder.
2. The lower, which represents the logic required to convert the cell into a full adder.

To distinguish between outputs from the upper and lower blocks, the following symbols will be defined:

$S_{hi}$: Sum output of half adder block(i)
S(ai): Sum output of ASAD(i)
C(hi+1): Carry-out from half adder block i
C(ai+1): Carry-out from the ASAD block i
C(ai): Carry-in from the ASAD cell i-1

The relationship between inputs and outputs of the lower block is given in Table 1.

Table 1
I/O Relations of the Lower Block of ASAD

<table>
<thead>
<tr>
<th>ROW#</th>
<th>C(hi+1)</th>
<th>S(hi)</th>
<th>C(Ai)</th>
<th>S(Ai)</th>
<th>C(Ai)</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>

Before we discuss the design of ASAD, let us define the following:
Definition 1: The number of logic level is the maximum number of gates through which the sum and/or the carry propagate.

Definition 2: The number of Clock Cycle is the maximum number of AHPL steps used in determining the sum.

Next, we redefine the well-known Carry Generate and Carry Propagate according to the layout of ASAD. In addition, we will define an Idle state for a half adder.

Definition 3: A cell \(i\) is in the Carry Propagate State (CPS); if and only if, \(S(hi)=1\) and \(C(hi+1)=0\).

Definition 4: A cell \(i\) is in the Carry Generate State (CGS); if and only if, \(C(hi+1)=1\).

Definition 5: A cell \(i\) is in the Idle State (IDS); if and only if, it is neither in CPS nor in CGS.

The layout of a 4-bit ASAD is given in Figure 3.

![Figure 3. Layout of 4-bit ASAD.](image-url)
Before we discuss the 4-bit ASAD in details, let us analyze the improved CSA.

Improved CSA

As we mentioned in Chapter II, CSA can be improved using some of the above definitions. In a 4-bit CSA, once we generate the first level, we can generate the final sum at the second level. To generate the second level, at the first level we scan the carry-ins and carry-outs of cells (1, 2, 3), but not cell 0, because we always select its $S^0$ under the assumption that the carry-in of cell 0 is always 0. We can then make selection according to Table 2.

Table 2
Selection According to the States of the Cells

<table>
<thead>
<tr>
<th>X</th>
<th>CGS/CPS</th>
<th>CGS/CPS</th>
<th>CGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>CGS</td>
<td>IDS</td>
<td>CGS</td>
</tr>
</tbody>
</table>

select second line of cells (1, 2, 3)

<table>
<thead>
<tr>
<th>X</th>
<th>IDS</th>
<th>CGS/CPS</th>
<th>CGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>IDS/CPS</td>
<td>IDS</td>
<td>CGS</td>
</tr>
<tr>
<td>select first line of cells 2, 3</td>
<td>select second line of cell 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>IDS/CPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IDS/CPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>IDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CGS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CPS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

select second line of cell 2 and 3 of cell 1

select first line of cells (1, 2, 3)

<table>
<thead>
<tr>
<th>select first line of cells (1, 2, 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

select second line of cells 2 and 3 of cell 1

select first line of cells (1, 2, and 3)

<table>
<thead>
<tr>
<th>select first line of cells (1, 2, and 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

select second line of cells 2 and 3 of cell 1

select first line of cells (1, 2, and 3)
To illustrate the difference between CSA and the improved CSA, let us consider the example in Figure 1 which is repeated in Figure 4 for convenience.

\[
\begin{array}{ccccccc}
Ai & 1 & 1 & 1 & 1 & \text{carry in} & \text{level} \\
Bi & 1 & 1 & 1 & 0 & \\
\end{array}
\]

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

Figure 4. Addition Using Improved CSA.

For Figure 4, the logical configuration of the cells is:

\[
\text{CGS} \quad \text{CGS} \quad \text{CGS} \quad \text{CP}
\]

As in Table 2, we choose the second line of the most significant half and the first line of the least significant half as the final sum, which is shown below:

\[
1 \ 1 \ 1 \ 0 \ 0 \ 1 \quad \text{(level 2)}.
\]

As we notice, only 2 logic levels and not 3 are needed.
CHAPTER IV

PROBABILISTIC ANALYSIS OF A 4-BIT ASAD

Probability of Every Level in a 4-bit ASAD

In a 4-bit ASAD, the number of logic levels needed to complete addition ranges from 2 to 8, each of which is separately analyzed in Table 3.

Table 3
Logic Levels and Their Probabilities

<table>
<thead>
<tr>
<th>Logic level 2;</th>
</tr>
</thead>
<tbody>
<tr>
<td>None of the cells produces a carry that affects the next most significant neighbor.</td>
</tr>
<tr>
<td>The logical configurations and their probabilities will then be:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>IDS/CPS</th>
<th>IDS/CPS</th>
<th>IDS/CPS</th>
<th>108/256</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell 3</td>
<td>cell 2</td>
<td>cell 1</td>
<td>cell 0</td>
<td>where each of the three least significant cells is in the IDS or CPS.</td>
</tr>
</tbody>
</table>

15
Table 3—Continued

CPS, while the state of the most significant cell is insignificant (i.e., don’t care state).

Logic level 3;

The generated carry will not propagate through the next most significant cell.

The logical configurations and their probabilities will then be:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Probability.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDS/CGS</td>
<td>IDS/CGS</td>
<td>IDS/CGS</td>
<td>CGS</td>
<td></td>
<td>8/25</td>
</tr>
<tr>
<td>IDS/CGS</td>
<td>IDS/CGS</td>
<td>CGS</td>
<td>IDS/CPS</td>
<td></td>
<td>12/25</td>
</tr>
<tr>
<td>IDS/CGS</td>
<td>CGS</td>
<td>IDS/CPS</td>
<td>IDS/CPS</td>
<td></td>
<td>18/256</td>
</tr>
<tr>
<td>CPS</td>
<td>IDS</td>
<td>CGS</td>
<td>CGS</td>
<td></td>
<td>2/25</td>
</tr>
<tr>
<td>X</td>
<td>CPS</td>
<td>IDS</td>
<td>CGS</td>
<td></td>
<td>8/256</td>
</tr>
<tr>
<td>CPS</td>
<td>IDS</td>
<td>IDS</td>
<td>CGS</td>
<td></td>
<td>2/256</td>
</tr>
<tr>
<td>CPS</td>
<td>IDS</td>
<td>CGS</td>
<td>IDS/CPS</td>
<td></td>
<td>6/256</td>
</tr>
<tr>
<td>cell 3</td>
<td>cell 2</td>
<td>cell 1</td>
<td>cell 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 3—Continued

The total probability is 56/256

Logic level 4;

Carry is propagated from the next most significant cell into the most significant cell.

The logical configurations and their probabilities will then be:

<table>
<thead>
<tr>
<th>CPS</th>
<th>CGS</th>
<th>X</th>
<th>IDS/CPS</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>24/256</td>
</tr>
<tr>
<td>CPS</td>
<td>CGS</td>
<td>IDS/CGS</td>
<td>CGS</td>
<td>4/256</td>
</tr>
</tbody>
</table>

Cell 3 cell 2 cell 1 cell 0

The total probability is 28/256

Logic level 5;

The generated carry is propagated through one and only one cell.

The logical configuration of the 4-ASAD is as follows

<table>
<thead>
<tr>
<th>X</th>
<th>IDS/CGS</th>
<th>CPS</th>
<th>CGS</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16/256</td>
</tr>
</tbody>
</table>
Table 3—Continued

<table>
<thead>
<tr>
<th>IDS/CGS</th>
<th>CPS</th>
<th>CGS</th>
<th>X</th>
<th>16/256</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell 3</td>
<td>cell 2</td>
<td>cell</td>
<td>cell 0</td>
<td></td>
</tr>
</tbody>
</table>

The total probability is \(\frac{32}{236}\)

Logic level 6:

The generated carry is propagated through the two most significant cells.
The logical configuration and its probability will then be:

<table>
<thead>
<tr>
<th>CPS</th>
<th>CPS</th>
<th>CGS</th>
<th>X</th>
<th>16/256</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell 3</td>
<td>cell 2</td>
<td>cell 1</td>
<td>cell 0</td>
<td></td>
</tr>
</tbody>
</table>

Logic level 7:

The generated carry from the least significant cell is propagated through the two next significant cells.
The logical configuration and its probability will then be:
Table 3—Continued

<table>
<thead>
<tr>
<th>IDS/CGS</th>
<th>CPS</th>
<th>CPS</th>
<th>CGS</th>
<th>8/256</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell 3</td>
<td>cell 2</td>
<td>cell 1</td>
<td>cell 0</td>
<td></td>
</tr>
</tbody>
</table>

Logic level 8;

The generated carry from the least significant cell is propagated through the next three most significant cells.

The logical configuration and its probability will then be:

<table>
<thead>
<tr>
<th>CPS</th>
<th>CPS</th>
<th>CPS</th>
<th>CGS</th>
<th>8/256</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell 3</td>
<td>cell 2</td>
<td>cell 1</td>
<td>cell 0</td>
<td></td>
</tr>
</tbody>
</table>

Probabilities for all the levels can be summarized as follows:

<table>
<thead>
<tr>
<th>Logic level</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>108/256</td>
</tr>
<tr>
<td>3</td>
<td>56/256</td>
</tr>
<tr>
<td>4</td>
<td>28/256</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
A comparison between the number of logic levels for 4-bit six addition algorithms is given in Table 4.

Table 4

<table>
<thead>
<tr>
<th>ADDER</th>
<th>NUMBER OF LOGIC LEVELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCA</td>
<td>8</td>
</tr>
<tr>
<td>CLAAD</td>
<td>2</td>
</tr>
<tr>
<td>RCLAD</td>
<td>4</td>
</tr>
<tr>
<td>CSA</td>
<td>6</td>
</tr>
<tr>
<td>CSCA</td>
<td>5*</td>
</tr>
<tr>
<td>ASAD</td>
<td>3.4*</td>
</tr>
</tbody>
</table>
CHAPTER V

A 4-ASAD ANALYSES

Logic Level Analysis

In the previous chapter, probability was used to analyze each level in the 4-ASAD. Using the AHPL language, we found that simulation results are consistent with the computed probabilities. From these results, one may study two factors: (1) logic levels, and (2) AHPL steps.

Logic Levels

The number of logic levels required for addition of any two numbers between 0 and 15 is given in Table 5. From Table 5 it is obvious that:

1. The number of logic levels ranges from 2 to 8;
2. The results are consistent with the probabilities given in the previous chapter; and
3. Eighty eight percent of the addition cases require a maximum of 5 logic levels.
Table 5  
Logical Levels Required for Adding Two Numbers Between 0 and 15 Using 4-ASAD  

<table>
<thead>
<tr>
<th></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>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

AHPL Steps Analysis  
A complete AHPL sequences for 4-ASAD and 8-ASAD are...
given in Appendices C and D. The maximum number of steps required to complete addition is using 4-ASAD 3. This is because there will be a maximum of three carries that may affect the next most significant cell(s). These are the only possibilities:

1. One Step: takes 2 logic levels
2. Two Steps: take 3 or 4 logic levels
3. Three Steps: take 5, 6, 7, or 8 logic levels

Simulation of an AHPL step may require a variable number of logic levels. This number is controlled by the state of the cells.

Table 6 shows the number of AHPL steps required for addition of numbers between 0 and 15. From Table 6 one notices the following:

1. The probability that addition completes in 1 step is 108/256.
2. The probability that addition completes in 2 steps is 82/256.
3. The probability addition completes in 3 steps is 66/256.
Table 6
AHPL Steps Required for Adding Two Numbers Between 0 and 15 Using 4-ASAD

<table>
<thead>
<tr>
<th></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>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Therefore, 42%, 32%, and 26% of addition operations will complete after 1, 2 and 3 steps respectively. From Table 3, one may determine the percentage for every level
as in Table 7.

Table 7
Logical Levels and Their Completion Probability

<table>
<thead>
<tr>
<th>Percentage of Addition Completion</th>
<th>Number of Logic Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>42%</td>
<td>2</td>
</tr>
<tr>
<td>21%</td>
<td>3</td>
</tr>
<tr>
<td>11%</td>
<td>4</td>
</tr>
<tr>
<td>14%</td>
<td>5</td>
</tr>
<tr>
<td>06%</td>
<td>6</td>
</tr>
<tr>
<td>03%</td>
<td>7</td>
</tr>
<tr>
<td>03%</td>
<td>8</td>
</tr>
</tbody>
</table>
CHAPTER VI

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 can not build the described ASAD. The clock period for an ASAD should be based on two logic levels delay. If one assumes an average propagation delay of 10ns per gate, then a two logic level function requires 20 ns. This means that a clock of 50MHz is required. Since 50MHz processors are not feasible, 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 the number gates of CSCA and has better convergence time when compared to CSCA. In this chapter, 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 CSA with additional logic for the detection of early completion. The following proposition
Proposition 1:

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

\[ S_{0,j,r} = 0 \text{ where } p \times r \text{ and } r = 1, 2, 3, \ldots, (2^{n-j+1} - 1). \]

Proof:

\( S_{0,j,r} \) 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 for up to 64 bits.

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 (LSC) is merged with cell 1, the Next Least Significant Cell (NLSC) and cell 3 the Most Significant Cell (MSC) is
merged with the Next Most Significant Cell (NMSC), which is cell 2. The selection between $S_{0,0}^0$ and $S_{1,0}^1$ is made depending on the carry-in. This requires a maximum of 4 gates to realize (see RHS of Figure 5 A). Similarly $C_{0,0}^0$ or $C_{1,0}^1$ is selected using the carry-in, which in turn requires 3 gates to realize (see RHS of Figure 5 B). The Next Least Significant (NLS) sum and carry are selected using $C_{0,0}^0$ or $C_{1,0}^1$. The required logic is shown in the RHS of Figure 5 (A) and Figure 5 (B) with the following transformation taking place:

$$C^0 \rightarrow C_{0,0}^0/C_{1,0}^1, \quad S_{1,0}^1 \rightarrow S_{1,1}^1, \quad S_{0,1,0}^0 \rightarrow S_{0,1,1}^0, \quad C_{0,0}^0 \rightarrow C_{0,1,1}^0, \quad C_{1,0}^1 \rightarrow C_{1,1}^1$$

Since the carry-in into cell 2 is unknown at this level, no selection can be made between $S_{0,2}^0$ and $S_{1,2}^1$. Hence, both are considered. The most significant sum (MSS) is selected using $C_{0,1,2}$ and/or $C_{1,1,2}$. In Figures 5 (A) and 5 (B), the following transformation should take place:

$$C^0 \rightarrow C_{0,1,2}^0, \quad C^1 \rightarrow C_{1,1,2}^1, \quad S_{1,0}^1 \rightarrow S_{1,3}^1, \quad S_{0,1,0}^0 \rightarrow S_{0,1,3}^0, \quad C_{0,1,0}^1 \rightarrow C_{0,1,3}^1, \quad C_{1,1,0}^1 \rightarrow C_{1,1,3}^1$$

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

Level 3 consists of two cells, which will be merged to obtain the final result. The carry-out from cell 1 is used to select $S_{0,2}^0$ or $S_{1,2}^1$. The same carry is used to select $S_{0,3}^0$ or $S_{1,3}^1$ and $C_{0,4}^0$ or $C_{1,4}^1$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 5. Number of Gates Required at Level 1 to Compute the SUM and the CARRY in HSCSA.

In RHS of Figure 5 (A) the following should be transformed:

\[ C^0 \rightarrow C^0_{2,1}, \quad S^1_{1,0} \rightarrow S^1_{2,2}, \quad S^0_{1,0} \rightarrow S^0_{2,2}, \quad C^0_{1,0} \rightarrow C^0_{1,3}, \quad C^1_{1,0} \rightarrow C^1_{1,3} \]
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 number of gates required to construct 8, 16, 32, and 64-bit CSA's are 161, 341, 695, and 1407.

The convergence formula of an \( N \)-bit CSA is:

\[
\log_2 N + 1
\]

The number of gates required for a 4-bit CSCA is calculated as follows:

At every level in a CSA, \( S, S', C, C' \) are computed. At level 1, the number of gates required to compute \( S_{1,i}, S'_{1,i}, C_{1,i}, C'_{1,i} \) where \( i = 0 \) to 3, is 40 (using 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 a 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, and 5760.

The convergence of 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 comple-
tion detection time.

A comparison between the number of gates for HSCSA and CSCA is shown Table 8.

Table 8

Number of Gates Required by HSCSA and CSCA

<table>
<thead>
<tr>
<th>NUMBER OF BITS</th>
<th>HSCSA</th>
<th>CSCA</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>3760</td>
</tr>
</tbody>
</table>

Complete AHPL sequence for 16-bit CSA and HSCSA are given in Appendices E and F respectively.
CHAPTER VII

CONCLUSION

Seven adders have been evaluated in this thesis. The aim of the study was to come up with a fast adder suitable for use in a self-timed subsystem. Using the probability theory and the AHPL CAD tools, a 4-bit ASAD was evaluated. The superiority of the 4-bit was verified. Due to technology constraints, however, ASAD, as designed in Chapter III, can not be physically realizable. This is due to the fact that ASAD requires above 50MHz clock frequency.

From the careful review of all adders, CSA has been selected as the best candidate for high speed addition. A very early completion detection logic has been designed for CSA. The resulting adder was given the name HSCSA.
Appendix A

A Complete AHPL Sequence for a 4-CSA
AHPLMODULE : MCA.

MEMORY : S0[4]; S1[4]; C0[4]; C1[4];
          STEP[5]; S3[5].


OUTPUTS : S2[5].

1 STEP<=5$1; S0<=A@B; S1<=(A@B); C0<=A&B; C1<=A+B.

2 STEP<=5$2; S0[0] <= (S1[0] & C0[1]) + (S0[0] & (C0[1]));
     S1[0] <= (S1[0] & C1[1]) + (S0[0] & (C1[1]));
     S0[2] <= (S1[2] & C0[3]) + (S0[2] & (C0[3]));
     C0[0] <= (C1[0] & C0[1]) + (C0[0] & (C0[1]));
     C1[0] <= (C1[0] & C1[1]) + (C0[0] & (C1[1]));
     C0[2] <= (C1[2] & C0[3]) + (C0[2] & (C0[3]));

3 STEP<=5$3; S0[0] <= (S1[0] & C0[2]) +
               (S0[0] & (C0[2]));
     S1[0] <= (S1[0] & C1[2]) + (S0[0] & (C1[2]));
     S0[1] <= (S1[1] & C0[2]) + (S0[1] & (C0[2]));
     S1[1] <= (S1[1] & C1[2]) + (S0[1] & (C1[2]));
     C0[0] <= (C1[0] & C0[2]) + (C0[0] & (C0[2]));
     C1[0] <= (C1[0] & C1[2]) + (C0[0] & (C1[2])).

4 STEP<=5$4; S3 <= ((C1[0], S1[0], S1[1], S0[2:3])!
       (C0[0], S0)) * (C0[2], (C0[2])).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
5 \text{STEP} <= 5; S2 = S3; S0 <= \{0, 0, 0\}; => (1).

\text{ENDSEQUENCE}

\text{CONTROLRESET}(1).

\text{END}.
Appendix B

A Complete AHPL Sequence for a 4-CSCA
AHPLMODULE : MCSCA.
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]));
    CS[3]<=CS[3];
3 \text{STEP} \leq 4$3$; S[0] \leq (CS[2]\&SS[0]) +$
\hspace{1cm} (^{\text{CS}[2]\&S[0]} + S[0]\&SS[0]);$
\hspace{1cm} S[1] \leq (CS[3]\&SS[1]) + (^{\text{CS}[3]\&S[1]} + S[1]\&SS[1]);$
\hspace{1cm} S[2] \leq (S[2]\&SS[2]) + S[2];$
\hspace{1cm} S[3] \leq S[3]\&SS[3];$
\hspace{1cm} SS[0] \leq (C[2]\&S[0]) + (^{\text{C}[2]\&SS[0]} + S[0]\&SS[0]);$
\hspace{1cm} SS[1] \leq (C[3]\&S[1]) + (^{\text{C}[3]\&SS[1]} + S[1]\&SS[1]);$
\hspace{1cm} SS[2] \leq (S[2]\&SS[2]) + S[2];$
\hspace{1cm} SS[3] \leq (S[3]\&SS[3]) + S[3];$
\hspace{1cm} C[0] \leq C[0] + (C[2]\&(^{\text{CS}[0]}));$
\hspace{1cm} C[1] \leq C[1] + (C[3]\&(^{\text{CS}[1]}));$
\hspace{1cm} C[2] \leq C[2] + (^{\text{CS}[2]});$
\hspace{1cm} C[3] \leq C[3] + (^{\text{CS}[3]});$
\hspace{1cm} CS[0] \leq CS[0] + (CS[2]\&(^{\text{C}[0]}));$
\hspace{1cm} CS[1] \leq CS[1] + (CS[3]\&(^{\text{C}[1]}));$
\hspace{1cm} CS[2] \leq CS[2];$
\hspace{1cm} CS[3] \leq CS[3].$

4 \text{STEP} \leq 4$4$; S2 = CS[0], S.

5 \text{STEP} \leq 4$5$; S = \{0, 0, 0, 0\}; \Rightarrow (1).
CONTROLRESET(1).

END.
Appendix C

A Complete AHPL Sequence for a 4-ASAD
AHPLMODULE : MADR.
MEMORY : SUM[4]; C[4]; STEP[5]; CSUM[5].
OUTPUTS : C1SUM[5].

1 STEP<=5$1; SUM[0:3]<=A[0:3]@B[0:3];
  C<=A&B; CSUM<=\0,0,0,0,0\.

2 STEP<=5$2; SUM[0:2]<=SUM[0:2]@C[1:3];
  C[0]<=C[0]+SUM[0]&C[1];
  C[1:2]<=SUM[1:2]&C[2:3];
  =>(C[1]&^C[2]&^C[3])/5).

3 STEP<=5$3; SUM[0:1]<=SUM[0:1]@C[1:2];
  C[0]<=C[0]+SUM[0]&C[1];
  C[1]<=SUM[1]&C[2];
  =>(C[1]&C[2])/5).

4 STEP<=5$4; SUM[0]<=SUM[0]@C[1];
  C[0]<=C[0]+SUM[0]&C[1].

5 STEP<=5$5; CSUM<=C[0], SUM; C1SUM=CSUM; =>(1).
ENDSEQUENCE

CONTROLRESET (1).

END.
Appendix D

A Complete AHPL Sequence for a 8-ASAD
APPENDIX D

This is a complete AHPL sequence of 8-ASAD.

AHPLMODULE :AHK.
MEMORY :SUM[8];C[8];STEP[8];CSUM[9].
EXINPUTS :A[8];B[8].
OUTPUTS :C1SUM[9].

1 STEP<=8$1;SUM<=A&B;C<=A&B.
2 STEP<=8$2;SUM[0:6]<=SUM[0:6]@C[1:7];
    C[0]<=C[0]+SUM[0]&C[1];
3 STEP<=8$3;SUM[0:5]<=SUM[0:5]@C[1:6];
    C[0]<=C[0]+SUM[0]&C[1];C[1:5]<=SUM[1:5]@C[2:6];
4 STEP<=8$4;SUM[0:4]<=SUM[0:4]@C[1:5];
    C[0]<=C[0]+SUM[0]&C[1];

5 STEP<=8$5;SUM[0:3]<=SUM[0:3]@C[1:4];
    C[0]<=C[0]+SUM[0]&C[1];C[1:3]<=SUM[1:3]@C[2:4];

6 STEP<=8$6;SUM[0:2]<=SUM[0:2]@C[1:3];
\[ C_0 = C_0 + \text{SUM}_0 \cdot C_1; C_{1:2} = \text{SUM}_{1:2} \cdot C_{2:3}; \]
\[ \Rightarrow (\cdot C_1 \cdot C_2 \cdot C_3) / 9. \]

7 STEP<=8$7; \text{SUM}[0:1] <= \text{SUM}[0:1] \cdot C[1:2];
\[ C_0 = C_0 + \text{SUM}_0 \cdot C_1; C_1 = \text{SUM}_1 \cdot C_2; \]
\[ \Rightarrow (\cdot C_1 \cdot C_2) / 9. \]

8 STEP<=8$8; \text{SUM}[0] <= \text{SUM}[0] \cdot C[1];
\[ C_0 = C_0 + \text{SUM}_0 \cdot C_1. \]

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

ENDSEQUENCE

CONTROLRESET(1).

END.
Appendix E

A Complete AHPL Sequence for a 16-CSA
AHPLMODULE : MCSA.

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\text{S}1; S0 \leq \text{A} \oplus \text{B};
   \text{S}1 \leftarrow (\text{^A} \oplus \text{B}); \text{C}0 \leq \text{A} \& \text{B}; \text{C}1 \leq \text{A} \oplus \text{B}.

2  \text{STEP} \leq 5\text{S}2; S0[0] \leq (S0[0], S0[0], S1[0]) *
   ((\text{^C}0[2] \& \text{^C}1[2]), (\text{^C}0[2] \& \text{C}1[2]),
    (\text{C}0[2] \& \text{C}1[2]));

   S0[1] \leq (S0[1], S0[1], S1[1]) *
   ((\text{^C}0[2] \& \text{^C}1[2]), (\text{^C}0[2] \& \text{C}1[2]),
    (\text{C}0[2] \& \text{C}1[2]));

   S1[0] \leq (S0[0], S1[0], S1[0]) *
   ((\text{^C}0[2] \& \text{^C}1[2]), (\text{^C}0[2] \& \text{C}1[2]),
    (\text{C}0[2] \& \text{C}1[2]));

   S1[1] \leq (S0[1], S1[1], S1[1]) *
   ((\text{^C}0[2] \& \text{^C}1[2]), (\text{^C}0[2] \& \text{C}1[2]),
    (\text{C}0[2] \& \text{C}1[2]));

   C0[0] \leq (C0[0], C0[0], CL[0]) *
   ((\text{^C}0[2] \& \text{^C}1[2]), (\text{^C}0[2] \& \text{C}1[2]),
    (\text{C}0[2] \& \text{C}1[2]));

   C1[0] \leq (C0[0], C1[0], CL[0]) *
   ((\text{^C}0[2] \& \text{^C}1[2]), (\text{^C}0[2] \& \text{C}1[2]),
    (\text{C}0[2] \& \text{C}1[2]));
(CO[2] & C1[2]));

S0[4] <= (S0[4], S0[4], S1[4]) *
    ( ( ^C0[6] & ^C1[6]), ( ^C0[2] & 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], S0[12], S1[12])*
    ((^C0[14] & ^C1[14]), (^C0[14] & C1[14]),
     (C0[14] & C1[14]));
S0[13] <= (S0[13], S0[13], S1[13])*
    ((^C0[14] & ^C1[14]), (^C0[14] & C1[14]),
     (C0[14] & C1[14]));
S1[12] <= (S0[12], S1[12], S1[12])*
    ((^C0[14] & ^C1[14]), (^C0[14] & C1[14]),
     (C0[14] & C1[14]));
S1[13] <= (S0[13], S1[13], S1[13])*
    ((^C0[14] & ^C1[14]), (^C0[14] & C1[14]),
     (C0[14] & C1[14]));
C0[12] <= (C0[12], C0[12], C1[12])*
(\(^{\land}C0[14] \&^{\land}C1[14]\), \(^{\land}C0[14] \&C1[14]\),
\(^{\land}C0[14] \&C1[14]\)).

3 \text{STEP} \leq 5$3; S0[0] \leftarrow (S0[0], S0[0], S1[0]) *
\(^{\land}C0[4] \&C1[4]\));
S0[1] \leftarrow (S0[1], S0[1], S1[1]) *
\(^{\land}C0[4] \&C1[4]\));
S1[0] \leftarrow (S0[0], S1[0], S1[0]) *
\(^{\land}C0[4] \&C1[4]\));
S1[1] \leftarrow (S0[1], S1[1], S1[1]) *
\(^{\land}C0[4] \&C1[4]\));
S0[2] \leftarrow (S0[2], S0[2], S1[2]) *
\(^{\land}C0[4] \&C1[4]\));
S0[3] \leftarrow (S0[3], S0[3], S1[3]) *
\(^{\land}C0[4] \&C1[4]\));
S1[2] \leftarrow (S0[2], S1[2], S1[2]) *
\(^{\land}C0[4] \&C1[4]\));
S1[3] \leftarrow (S0[3], S1[3], S1[3]) *
\[
\begin{align*}
(C0[4] & C1[4]));
\end{align*}
\]

\[
C0[0] <= (C0[0], C0[0], C1[0]) *
\]

\[
(C0[4] & C1[4]));
\]

\[
C1[0] <= (C0[0], C1[0], C1[0]) *
\]

\[
(C0[4] & C1[4]));
\]

\[
S0[8] <= (S0[8], S0[8], S1[8]) *
\]

\[
((^C0[12] & ^C1[12]), (^C0[12] & ^C1[12]), \\
(C0[12] & C1[12]));
\]

\[
S0[9] <= (S0[9], S0[9], S1[9]) *
\]

\[
((^C0[12] & ^C1[12]), (^C0[12] & ^C1[12]), \\
(C0[12] & C1[12]));
\]

\[
S1[8] <= (S0[8], S1[8], S1[8]) *
\]

\[
((^C0[12] & ^C1[12]), (^C0[12] & ^C1[12]), \\
(C0[12] & C1[12]));
\]

\[
S1[9] <= (S0[9], S1[9], S1[9]) *
\]

\[
((^C0[12] & ^C1[12]), (^C0[12] & ^C1[12]), \\
(C0[12] & C1[12]));
\]

\[
S0[10] <= (S0[10], S0[10], S1[10]) *
\]

\[
((^C0[12] & ^C1[12]), (^C0[12] & ^C1[12]), \\
(C0[12] & C1[12]));
\]

\[
S0[11] <= (S0[11], S0[11], S1[11]) *
\]

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(((C0[12]&^C1[12]), (^C0[6]&C1[12]),
(C0[12]&C1[12]));
S1[10]<= (S0[10], S1[8], S1[10]) *
((^C0[12]&^C1[12]), (^C0[12]&C1[12]),
(C0[12]&C1[12]));
S1[11]<= (S0[11], S1[11], S1[11]) *
((^C0[12]&^C1[12]), (^C0[12]&C1[12]),
(C0[12]&C1[12]));
C0[8]<= (C0[8], C0[8], C1[8]) *
((^C0[12]&^C1[12]), (^C0[12]&C1[12]),
(C0[12]&C1[12]));

4 STEP<=5$4; S0[0]<= (S1[0]&C0[4]) +
(S0[0]&(^C0[4]));
S1[0]<= (S1[0]&C1[4]) + (S0[0]&(^C1[4]));
S0[8]<= (S1[8]&C0[12]) + (S0[8]&(^C0[12]));
S1[8]<= (S1[8]&C1[12]) + (S0[8]&(^C1[12]));
C0[0]<= (C1[0]&C0[4]) + (C0[0]&(^C0[4]));
C1[0]<= (C1[0]&C1[4]) + (C0[0]&(^C1[4]));
C0[8]<= (C1[8]&C0[12]) + (C0[8]&(^C0[12]));
C1[8]<= (C1[8]&C1[12]) + (C0[8]&(^C1[12]));

5 STEP<=5$5; S0[0]<= (S1[0]&C0[8]) +
(S0[0]&(^C0[8]));
S1[0]<= (S1[0]&C1[8]) + (S0[0]&(^C1[8]));
C0[0]<= (C1[0]&C0[8]) + (C0[0]&(^C0[8]));
C1[0]<= (C1[0]&C1[8]) + (C0[0]&(^C1[8]));
6 \text{STEP} \leq 5 \text{S}6; \text{S}3 \leq (((\text{C}1[0], \text{S}1[0:7], \text{S}0[8:15])
nodex\(\text{C}0[0], \text{S}0\)) \times (\text{C}0[8], \text{C}0[8])).
\text{STEP} \leq 5 \text{S}7; \text{S}2 \leq \text{S}3;
\text{S}0 \leq \{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\}; \Rightarrow (1).
\text{ENDSEQUENCE}
\text{CONTROLRESET}(1).
\text{END}.
Appendix F

A Complete AHPL Sequence for a 16-HSCSA
AHPLMODULE : MCSA.

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

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

OUTPUTS : S2[17].

1  STEP<=5$1; S0<=A&B; S1<=(^A&B);
    C0<=A&B; C1<=A+B.

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

(7).

3 \text{STEP} \leq 5; \ S_0[0] = (^{S_0[0]} ! S_0[0] ! S_1[0]) * \]
\[
( ^{C_0[2]} & ^{C_1[2]} , ^{C_0[2]} & ^{C_1[2]} , \]
\[
( ^{C_0[2]} & ^{C_1[2]} )) ;
\]
\[ S_0[1] = (^{S_0[1]} ! S_0[1] ! S_1[1]) * \]
\[
( ^{C_0[2]} & ^{C_1[2]} , ^{C_0[2]} & ^{C_1[2]} , \]
\[
( ^{C_0[2]} & ^{C_1[2]} )) ;
\]
S1[0] <= (S0[0] ![S1[0] ! S1[0]) * 
\(C0[2] & C1[2]));

S1[1] <= (S0[1] ![S1[1] ! S1[1]) * 
\(C0[2] & C1[2]));

C0[0] <= (C0[0] ![C0[0] ! C1[0]) * 
\(C0[2] & C1[2]));

C1[0] <= (C0[0] ![C1[0] ! C1[0]) * 
\(C0[2] & C1[2]));

\(C0[6] & C1[6]));

\(C0[6] & C1[6]));

\(C0[6] & C1[6]));

\(C0[6] & C1[6]));
C0[4] <= (C0[4]!C0[4]!C1[4])* 
(C0[6] & C1[6]));

C1[4] <= (C0[4]!C1[4]!C1[4])* 
(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]));
59

\[ S_0[13] \leq (S_0[13] \lor S_1[13]) \times (C_0[14], C_0[14]); \]
\[ C_0[12] \leq (C_0[12] \lor C_1[12]) \times (C_0[14], C_0[14]); \]
\[ \Rightarrow (S_0[2] + S_0[6] + S_0[10]) / (8). \]

4 \quad \text{STEP} \leq 5; S_0[0] \leq (S_0[0] \lor S_0[0] \lor S_1[0]) \times

\[ ((C_0[4] \land C_1[4]), (C_0[4] \land C_1[4]), \]
\[ (C_0[4] \land C_1[4])); \]
\[ S_0[1] \leq (S_0[1] \lor S_0[1] \lor S_1[1]) \times

\[ ((C_0[4] \land C_1[4]), (C_0[4] \land C_1[4]), \]
\[ (C_0[4] \land C_1[4])); \]
\[ S_1[0] \leq (S_0[0] \lor S_1[0] \lor S_1[0]) \times

\[ ((C_0[4] \land C_1[4]), (C_0[4] \land C_1[4]), \]
\[ (C_0[4] \land C_1[4])); \]
\[ S_1[1] \leq (S_0[1] \lor S_1[1] \lor S_1[1]) \times

\[ ((C_0[4] \land C_1[4]), (C_0[4] \land C_1[4]), \]
\[ (C_0[4] \land C_1[4])); \]
\[ S_0[2] \leq (S_0[2] \lor S_0[2] \lor S_1[2]) \times

\[ ((C_0[4] \land C_1[4]), (C_0[4] \land C_1[4]), \]
\[ (C_0[4] \land C_1[4])); \]
\[ S_0[3] \leq (S_0[3] \lor S_0[3] \lor S_1[3]) \times

\[ ((C_0[4] \land C_1[4]), (C_0[4] \land C_1[4]), \]
\[ (C_0[4] \land C_1[4])); \]
\[ S_1[2] \leq (S_0[2] \lor S_1[2] \lor S_1[2]) \times

\[ ((C_0[4] \land C_1[4]), (C_0[4] \land C_1[4]), \]
\[ (C_0[4] \land C_1[4])); \]
\begin{align*}
S1[3] & \leq \left( S0[3] \land S1[3] \land S1[3] \right) * \\
& \quad \left( \left( \neg C0[4] \land C1[4] \right), \left( C0[4] \land C1[4] \right), \right. \\
& \left. \left( C0[4] \land C1[4] \right) \right) ; \\
C0[0] & \leq \left( C0[0] \land \neg C0[0] \land C1[0] \right) * \\
& \quad \left( \left( \neg C0[4] \land C1[4] \right), \left( C0[4] \land C1[4] \right), \right. \\
& \left. \left( C0[4] \land C1[4] \right) \right) ; \\
C1[0] & \leq \left( C0[0] \land \neg C0[0] \land C1[0] \right) * \\
& \quad \left( \left( \neg C0[4] \land C1[4] \right), \left( C0[4] \land C1[4] \right), \right. \\
& \left. \left( C0[4] \land C1[4] \right) \right) ; \\
S0[8] & \leq \left( S0[8] \land S1[8] \right) * \left( \neg C0[12] \land C0[12] \right) ; \\
S0[9] & \leq \left( S0[9] \land S1[9] \right) * \left( \neg C0[12] \land C0[12] \right) ; \\
S0[10] & \leq \left( S0[10] \land S1[10] \right) * \left( \neg C0[12] \land C0[12] \right) ; \\
S0[11] & \leq \left( S0[11] \land S1[11] \right) * \left( \neg C0[12] \land C0[12] \right) ; \\
C0[8] & \leq \left( C0[8] \land \neg C1[8] \right) * \left( \neg C0[12] \land C0[12] \right) . \\
\end{align*}

5 \text{ STEP} \leq 5S5; \quad S0[0] \leq \left( S0[0] \land S1[0] \right) * \left( \neg C0[8] \land C0[8] \right) ; \\
\quad S0[1] \leq \left( S0[1] \land S1[1] \right) * \left( \neg C0[8] \land C0[8] \right) ; \\
\quad S0[2] \leq \left( S0[2] \land S1[2] \right) * \left( \neg C0[8] \land C0[8] \right) ; \\
\quad S0[3] \leq \left( S0[3] \land S1[3] \right) * \left( \neg C0[8] \land C0[8] \right) ; \\
\quad S0[4] \leq \left( S0[4] \land S1[4] \right) * \left( \neg C0[8] \land C0[8] \right) ; \\
\quad S0[5] \leq \left( S0[5] \land S1[5] \right) * \left( \neg C0[8] \land C0[8] \right) ; \\
\quad S0[6] \leq \left( S0[6] \land S1[6] \right) * \left( \neg C0[8] \land C0[8] \right) ; \\
\quad S0[7] \leq \left( S0[7] \land S1[7] \right) * \left( \neg C0[8] \land C0[8] \right) ; \\
\quad C0[0] \leq \left( \neg C1[0] \land C0[8] \right) .
6  \text{STEP}<=5$6; S2=C0[0], S0;
\text{S0}<=\{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\};=>(1).

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

8  \text{STEP}<=5$9; S0[0]!=(S1[0]\mid S0[0])\times (C0[4],\neg C0[4]);
\quad S0[1]!=(S1[1]\mid S0[1])\times (C0[4],\neg C0[4]);
\quad S0[2]!=(S1[2]\mid S0[2])\times (C0[4],\neg C0[4]);
\quad S0[3]!=(S1[3]\mid S0[3])\times (C0[4],\neg C0[4]);
\quad S0[5]!=(S1[5]\mid S0[5])\times (C0[8],\neg C0[8]);
\quad S0[6]!=(S1[6]\mid S0[6])\times (C0[8],\neg C0[8]);
\quad S0[7]!=(S1[7]\mid S0[7])\times (C0[8],\neg C0[8]);
SO[9] <= (S1[9] ! S0[9]) * (C0[12], ^C0[12]);
SO[10] <= (S1[10] ! S0[10]) * (C0[12], ^C0[12]);

ENDSEQUENCE
CONTROLRESET(1).
END.
REFERENCES


