Counters and Multipliers with Threshold Logic

Tijs Huisman

Laboratory of Computer Architecture
and Digital Techniques (CARDIT)
Faculty of Electrical Engineering
Delft University of Technology
P.O. Box 5031, NL-2600 GA Delft,
The Netherlands

May 22, 1995

Delft University of Technology
Faculty of Electrical Engineering

Type	:	Master's Thesis
Number of pages	:	83
Date	:	May 22, 1995

Lab./Dept.	:	Laboratory of Computer Architecture and Digital Techniques (CARDIT)
Codenumber	:	1-68340-28(1995)08
Author	:	Tijs Huisman
Title	:	**Counters and Multipliers with Threshold Logic**

Supervisor	:	Prof. Dr. Ir. A. J. van de Goor
Mentor	:	Dr. Stamatis Vassiliadis

Abstract

We investigate conventional counters and their employment to produce multi-operand addition using threshold logic. The primary concern of the study is the area reduction of the implementations. We use direct reduction schemes that minimize the number of threshold elements and hierarchical reduction to diminish threshold gate requirements at the expenses of area.

Keywords

Digital Multiplier, Threshold Gate, Counters, Kautz Counters, Computer Arithmetic

Acknowledgements

The research project presented in this M.Sc thesis has been conducted at the laboratory of Computer Architecture and Digital Techniques at the Faculty of Electrical Engineering of the Delft University of Technology.

First of all, I wish to express my gratitude to my mentor, $\Sigma\tau\alpha\mu\acute{\alpha}\tau\eta$ $B\alpha\sigma\iota\lambda\varepsilon\iota\acute{\alpha}\delta\eta$ (Stamatis Vassiliadis), for his ideas and guidance throughout this period. I had the privilege of being his first master's student in the Netherlands. I thank him for the effort he put in the thesis. Most of all I thank him for his patience shown in the less enchanting periods telling me over and over again what a master's thesis is all about, and putting me back on the right track. I have benefited enormously, and learned much in the past ten months from working with $\Sigma\tau\alpha\mu\acute{\alpha}\tau\eta$.

Furthermore I would like to thank Sorin Cotofana for his critical proof reading of the manuscript, and his valuable comments on the thesis.

Gerard Speksnijder has been driving me around and accepting all requests in the final phase of the graduation with my defense, for which I am very grateful.

I thank my parents who made it possible for me to become an Engineer. The way to the graduation possessed a lot of road blocks, which had to be dealt with by using a lot of parental advice.

To Simone I am very much indebted for accepting and dealing with my numerous changes of moods, and her support during this period. I also thank her for working with me side-by-side during the nights short before the defense, including drawing the pictures needed for the presentation.

Contents

LIST OF FIGURES **vii**

LIST OF TABLES **ix**

1 Introduction **1**
- 1.1 Boolean Versus Threshold Logic and the Research Questions 6
- 1.2 Framework 6

2 Parallel Counters **9**
- 2.1 Graphical Representation 9
- 2.2 Input Considerations 12
- 2.3 Output Functions 12
- 2.4 Block Addition 15
- 2.5 Conclusions 16

3 Design of Parallel Counters **17**
- 3.1 The Basic Method 18
  - 3.1.1 Design of a Basic $(3 \mid 2)$ Counter 18
  - 3.1.2 Basic $(7 \mid 3)$ Counter 19
  - 3.1.3 A $(15 \mid 4)$ Counter 21

3.1.4 The General Case of the Basic Counters . . . 25
3.1.5 Cost and Delay of the Basic Method Counters . . . 26
3.2 The Minnick Method . . . 28
3.2.1 Design of a Minnick $(7 \mid 3)$ Counter . . . 28
3.2.2 A $(3 \mid 2)$ Counter . . . 30
3.2.3 A $(15 \mid 4)$ Counter . . . 30
3.2.4 The General Case of the Minnick Counters . . . 30
3.2.5 Cost and Delay of Minnick Counters . . . 32
3.3 The Kautz Method . . . 34
3.3.1 A Kautz $(3 \mid 2)$ Counter . . . 36
3.3.2 A Kautz $(15 \mid 4)$ Counter . . . 36
3.3.3 The general Case of the Kautz' Counters . . . 36
3.3.4 Cost and delay of Kautz Counters . . . 38
3.4 Cutting Unnecessary Lines . . . 39
3.5 Comparison . . . 39
3.6 Conclusions . . . 39

4 Multiple Addition **45**
4.1 Algorithm . . . 46
4.2 Cost Calculation . . . 48
4.3 Conclusions . . . 55

5 Multipliers **57**
5.1 Multiplication Algorithm . . . 57
5.2 Unsigned versus Signed Multiplications . . . 59
5.3 The algorithms . . . 59
5.4 Parallel Multipliers . . . 61
5.4.1 Cost Calculation . . . 62
5.5 Serial-Parallel Multipliers . . . 66
5.5.1 Algorithm . . . 67
5.5.2 Cost Calculation . . . 68
5.6 Conclusions . . . 71

6 Practical Considerations **73**
6.1 Hierarchical Addition . . . 74
6.2 Hierarchical Block Save Addition . . . 74
6.3 Conclusions . . . 78

7 Conclusions and Recommendations **79**
7.1 Conclusions . . . 79
7.2 Recommendations . . . 80

Bibliography **81**

List of Figures

1.1 Symbols for the Threshold Gates 1
1.2 Resistor-Transistor Circuit 3
1.3 AND, OR and NOT Gates Created with Threshold Logic 3

2.1 Graphical Representation of $(3 \mid 2)$, $(7 \mid 3)$ and $(15 \mid 4)$ Saturated Counters, and a $(2 \mid 2)$ Unsaturated Counter 10
2.2 Graphical Representation of $(2, 2, 2, 3 \mid 5)$ and $(4, 4 \mid 4)$ Counters 11
2.3 Using a $(7 \mid 3)$ Counter for a $(2, 3 \mid 3)$ Counter 11
2.4 Graphical Representation of a $m \times n$ Counter 15

3.1 Circuit of the Basic $(3 \mid 2)$ Counter 20
3.2 Element-Output Functions of the Basic $(3 \mid 2)$ Counter 20
3.3 Circuit of the Basic $(7 \mid 3)$ Counter 22
3.4 Circuit of the Basic $(15 \mid 4)$ Counter 24
3.5 Circuit of the Minnick $(7 \mid 3)$ Counter 29
3.6 Circuit of the Minnick $(3 \mid 2)$ Counter 30
3.7 Circuit of the Minnick $(15 \mid 4)$ Counter 31
3.8 Circuit of the Kautz $(7 \mid 3)$ Counter 35
3.9 Element-Output Functions of the Kautz $(7 \mid 3)$ Counter 35
3.10 Circuit of the Kautz $(3 \mid 2)$ Counter 36
3.11 Circuit of the Kautz $(15 \mid 4)$ Counter 37
3.12 General Kautz-Network 37

3.13 $(2,3 \mid 3)$ Counter with Reduced Input Lines 40

4.1 Addition of 8 15-bit Numbers with **0**-Order Counters. Dotted Lines Connect Inputs of a Counter, Full Lines the Outputs 46
4.2 Addition of 8 15-bit Numbers with **0**-Order Counters. (The numbers next to the horizontal lines denote the stage in the adding process.) . . . 47
4.3 Addition of 8 15-bit Numbers with Saturated **0**-Order Counters. 50
4.4 Addition of 8 15-bit Numbers with **1**-Order Counters. 51
4.5 Addition of 8 15-bit Numbers with **Logarithmic**-Order Counters. . . . 53
4.6 Addition of 8 15-bit Numbers with **Logarithmic**-Order Counters. (The numbers next to the horizontal lines denote the stage in the adding process.) 53

5.1 Example of Binary Multiplication 58
5.2 General Binary Multiplication 58
5.3 16x16 bit multiplication in a single stage using parallel counters. Carries are propagated through the counters 62
5.4 Logarithmic Partitioning of the 16x16 Bit Multiplication Matrix 64
5.5 Circuit of the serial-parallel multiplier counting one column at a time . . 68
5.6 Circuit of the serial-parallel multiplier counting two columns at a time . 69

6.1 Hierarchical Addition using Two Reduction Steps. 74
6.2 Block Save Addition of 8 15-bit Numbers 75
6.3 Addition of 2 16-bit Numbers, digits indicate the number of gate delays to calculate that output. 75
6.4 Hierarchical Block Save Addition Using Three Reduction Steps, All Carry Handling is Confined to the Last Step 77

List of Tables

2.1 Truth Table for a $(15 \mid 4)$ Counter 13

3.1 Truth Table of a Full Adder 19
3.2 Truth Table of a $(7 \mid 3)$ Counter 21
3.3 Truth Table for y_3, y_2 and y_1 of a $(15 \mid 4)$ Counter 23
3.4 Truth Table for y_0 of a $(15 \mid 4)$ Counter 23
3.5 Truth Table of y_0 of a Minnick $(7 \mid 3)$ Counter 29
3.6 Formulas for Comparison of the Different Design Methods 40
3.7 Number of Gates Used by the Different Designs 41
3.8 Cost of the Different Designs 41
3.9 Delay Required by the Different Designs 41
3.10 Maximum Fan-in Required by the Different Designs 42
3.11 Maximum Weight Required by the Different Designs 42

4.1 Cost Parameters for Multiple Adders, Using Kautz **0**-Order Counters. This Table Corresponds to Figure 4.1 50
4.2 Cost Parameters for Multiple Adders, using Kautz **1**-Order Counters. This Table Corresponds to Figure 4.4 51
4.3 Cost Parameters for Multiple Adders, using Kautz **Logarithmic**-Order Counters. This Table Corresponds to Figure 4.5 53

5.1 Cost and Delay of Partial Product Matrix Reduction Using Kautz **0**-Order Counters Calculated with the Algorithm 64

5.2 Cost and Delay of Partial Product Matrix Reduction Using Kautz Logarithmic Order Counters Calculated with the Algorithm 65
5.3 Cost and delay of partial product matrix reduction using Kautz **2n**-order counters 66
5.4 Cost Parameters of Serial-Parallel Partial Product Matrix Reduction Using Kautz **0**-Order Counters 70
5.5 Cost Parameters of Serial-Parallel Partial Product Matrix Reduction Using Kautz **Logarithmic**-Order Counters 70

6.1 Cost Parameters for Hierarchical Multiple adders, Using Kautz $(8, 8, 8 \mid 6)$ and Smaller Counters. 77

1 Introduction

Due to recent technological developments, see for example [SO92, SO93a, SO93b], threshold logic appears to be promising for the design of arithmetic units and other hardwired operations.

Formally speaking threshold logic, depicted symbolically in Figure 1.1, can be described in two different ways. In the first way a gate assumes the logical "one" when the weighted sum of the gate inputs is equal to or *greater* than some threshold value. In the second way a gate assumes the logical "one" when the weighted sum of the gate inputs is equal to or *less* than some threshold value.

Mathematically speaking, the "equal or greater than" element performs the following function:

$$y = \text{sgn}\left[\sum_{i=1}^{n} w_i x_i - T\right]$$

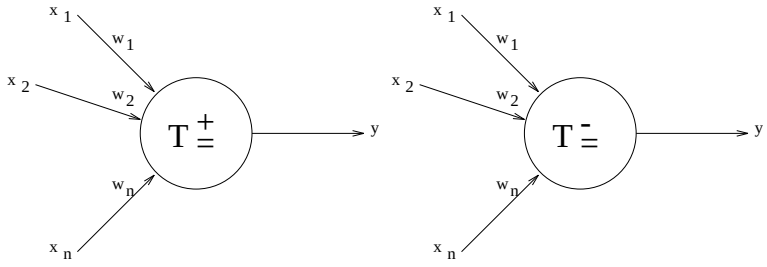

Figure 1.1: Symbols for the Threshold Gates

In the equation it is assumed that the threshold value is T, the n inputs are depicted by x_i, the weights by w_i and that for a given variable x the sgn(.) function performs the following operation:

$$\text{sgn(x)} = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{1.1}$$

The equation for the "equal or less than" element is as follows:

$$y = \text{sgn}\left[T - \sum_{i=1}^{n} w_i x_i\right]$$

With the sgn(.) function performing the same operation described previously in Equation 1.1.

To visualize the threshold devices, Figure 1.2 depicts a resistor transistor implementation. This circuit produces the inverse of the wanted function, thus it should be taken into account when used. The transistor conducts when the summed current through the input resistors times the R is greater than the basis-emitter voltage of the transistor, thus producing a low voltage at its output. This is also stated in Equation 1.2, with g_i the conductance of a resistor.

$$\text{y} = \begin{cases} 1 & \text{if } R\sum_{i=1}^{n} g_i x_i \geq U_{BE} \\ 0 & \text{if } R\sum_{i=1}^{n} g_i x_i < U_{BE} \end{cases} \tag{1.2}$$

This implementation is a slight variation of the one found in [Min61]. It is a very simple implementation, and does not support high speeds and high fan-in or fan-out gate requirements. More sophisticated resistor transistor implementations can be found in [CL64]. Modern implementations, including designs based on switched capacitors, are described in [LB91].

Regarding the usefulness of such gates consider the following discussion. It is well known that logical functions can be created using the boolean AND, OR and NOT gates. It can be shown that these basic gates can also be implemented with threshold logic. To prove such a statement consider a threshold gate with $w_1 = w_2 = \cdots = w_n = 1$ and the threshold value of n. This gate will yield a one if all inputs are one. Thus an AND gate can be emulated with a threshold element. Consequently, consider the threshold gate with $w_1 = w_2 = \cdots = w_n = 1$ and the threshold value of 1. A gate can be constructed that will produce an "one" output if at least one of the inputs is one. Thus an OR gate can be constructed using threshold gates. Finally, the NOT gate can be created by a negative weight and a threshold value of zero. Figure 1.3 shows the corresponding diagrams for the three basic functions. It is noted that in the figure a $\frac{+}{=}$ indicates the threshold of the gate. Consequently, a conclusion can be drawn suggesting that the AND, OR and NOT logic functions can be implemented with threshold elements. Since, as already stated, any switching function can be expressed using only these gates, any switching function can also be expressed by a network of threshold gates.

The overall conclusion is that, theoretically speaking, feed-forward (using threshold elements) networks are as powerful as the Boolean networks (they can compute what

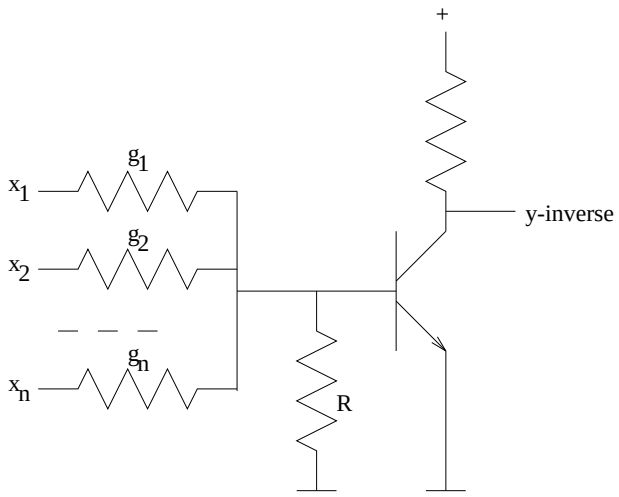

Figure 1.2: Resistor-Transistor Circuit

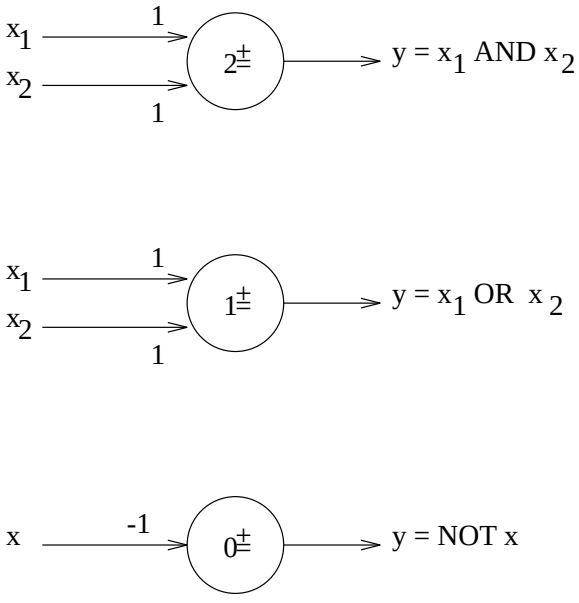

Figure 1.3: AND, OR and NOT Gates Created with Threshold Logic

Boolean networks can). This however (from the practical and commercial point of view) is half of the requirement. The other half relates to the feasibility, the speed, and the size of the network required for the computations. To quantify the limits, the possibilities, and the feasibility of the threshold logic based computation of Boolean functions, it is required to establish the parameters influencing the design and implementation of a Boolean function. In this context, investigations mainly focus on four parameters namely:

- the depth of the network (determined by the number of layers in the circuit),
- the size of the network (determined by the number of threshold gates),
- the maximum number of inputs required by the functional elements, and
- the size of the weights.

All investigations attempt to improve one or multiple parameters. For example, consider Boolean symmetric functions, i.e. boolean functions whose output depends only on the sum of their input values, with n variables. It has been shown by Muroga [Mur59] that such a class of functions can be computed by depth-2 feed-forward networks with $(n+1)$ threshold gates. This size, without increasing the depth, has been improved by Minnick [Min61] to be $(n \div 2 + 1)$ and recently in [PS90] it has been shown that concerning the parity function, one of the symmetric Boolean functions, a depth-2 network requires a size of at least $O(n \div log^2 n)$ threshold gates. Finally, by increasing the depth by one, Siu et al [SRK91] have shown that boolean symmetric functions require at most $2\sqrt{n} + 1$ threshold gates.

The study of general classes of Boolean functions, such as symmetric functions, using feed-forward networks, is useful in determining the upper/lower bounds of various parameters, e.g. depth and cost in terms of size, to be expected in general. This however does not dictate the limits to specific Boolean functions. It is entirely possible to improve the various limits of specific functions, such as arithmetic functions, by exploiting their specific requirements.

Regarding the addition and comparison operations the following has been established when feed-forward threshold gate based networks are assumed as the building components:

- For the binary addition, Siu et al [SRK91, SB90b] suggested that each bit of the sum is computable with depth-2 networks with a network size of $O(n^4)$ and that the network size can be reduced to $O(n^2)$ for depth-3 networks.
- In [SRK91] it has been indicated that the comparison function, performed in two operands of length n, can be computed in depth-2 networks with size of $O(n^4)$. Further with depth-3 networks, it has been suggested that the comparison can be realized with size of $O(n)$.

Given that the previously discussed schemes do not produce optimal size networks for the addition and comparison this open issue has been investigated recently and produced the following results:

- First, in [VBP94b] is shown that the 2-1 binary n-bit addition can be computed in a depth-2 network of optimal size $O(n)$ with maximum fan-in of 2n with unbounded weights.

- Based on the results obtained for unbounded weight networks for the sum, Vassiliadis and Bertels [VB94] proposed a new set of equations for the addition that allows the 2-1 binary n-bit addition to be computed in a depth-3 network of optimal asymptotic size $O(n)$ with maximum fan-in less than 2n and relatively small weights for existing architecture formats.

- Regarding the binary comparison first was proved that it can be computed in a depth-1 network of optimal size $O(1)$ with maximum fan-in of 2n+1 lowering both the size and depth theoretical bounds of the comparison [VBP94b].

- It has been proved that the comparison can be computed by a depth-2 network requiring an O($n \div logn$) size complexity with relatively small weights for existing architecture formats, improving both the depth and size [VBP94a].

- Regarding the practical consequences of the results is estimated that the proposed schemes for both addition and comparison provide substantial improvements when compared with the scheme proposed by Siu et al [SRK91], the scheme known to produce the best addition and comparison networks regarding the sizes with small network depths. For example for 32 and 64 bit operands is estimated that the proposed scheme requires respectively only 18.01% and 9.32% of the neurons required for the [SRK91] scheme for the addition and 11.45% and 8.85% for the comparison while maintaining or improving the depth of the network .

- Finally, it is proved that depth-3 bounded weights networks can be used to produce binary addition related operations, suggesting that the proposed schemes can be used in the design of the frequent binary n-bit integer operations defined by current computer architectures in constant small depth neural networks [VBP94a].

Regarding the multiplication, it has been shown [LB91, HHK91] that :

- The reduction of the multiplication matrix into two rows can be achieved in depth-2 neural networks with the cost of the network, in terms of number of wires being $O(N^2)$, the number of neurons required for the implementation being $O(N^2)$ and the maximum fan-in being $O(N \log N)$.

- The entire multiplication can be achieved by a depth-4 neural network.

1.1 Boolean Versus Threshold Logic and the Research Questions

The previous section provided background information and discussed some recent developments. In this subsection we review the issues leading to the research questions we intend to investigate. We note in advance that our interest is to investigate low cost implementations for counting functions and multiplication.

Regarding the multiplication, the best known reduction of the multiplication matrix is achieved using carry save addition techniques described first in [Wal64, Dad65]. In assuming the multiplication of 32X32 bits and a direct implementation then a depth-9 Boolean network is required to produce the partial product of two rows (which then are feed to a 2-1 adder that produces the final result). The same matrix reduction, independent of the operand lengths, can be achieved with a depth-2 threshold gate network [LB91, SB90a, HHK91]. If for the moment the cost in terms of threshold gates is excluded the advantage of using feed-forward networks is apparent. The cost however of the reduction in term of threshold gates is still large. The most cost effective method known today is presented in [LB91]. In assuming such a method, and assuming that the length of the operands is 32 bits, it has been suggested [LB91] that networks having 4966 neuron gates are required for the matrix reduction with the total number of wires being 593892. Clearly the cost is rather high. By augmenting the depth of the neural network, the cost can be improved however. This is indeed a research question we propose to investigate. The general question we intend to investigate can be formulated as follows:

- Can the size, in terms of threshold gates, for more complex arithmetic operations such as counting and multiplication be diminished?

We will show that there is an affirmative answer to this research question because the scheme presented in [LB91] for the multiplication did not use the telescopic sum [Min61] and the Kautz [Kau61] network, two ingenious schemes that substantially reduce the size of neural networks based on threshold gates possibly increasing the circuit depth.

In the presentation to follow we will investigate in particular the following:

- We will assume unrestricted weights, fan-in and gate delay to establish the lowest possible area required by an implementation of counters, multi-operand additions, and multiplications using threshold logic.
- For feasibility purposes we will restrict the weight and fan-in gate requirements, and investigate the influence such a restriction has to the area and delay of multi-operand additions and by extension to multiplications.

1.2 Framework

This report is organized as follows:

- Chapter 2 Describes theoretical background on parallel counters.
- Chapter 3 Shows three methods of designing the parallel counters with threshold logic. The methods are compared on cost and calculation delay. The investigation in this chapter provides us with the choices for the multiplication described in a later chapter.
- Chapter 4 Investigates multi-operand addition, and counters. Several schemes are considered and compared on cost, delay and weight and gate fan-in requirements.
- Chapter 5 Investigates the parallel counters in binary multipliers. Again, several schemes are considered and compared on cost, delay and weight and gate fan-in requirements.
- In both Chapters 4 and 5 we first assume the reduction of the area to be the major concern. We assume that all other parameters are not restricted by the technology. While such an assumption provides the potential lower count for the area the schemes may not be implementable. In Chapter 6 we pose restrictions on weight and gate fan-in requirements and investigate the influence such restrictions have on the area and delay.
- Chapter 7 gives the conclusions of the research, and recommendations for further investigations.

2 Parallel Counters

A $(p \mid q)$ parallel counter is a logical device that determines how many of its p inputs are in the logic one state, expressing the result as a binary number at its q outputs. A parallel counter can be viewed as a multiple-word input adder with p 1-bit words. In [Dad80], it is suggested that the counters are useful in the realization of fast multipliers, multiple-input adders, associative memories, computer arithmetic units and fast digital correlators. Clearly, because of their usefulness, a number of designs have been proposed. In a realistic engineering environment, the hardware used imposes a restriction on the maximum number of inputs of a counter. If counters with more inputs are needed they can be constructed by combining several smaller counters.

In this chapter we discuss counters in general and some of the issues regarding counter design. We organize the chapter as follows: section 2.1 introduces a graphical representation, to represent the counters in a convenient way. Section 2.2 gives the mathematical relations regarding in- and output values. In section 2.3 the functions for the outputs of counters are evaluated. A special organization known as the *block addition* is presented in 2.4. Finally, section 2.5 states the conclusions of this chapter and the relation to the next chapter.

2.1 Graphical Representation

To be able to represent the counters in a convenient way, we will use the graphical representation proposed by L. Dadda in [Dad80]. Some examples of this representation

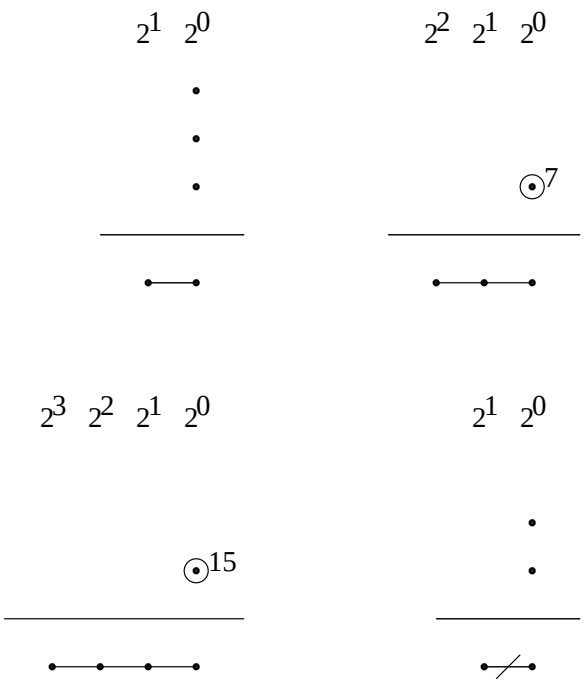

Figure 2.1: Graphical Representation of $(3 \mid 2)$, $(7 \mid 3)$ and $(15 \mid 4)$ Saturated Counters, and a $(2 \mid 2)$ Unsaturated Counter

for the counters are given in Figure 2.1.

The representation shows all relevant properties of a counter, that is the number of inputs an outputs and their respective weights. The outputs and inputs are depicted by dots, separated by a horizontal line. The dots in the same vertical line have the same weight, progressing from right to left by the number-base (in the binary case this is two). The output bits are connected by a horizontal line to show that they are outputs of the same counter, which makes things clearer in a composite situation. To make the diagrams more readable, a multitude of input bits is represented by a encircled dot, with the number of inputs written next to it.

The weights of the columns are stated above the dots. The maximum possible count of a $(p \mid q)$ counter is the maximum number that can be described by the q output bits, which is $2^q - 1$. If p is equal to $2^q - 1$, the counter is called *saturated*. The first three examples in Figure 2.1 are all saturated counters. Conversely, when this condition does not hold true, the counter is called *unsaturated*. The last example in Figure 2.1 shows a $(2 \mid 2)$ unsaturated counter which has the same function as a half-adder.

The class of counters is extended to allow inputs of different weights. A counter with the maximum input weight 2^i is called an **i**-order counter. The before mentioned counters all have inputs of weight $2^0 = 1$, so they are called **0**-order counters. Dadda notates an **i**-order counter as $(p_i, p_{i-1}, \cdots, p_0 \mid q)$, with p_i the number of weight 2^i inputs, and as before q the number of outputs. Figure 2.2 shows two examples. The first Figure

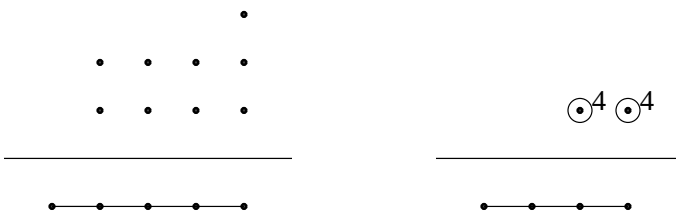

Figure 2.2: Graphical Representation of $(2, 2, 2, 3 \mid 5)$ and $(4, 4 \mid 4)$ Counters

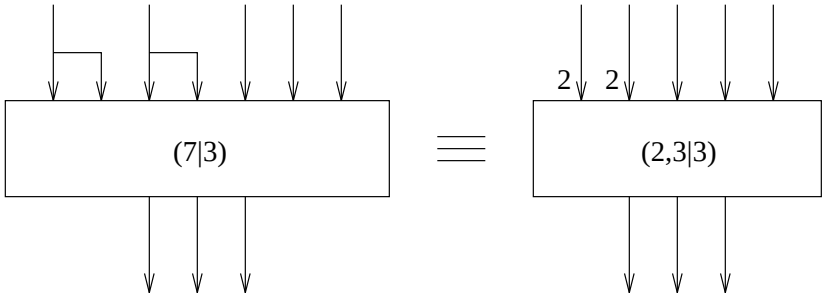

Figure 2.3: Using a $(7 \mid 3)$ Counter for a $(2, 3 \mid 3)$ Counter

is a $(2, 2, 2, 3 \mid 5)$ counter which acts like a 4-bit adder with carry in and carry out. The second Figure is a $(4, 4 \mid 4)$ non saturated counter.

It is possible to "miss use" any counter as one with less inputs or as one with different input weights. In the case with less inputs, set any unused inputs permanently to zero. This can result in the fact that also some outputs will not be necessary. For instance, if in a $(7 \mid 3)$ counter only 3 inputs are used, it is effectively an $(3 \mid 2)$ counter. The case with different input weights is more interesting. For example a **0**-order counter can be used as a **1**-order counter by grouping in parallel two inputs, for a signal applied to both inputs in parallel will be weighted two. Likewise, a **2**-order counter can be obtained from a **0**-order counter by grouping in parallel four inputs. The mixed variants are more appealing, for example make an $(2, 3 \mid 3)$ counter out of an $(7 \mid 3)$ and so on and so forth. See Figure 2.3.

The above mentioned procedure for creating higher order counters is convenient in certain situations, for example when no other counter is available or for reasons of uniformity when the same counter is used throughout the entire circuit. Nonetheless it wastes input pins, so a procedure which uses input weights instead of combining pins is preferred.

2.2 Input Considerations

The mathematical properties of the counters are given based on the general notation: $(p_i, p_{i-1}, \cdots, p_0 \mid q)$. The output value of this counter is given by:

$$v = \sum_{k=0}^{i} \sum_{j=0}^{p_k - 1} b_{ij} 2^i \tag{2.1}$$

With b_{ij} the value of bit j in column i. For such a counter, the maximum output value is given by:

$$C = \sum_{k=0}^{i} p_k 2^k \tag{2.2}$$

In this thesis the maximum output value is referred to as the "capacity". In Equation 2.2 the capacity is based on the input configuration, it is calculated as the maximum number the input can represent. In this thesis it is assumed that the number of output bits is sufficient to represent the maximum input count (the result will be a strange counter if this was not the case). To be able to represent the maximum count we need at least [1] [2] [3]:

$$\begin{aligned} q &= \lceil \log(C+1) \rceil \\ &= 1 + \lfloor \log(C) \rfloor \end{aligned} \tag{2.3}$$

output bits. More output bits is useless because they will never be triggered. The two variants of Equation 2.3 will both be used, depending on which is convenient in a certain situation. The $+1$ in both variants comes from the fact that to represent for example 8, we need 4 bits. While the log of 8 would be 3. And the same for all boundary values.

If the maximum input count is equal to the maximum output count, the counter is called *saturated*. Hence the counter will be saturated when

$$C = 2^q - 1 \tag{2.4}$$

This means that the output "fits" the input and no (fraction of an) output bit is spilled. In the chapter to follow regarding the calculation of the cost of various counters, we will mostly consider saturated counters. This is done to avoid cumbersome notation and because it gives us very nice and clean, easy to use, formulas. Furthermore, the saturated counter is the most expensive for a stated number of output bits. This is because if it is not saturated, less inputs are needed and potentially less gates. Thus is that case the formulas can be regarded as an upper bound.

2.3 Output Functions

To obtain the functions representing the output bits first observe the truth table of a $(15 \mid 4)$ counter in Table 2.1. The q output bits are notated by $y_{q-1}, y_{q-2} \cdots y_0$.

[1] $\lceil x \rceil$ is the smallest integer greater than or equal to x. Known from APL as the "ceiling".
[2] $\lfloor x \rfloor$ is the smallest integer less than or equal to x. Known from APL as the "floor".
[3] Throughout this thesis, log denotes logarithm to the base 2.

v	y_3	y_2	y_1	y_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Table 2.1: Truth Table for a $(15 \mid 4)$ Counter

In the truth table only the weighted input sum v (calculated with Equation 2.1) is given. This is possible because the functions generating the q_i's are *symmetric*. A symmetric function is a switching function which remains invariant to all permutations of the input variables. Thus it can be described uniquely by stating its truth value for the number of true inputs, which is done in Table 2.1. The number of true inputs, which is the sum of all inputs, is given in column v.

Another way to describe symmetric functions which will be used in this thesis, is by stating the *ranges* or the number of true inputs for which the function is true. For example in Table 2.1, output y_2 is true for $4 \leq v \leq 7$ and for $12 \leq v \leq 15$.

Given a symmetric binary function y with n inputs x that is defined to be true if the number of true inputs $v = \sum_{i=1}^{n} x_i$ lies inside the intervals $[s_j, S_j]$, and y is defined to be false outside these intervals. That is:

$$y(x_1, x_2, \cdots, x_n) = 1 \text{ for } v = \sum_{i=1}^{n} x_i \text{ inputs true, with } s_j \leq v \leq S_j, j = 1, 2, \cdots, r \tag{2.5}$$

where s_j, the lower bounds, and S_j, the upper bounds, are known for a certain function; r denotes the number of intervals.

The question remains whether in the case of the counters the range boundaries can be stated in an simple way. Looking at the truth tables , we observe that the first sequence of

ones in every column starts in the row equal to the weight of the column. Consequently, q_0 with weight $2^0 = 1$ has its first one in that row where v is one. The length of the range follows the same pattern. This leads to the general case for a q_i in Theorem 2.1. The proof of this theorem is derived from the proof of Theorem 1 in [HC73].

Theorem 2.1 *The i'th output y_i of a parallel counter with capacity C is true between the ranges:*

$$\begin{aligned} s_j &= 2^i + (j-1)2^{i+1} \\ S_j &= 2^{i+1} - 1 + (j-1)2^{i+1} \\ \textit{with } j &= 1, 2, \cdots, \frac{C+1}{2^{i+1}} \end{aligned}$$

Proof: Rewrite the input count v as a function of the outputs, and "isolate" y_i:

$$\begin{aligned} v &= \sum_{n=0}^{q-1} y_n 2^n \\ &= \sum_{n>i} y_n 2^n + y_i 2^i + \sum_{n<i} y_n 2^n \\ &= 2^{i+1} M + y_i 2^i + N \end{aligned}$$

With M, N nonnegative integers and $N \leq 2^i - 1$. Then

$$v \bmod 2^{i+1} = y_i 2^i + N \quad \begin{array}{ll} \geq & 2^i, \quad \text{if } y_i = 1 \\ < & 2^i, \quad \text{if } y_i = 0 \end{array}$$

If we put it the other way around and expand the mod operator to a multiplication:

$$\begin{array}{lll} \text{if } y_i = 1 & \text{then} & v + x \cdot 2^{i+1} \geq 2^i, \\ \text{if } y_i = 0 & \text{then} & v + x \cdot 2^{i+1} < 2^i, x \in N \end{array}$$

To explain this equation we will first assume a v rising from 0 upwards. y_i will become 1 when $v \geq 2^i$. When v reaches 2^{i+1} then $v \bmod 2^{i+1} = 0$ and y_i returns to zero. Because of the mod operator, this sequence will repeat every interval of length 2^{i+1}. Thus the intervals start at every $v = 2^i (\text{mod} 2^{i+1})$ and end one unit before $v = 2^{i+1} (\text{mod} 2^{i+1})$ which is $v = 2^{i+1} - 1 (\text{mod} 2^{i+1})$.

The total number of intervals is the maximum input count (the capacity) divided by the width of an interval, which is 2^{i+1}. Thus the index j goes up to that amount.

□

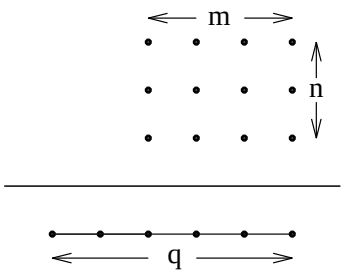

Figure 2.4: Graphical Representation of a $m \times n$ Counter

2.4 Block Addition

Using specific formats for the input matrix gives some simplifications to the formulas. Here we will investigate a rectangular array. The term block addition stems from the fact that the input matrix is rectangular. A block adder is notated as a BA(m, n, q), which adds n words of m bits resulting in q output bits. In our previous notation a $(n_{m-1}, n_{m-2}, \cdots, n_0 \mid q)$ adder. Wherein all n_i's equal n. Figure 2.4 shows a diagram.

Due to the regular structure, the maximum output value (capacity) is given by:

Theorem 2.2 *The capacity of a BA(m,n,q) counter is given by:*

$$C_{BA(m,n,q)} = n \times (2^m - 1) \tag{2.6}$$

Proof: Begin with Equation 2.2, and note that $p_k = n$ and $i = m - 1$ Equation 2.6 follows readily:

$$\begin{aligned} C &= \sum_{k=0}^{i} p_k 2^k \\ &= \sum_{k=0}^{m-1} n2^k = n \sum_{k=0}^{m-1} 2^k \\ &\Rightarrow n(2^m - 1) \end{aligned}$$

□

To be able to represent this maximum output value, combining Equations 2.3 and 2.6 gives:

$$q_{BA(m,n,q)} = 1 + \lfloor \log(n(2^m - 1)) \rfloor \tag{2.7}$$

This can be simplified to the following theorem:

Theorem 2.3 *The needed number of the output bits of a BA(m,n,q) counter is given by:*

$$q_{BA(m,n,q)} = m + \lceil \log(n) \rceil$$

Proof: Rewriting Equation 2.7 with the other variant of Equation 2.3 gives:

$$\begin{aligned} q_{BA(m,n,q)} &= \lceil \log(n(2^m - 1) + 1) \rceil \\ &= \lceil \log(n2^m - (n-1)) \rceil \\ &\leq \lceil \log(n2^m) \rceil = \lceil \log(2^{m+\log n}) \rceil \\ &= \lceil m + \log n \rceil = m + \lceil \log(n) \rceil \end{aligned}$$

□

Another significant simplification can be achieved if we restrain ourselves to n words of $\log n$ bits (see [LB91]), i.e. a BA$(\log n, n, q)$, with $\log n$ a natural number. In this case Equation 2.6 simplifies to:

$$\begin{aligned} C_{BA(\log n,n,q)} &= n(2^{\log n} - 1) \\ &= n(n-1) < n^2 \end{aligned}$$

and the result of Theorem 2.3 becomes:

$$\begin{aligned} q_{BA(\log n,n,q)} &= \log n + \lceil \log n \rceil \\ &= 2 \log n, \text{ for } \log n \in N \end{aligned} \tag{2.8}$$

It can be observed from Equation 2.8 an aspect of this metric of the counters is that the number of output bits is approximate twice the width (but not larger) of the input words. This leads to the fact that if these counters are used in parallel no overlap of more than one block away occurs, so that the odd and even blocks can be calculated independently. The significance of this will become clear in the chapter on multiplication.

As a last remark we state that an input column only influences the output bit direct beneath it and output bits to the right of it. That is of the same or higher signification. This can be seen by looking at the adder and perform the calculation by hand. Another way to see this is by looking at a truth table. It comes out that by adding an amount to a certain input column, the output value of a lower column stays the same, i.e. with $v = 3$ then $y_0 = 1$ and with $v = 3 + 2$ also $y_0 = 1$.

2.5 Conclusions

In this chapter we have provided the background information on parallel counters needed in the following chapters. We have described a convenient graphical representation which enables us to interpret circuits involving a multiple of parallel counters with ease. Furthermore a mathematical background, which includes the binary functions involved, has been discussed which will be used in following chapters to evaluate larger circuits built with parallel counters. A valuable item is that a regular input metric like the block addition leads to a simplification of the formulas.

In the next chapter it is shown how the parallel counters can be build with the threshold logic elements introduced in Chapter 1.

Design of Parallel Counters 3

An efficient design of elementary counters can be achieved with the threshold gates described in the introduction. Three design schemes will be reviewed in this chapter. As it will be shown, the three schemes are simple to use and provide circuits with little design effort.

To compare the different schemes the following parameters[1] are calculated:

- **Gates**: The total number of threshold gates in the circuit. This is proportional to the number of elements in an implementation (see chapter 1) of the threshold gates.

- **Cost**: The cost of a circuit is defined as the number of gates in the circuit, each multiplied with its number of inputs. This definition makes the cost proportional to the number of wires needed to implement the circuit which influences the chip area occupied by the design. For our threshold circuits, this corresponds to the number of synapses in the circuit, where a synaps implements the multiplication of an input x_i with a weight w_i in a threshold gate.

- **Fan-in** $_{\max}$: Is defined as the maximum number of inputs for the gates in the circuit. This parameter is important since the hardware puts an upper limit to the number of inputs of a single gate.

[1] The **Cost** and **Fan-in** $_{\max}$ parameters are equal to those used in [LB91].

- **Delay**: measured in terms of number of gate delays needed to stabilize all output bits of a circuit, or the maximum number of gates that a single signal has to travel through the circuit.
- **Weight** $_{max}$: Largest weight factor of a single gate in the circuit. This parameter is important because the hardware puts an upper limit to the precision and value of the weight factors.

These parameters will be referenced to as the "Cost and Delay" of the various circuits.

The differences of these parameters for the different schemes will be discussed in the last section of this chapter.

As already mentioned in the previous chapter (section 2.2) the formulas for the parameters get simpler when saturated counters are considered. The equations in this chapter will therefore be based on saturated counters. To be more precise, they are in most cases based on the number of outputs of a counter, and thus assuming a saturated number of input bits. This implies that for non-saturated counters these formulas should be looked upon as an upper limit, which in most cases is close to the exact answers.

When explaining the methods and deriving the calculations, we will consider at first the counters constructed with all inputs taken together (taking account of the weights) and these inputs are all together fed to the appropriate gates. This will be a correct assumption if we add all inputs to some analog line which leads to the gates. As mentioned in the previous chapter (section 2.4) it is not necessary to guide an input to the gates concerned only with the lower order output bits. Depending on the implementation we then get different formulas on number of connections in the circuit. This input lines optimization is mentioned in a separate section.

3.1 The Basic Method

This first design method is very straightforward. The drawback is that is uses many elements, and both the $T_{=}^{+}$ and $T_{=}^{-}$ types of the threshold devices are used. Which is not the case with the other methods. Nevertheless it provides some basic insight so it's worth looking at. In the next three subsections, three counters will be designed. By carefully examining them we will try to give the general case and some interesting remarks about the numbers of devices needed and the depth of the circuit, which influences the counting speed.

3.1.1 Design of a Basic $(3 \mid 2)$ Counter

To explain the method consider Table 3.1. In the first three columns the truth table of a $(3 \mid 2)$ counter is stated. This is a device which counts the number of one's in a three bit input word, and gives the result as a two bit binary word. This is exactly the same as the Full Adder. The truth table also shows some additional information, which we will use for the design.

v	y_1	y_0	for y_1	for y_0		
0	0	0			$1^-_=$	
1	0	1		$1^+_=$	$1^-_=$	
2	1	0	$2^+_=$	$1^+_=$		
3	1	1	$2^+_=$	$1^+_=$		$3^+_=$

Table 3.1: Truth Table of a Full Adder

The truth table column labeled y_1 states when the output y_1 should be true. This is for $v \geq 2$. In this very simple case, the function can be realized with one single $2^+_=$-gate. An effort has been made to show this in the table in the "for y_1" column. It states $2^+_=$ in the rows where that particular device would be triggered. In Figure 3.1 this device is at the underside of the figure.

The y_0 bit possesses an on and off pattern with the input. Still, it is possible to use only threshold gates to create this bit. Table 3.1 dictates how it is done. It resembles trying to close in the ranges where the one's occur (although the term range looks a bit overdone here, being only one bit long). Scanning Table 3.1 the first time that the y_0-bit turns one is when v is one. So we use the $1^+_=$ device to signal this. Scanning the table the y_0 bit turns zero in the next row already: $1^-_=$ signals this. For $v = 3$ y_0 is one again and we use $3^+_=$ to signal this. Note that we do not use the $3^-_=$ device, because that one would be triggered on all possible inputs, giving no extra information.

Thus far we defined the devices $1^+_=$, $1^-_=$ and $3^+_=$ which must be connected to give the single y_0 output bit. Looking closely at the table, the following is established: it appears that when y_0 must be logical one, two out of the three devices are triggered. Conversely, when y_0 must remain logical zero, one of the three is triggered. Ergo the three devices must be connected to a similar device with a threshold of two, which gives the y_0-bit. In Figure 3.2 the outputs of the mentioned devices are schematically depicted. Figure 3.1 shows the complete schematic of the counter.

Using the notations as explained in Chapter 1, we can write this schematic down in a compact form as:

$$\begin{aligned} y_0 &= \text{sgn}[1^+_= + 1^-_= + 3^+_= - 2] \\ y_1 &= 2^+_= \end{aligned}$$

The y_0 formula represents the addition of the outputs of the $1^+_=$, $1^-_=$ and $3^+_=$ devices into a second level device with a threshold of 2.

3.1.2 Basic $(7 \mid 3)$ Counter

The $(7 \mid 3)$ counter will be designed in the same way. Table 3.2 shows the truth table plus information on the threshold gates used. The most significant bit, y_2, is again the simplest. It is one if $v \geq 4$, thus a $4^+_=$ is used. The y_1 reveals more of the design

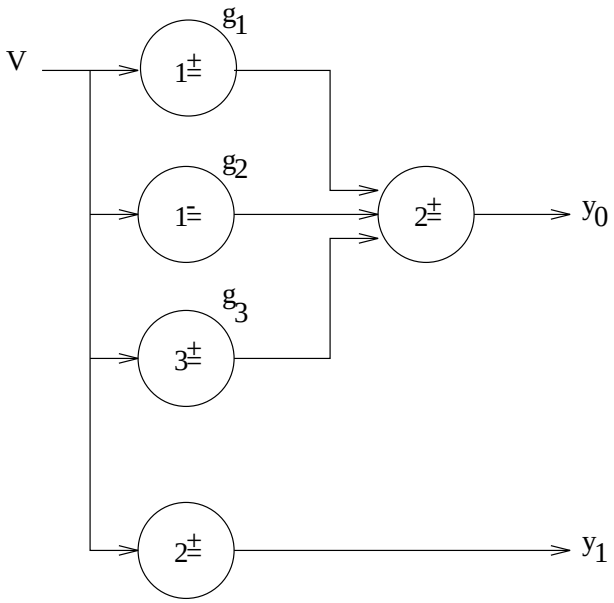

Figure 3.1: Circuit of the Basic $(3 \mid 2)$ Counter

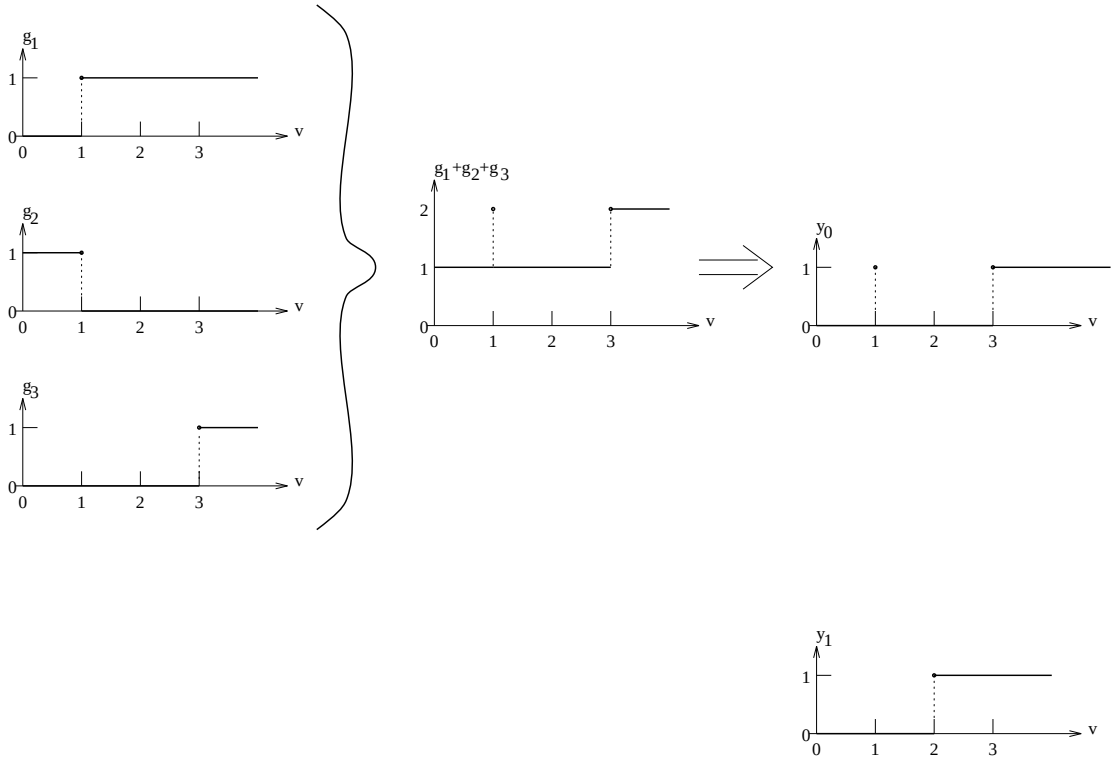

Figure 3.2: Element-Output Functions of the Basic $(3 \mid 2)$ Counter

v	y_2	y_1	y_0	for y_2	for y_1		for y_0					
0	0	0	0			$3^-_=$		$1^-_=$		$3^-_=$		$5^-_=$	
1	0	0	1			$3^-_=$	$1^+_=$	$1^-_=$		$3^-_=$		$5^-_=$	
2	0	1	0		$2^+_=$	$3^-_=$	$1^+_=$			$3^-_=$		$5^-_=$	
3	0	1	1		$2^+_=$	$3^-_=$	$1^+_=$		$3^+_=$	$3^-_=$		$5^-_=$	
4	1	0	0	$4^+_=$	$2^+_=$		$1^+_=$		$3^+_=$			$5^-_=$	
5	1	0	1	$4^+_=$	$2^+_=$		$1^+_=$		$3^+_=$		$5^+_=$	$5^-_=$	
6	1	1	0	$4^+_=$	$2^+_=$	$6^+_=$	$1^+_=$		$3^+_=$		$5^+_=$		
7	1	1	1	$4^+_=$	$2^+_=$	$6^+_=$	$1^+_=$		$3^+_=$		$5^+_=$		$7^+_=$

Table 3.2: Truth Table of a $(7 \mid 3)$ Counter

method. Again we close in the ranges of ones for this bit. Here it gives us two ranges: $2 \leq v \leq 3$ and $6 \leq v \leq 7$. Because $v \leq 7$ is always true with this counter, the second range degrades to $6 \leq v$. To create these inequalities with threshold gates we use $2^+_=$, $3^-_=$ and $6^+_=$ gates. The table makes clear that y_1 should be one if two of them are one. For the y_0 bit a similar scheme is followed and Figure 3.3 shows the result.

The formulas are:

$$\begin{aligned} y_0 &= \text{sgn}[1^+_= + 1^-_= + 3^+_= + 3^-_= + 5^+_= + 5^-_= + 7^+_= - 4] \\ y_1 &= \text{sgn}[2^+_= + 3^-_= + 6^+_= - 2] \\ y_2 &= 4^+_= \end{aligned}$$

The -4 and -2 are the threshold of the second level gates. An important fact can be noted here: both the y_0 and the y_1 outputs use a $3^-_=$ device. In building the circuit we can use only one and connect it to both, thereby needing one device less. The schematic can be seen in Figure 3.3.

3.1.3 A $(15 \mid 4)$ Counter

As the last example a $(15 \mid 4)$ counter is given. Tables 3.3 and 3.4 show the truth table. In the same way as the last two sections, the ranges of ones are enclosed by the devices. The equations can be written by considering the truth table. The schematic can be seen in Figure 3.4. Note that some gates can be shared again. A more close look reveals that the $T^-_=$ gates only have to be defined for the y_0 bit. All other bits use this $T^-_=$ gates and only need additional $T^+_=$ gates.

$$\begin{aligned} y_0 &= \text{sgn}[1^+_= + 1^-_= + 3^+_= + 3^-_= + 5^+_= + 5^-_= + 7^+_= + 7^-_= + 9^+_= + 9^-_= \\ &\quad + 11^+_= + 11^-_= + 13^+_= + 13^-_= + 15^+_= - 8] \\ y_1 &= \text{sgn}[2^+_= + 3^-_= + 6^+_= + 7^-_= + 10^+_= + 11^-_= + 14^+_= - 4] \\ y_2 &= \text{sgn}[4^+_= + 7^-_= + 12^+_= - 2] \\ y_3 &= 8^+_= \end{aligned}$$

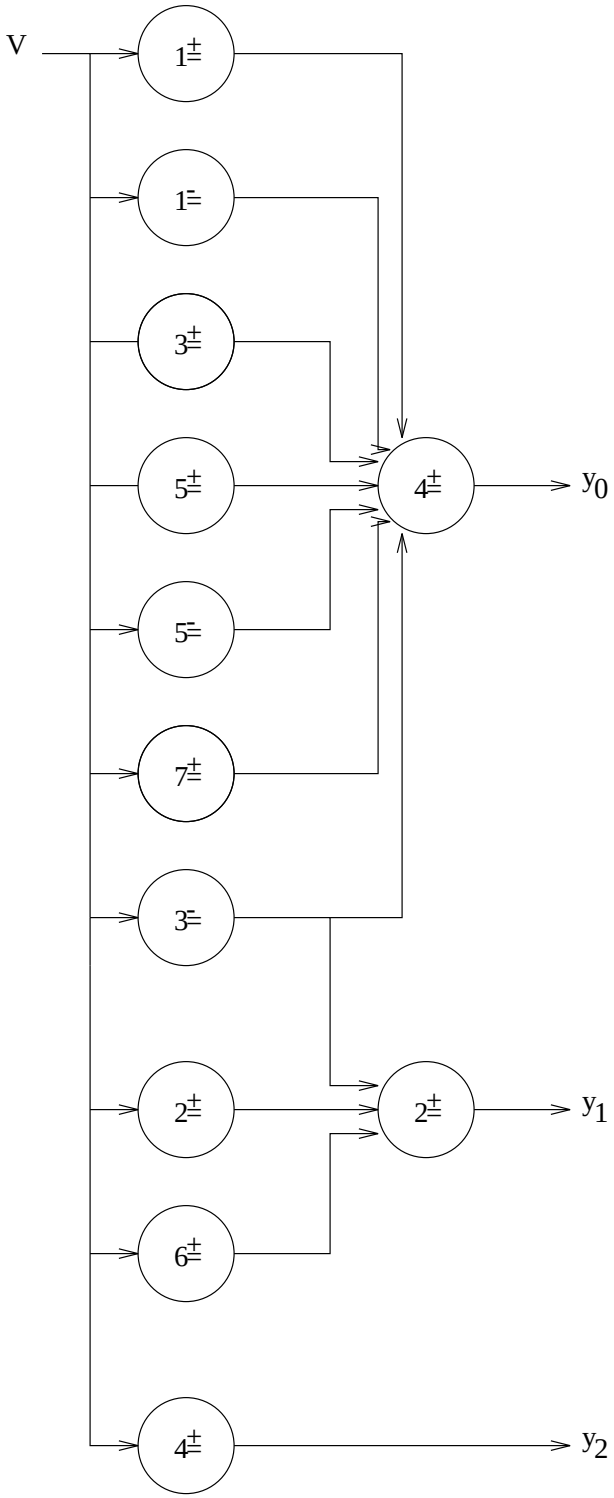

Figure 3.3: Circuit of the Basic $(7 \mid 3)$ Counter

v	y_3	y_2	y_1	y_0	for y_3	for y_2	for y_1
0	0	0	0	0		$7^-_=$	$3^-_=$ $7^-_=$ $11^-_=$
1	0	0	0	1		$7^-_=$	$3^-_=$ $7^-_=$ $11^-_=$
2	0	0	1	0		$7^-_=$	$2^+_=$ $3^-_=$ $7^-_=$ $11^-_=$
3	0	0	1	1		$7^-_=$	$2^+_=$ $3^-_=$ $7^-_=$ $11^-_=$
4	0	1	0	0		$4^+_=$ $7^-_=$	$2^+_=$ $7^-_=$ $11^-_=$
5	0	1	0	1		$4^+_=$ $7^-_=$	$2^+_=$ $7^-_=$ $11^-_=$
6	0	1	1	0		$4^+_=$ $7^-_=$	$2^+_=$ $6^+_=$ $7^-_=$ $11^-_=$
7	0	1	1	1		$4^+_=$ $7^-_=$	$2^+_=$ $6^+_=$ $7^-_=$ $11^-_=$
8	1	0	0	0	$8^+_=$	$4^+_=$	$2^+_=$ $6^+_=$ $11^-_=$
9	1	0	0	1	$8^+_=$	$4^+_=$	$2^+_=$ $6^+_=$ $11^-_=$
10	1	0	1	0	$8^+_=$	$4^+_=$	$2^+_=$ $6^+_=$ $10^+_=$ $11^-_=$
11	1	0	1	1	$8^+_=$	$4^+_=$	$2^+_=$ $6^+_=$ $10^+_=$ $11^-_=$
12	1	1	0	0	$8^+_=$	$4^+_=$ $12^+_=$	$2^+_=$ $6^+_=$ $10^+_=$
13	1	1	0	1	$8^+_=$	$4^+_=$ $12^+_=$	$2^+_=$ $6^+_=$ $10^+_=$
14	1	1	1	0	$8^+_=$	$4^+_=$ $12^+_=$	$2^+_=$ $6^+_=$ $10^+_=$ $14^+_=$
15	1	1	1	1	$8^+_=$	$4^+_=$ $12^+_=$	$2^+_=$ $6^+_=$ $10^+_=$ $14^+_=$

Table 3.3: Truth Table for y_3, y_2 and y_1 of a $(15 \mid 4)$ Counter

v	y_0	for y_0
0	0	$1^-_=$ $3^-_=$ $5^-_=$ $7^-_=$ $9^-_=$ $11^-_=$ $13^-_=$
1	1	$1^+_=$ $1^-_=$ $3^-_=$ $5^-_=$ $7^-_=$ $9^-_=$ $11^-_=$ $13^-_=$
2	0	$1^+_=$ $3^-_=$ $5^-_=$ $7^-_=$ $9^-_=$ $11^-_=$ $13^-_=$
3	1	$1^+_=$ $3^+_=$ $3^-_=$ $5^-_=$ $7^-_=$ $9^-_=$ $11^-_=$ $13^-_=$
4	0	$1^+_=$ $3^+_=$ $5^-_=$ $7^-_=$ $9^-_=$ $11^-_=$ $13^-_=$
5	1	$1^+_=$ $3^+_=$ $5^+_=$ $5^-_=$ $7^-_=$ $9^-_=$ $11^-_=$ $13^-_=$
6	0	$1^+_=$ $3^+_=$ $5^+_=$ $7^-_=$ $9^-_=$ $11^-_=$ $13^-_=$
7	1	$1^+_=$ $3^+_=$ $5^+_=$ $7^+_=$ $7^-_=$ $9^-_=$ $11^-_=$ $13^-_=$
8	0	$1^+_=$ $3^+_=$ $5^+_=$ $7^+_=$ $9^-_=$ $11^-_=$ $13^-_=$
9	1	$1^+_=$ $3^+_=$ $5^+_=$ $7^+_=$ $9^+_=$ $9^-_=$ $11^-_=$ $13^-_=$
10	0	$1^+_=$ $3^+_=$ $5^+_=$ $7^+_=$ $9^+_=$ $11^-_=$ $13^-_=$
11	1	$1^+_=$ $3^+_=$ $5^+_=$ $7^+_=$ $9^+_=$ $11^+_=$ $11^-_=$ $13^-_=$
12	0	$1^+_=$ $3^+_=$ $5^+_=$ $7^+_=$ $9^+_=$ $11^+_=$ $13^-_=$
13	1	$1^+_=$ $3^+_=$ $5^+_=$ $7^+_=$ $9^+_=$ $11^+_=$ $13^+_=$ $13^-_=$
14	0	$1^+_=$ $3^+_=$ $5^+_=$ $7^+_=$ $9^+_=$ $11^+_=$ $13^+_=$
15	1	$1^+_=$ $3^+_=$ $5^+_=$ $7^+_=$ $9^+_=$ $11^+_=$ $13^+_=$ $15^+_=$

Table 3.4: Truth Table for y_0 of a $(15 \mid 4)$ Counter

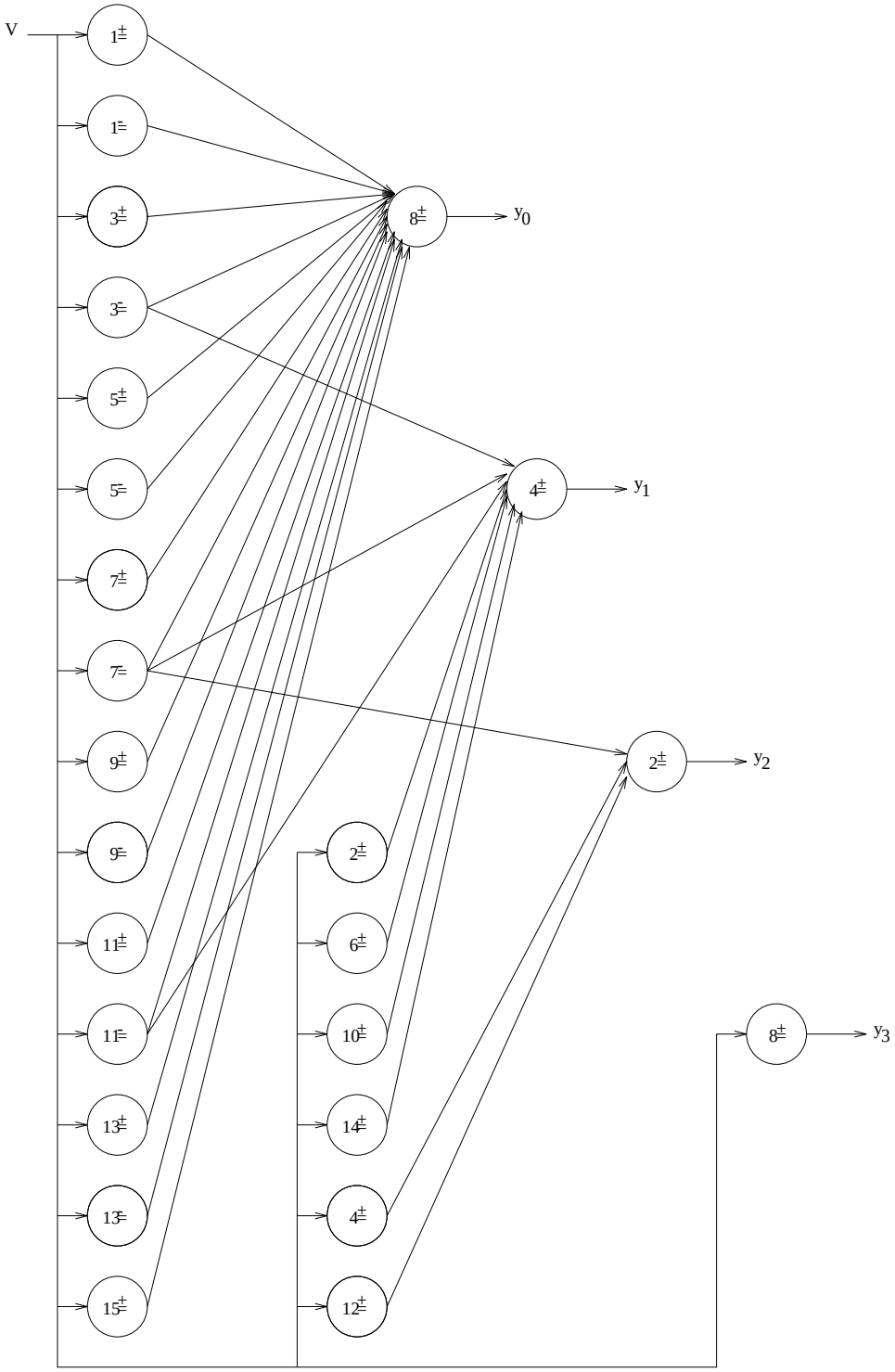

Figure 3.4: Circuit of the Basic $(15 \mid 4)$ Counter

3.1.4 The General Case of the Basic Counters

This design method works by enclosing the ranges of the truth table for which the function is true. That is, enclosing the lower bound with a positive element and the upper bound with a negative element.

With the notation, as in Equation 2.5, the general case of one bit of the circuit can be stated as in the theorem to follow:

Theorem 3.1 *A symmetric binary function y with n inputs x that is defined to be true if the number of true inputs $v = \sum_{i=1}^{n} x_i$ lies inside the intervals $[s_j, S_j]$, and y is defined to be false outside these intervals, can be constructed with threshold logic in the Basic Method by constructing a circuit like:*

$$y = sgn[s_{1=}^{+} + S_{1=}^{-} + s_{2=}^{+} + S_{2=}^{-} + \cdots + s_{r=}^{+} + S_{r=}^{-} - (r+1)] \tag{3.1}$$

Proof: The proof considers separately the cases y is true and y is false.
Assume a v for which y is true. Then there is a j for which $s_j \leq v \leq S_j$. Because the intervals do not overlap (and if they were, redefine to make them non-overlapping) we have $S_{j-1} < s_j$. Then the first level $s_{i=}^{+}$ are true only for $1 \leq i \leq j$. And the $S_{i=}^{-}$ are true for $j \leq i \leq r$. In this case Equation 3.1 becomes:

$$y = \text{sgn}[j + (r - j + 1) - (r + 1)] = \text{sgn}[0] = 1$$

Assume a v for which y is false. Then there is a j for which $S_j < v < s_{j+1}$. Then the first level $s_{i=}^{+}$ are true only for $1 \leq i \leq j$. And the $S_{i=}^{-}$ are true for $j + 1 \leq i \leq r$. In this case Equation 3.1 becomes:

$$y = \text{sgn}[j + (r - (j + 1) + 1) - (r + 1)] = \text{sgn}[-1] = 0$$

□

This can be seen as that for every interval except the one you're in, the upper or the lower bound is "one". Of the one you're in both are "one". Thus if $r + 1$ are one, the function is true. If, as in the case of saturated counters, the upper bound is equal to the maximum of v an simplification can be made. Because then $S_{r=}^{-}$ will always be true (of-course a similar argument can be made in case a lower bound is equal to zero). Thus this device can be omitted and subtract one from the overall threshold to conclude the following:

Theorem 3.2 *A saturated counter can be constructed with threshold logic in the Basic Method by constructing for every output bit $y_i, i = 0, 1, \cdots, q-1$ a circuit like:*

$$y_i = sgn[s_{i,1=}^{+} + S_{i,1=}^{-} + s_{i,2=}^{+} + S_{i,2=}^{-} + \cdots + s_{i,r_i=}^{+} - r_i]$$

Proof: Trivial from Theorem 3.1 and Theorem 2.1.

□

3.1.5 Cost and Delay of the Basic Method Counters

In this section we calculate the cost and delay parameters as explained in the introduction to this chapter for the counters designed with the Basic Method.

For the number of connections for higher order counters we assume that the higher order inputs are created by using the proper input weights. This implies that the total number of inputs coming into the circuit as a whole equals the sum of the p_i's. On the other side, for the determination of the number of output bits the maximum value of v, which is the capacity as calculated in Equation 2.2, is important.

Circuits designed in the Basic Method are always organized in two levels of threshold gates. In the derivation of the formulas we will therefore often calculate the cost and delay parameters separately for the two levels and add them to get the formulas for the whole circuit.

Gates

We start the calculation with the first level gates, that is the gates connected directly to the inputs from outside the circuit. In the first level of gates both the $T^+_=$ and the $T^-_=$ gates are used. First we calculate the number of $T^+_=$ gates in the first level. The most significant bit y_{q-1} uses one gate, for it has only one positive threshold to detect. The lesser significant output bits iteratively double this amount because we are counting binary. This gives 2^{q-1-i} for the number of $T^+_=$ gates in the first level of y_i. Summed over all output bits this gives:

$$\text{Number of } T^+_= \text{ Gates in the first level} = \sum_{i=0}^{q-1} 2^{q-1-i} = \sum_{i=0}^{q-1} 2^i = 2^q - 1$$

To calculate the number of $T^-_=$ gates notice that we define only $T^-_=$ gates for the y_0 output bit, and the other bits get them for free. Also, the $T^-_=$ gates are only used in the first level of gates. In general the number is equal to the number of $T^+_=$ gates in the first level of y_0. However, in the case of saturated counters one gate is always triggered and so can be deleted. Hence the number of $T^-_=$ gates $= 2^{q-1} - 1$.

In the second level every single output bit uses a second level gate to form its desired output function, with the exception of the most significant bit y_{q-1} for its second level gate degenerates to a redundant one-input $1^+_=$. Hence the number of gates in the second level $= q - 1$. Adding the three cases gives the following equation:

$$\begin{aligned} \text{Gates} &= \text{First level}(T^+_= + T^-_=) + \text{Second level}(T^+_=) \\ &= 2^q - 1 + 2^{q-1} - 1 + q - 1 = q + 3(2^{q-1} - 1) \end{aligned}$$

Cost

The number of inputs in the circuit is calculated by splitting the network again in two parts. The inputs from outside who enter all first level gates, and the inputs of the

second level gates who come from the first level gates. In the first level each device has $P = \sum_{k=0}^{i} p_k$ inputs. Thus multiplying P with the number of gates in the first level gives:

$$\begin{aligned} \text{Cost of the first level} &= P \cdot (2^q - 1 + 2^{q-1} - 1) \\ &= P \cdot (3 \cdot 2^{q-1} - 2) \end{aligned}$$

The gates in the second level connect to the gates in the first level that enclose the appropriate ranges for an output gate. Then the number of connections is effectively twice the number of ranges. Again, in the saturated case subtract one because the upper limit is equal to the maximum input capacity for saturated counters. The most significant bit does not incorporate a second level gate therefore that bit is not taken into account. The inputs form a nice sum.

$$\begin{aligned} \text{Cost of the second level} &= \sum_{i=0}^{q-2}(2^{q-1-i} + 2^{q-1-i} - 1) \\ &= \sum_{i=0}^{q-2}(2^{q-i} - 1) \\ &= \sum_{i=2}^{q}(2^i - 1) \\ &= \sum_{i=2}^{q}(2^i) - (q-1) \\ &= \sum_{i=0}^{q}(2^i) - 2^0 - 2^1 - (q-1) \\ &\Rightarrow 2^{q+1} - q - 3 \end{aligned}$$

Adding the two equations the desired result follows.

$$\text{Cost} = P \cdot (3 \cdot 2^{q-1} - 2) + 2^{q+1} - q - 3$$

Delay

The delay of the Basic method counters is always two, as this is a two layer method. In some cases, for example with carry propagation, it is worthwhile to know that the most significant bit needs only one calculation delay.

$$\begin{aligned} \text{Delay of } y_0, y_1, \cdots, y_{q-2} &= 2 \\ \text{Delay of } y_{q-1} &= 1 \end{aligned}$$

Fan-in $_{\text{max}}$

At the first sight, the highest Fan-in appears to occur at the gates in the first level, being equal to the number of input lines $P = \sum_{k=0}^{i} p_k$. This can however change if higher order counters are considered. In that case P will drop but the number of inputs of the y_0 output gate stays the same thus the highest fan-in occurs at the output gate of the least significant output bit. Because of the construction of the Basic Method counters this number is equal to the capacity C of the counter:

$$\text{Fan-in}_{\text{max}} = C$$

Weight $_{\text{max}}$

The maximum weight occurs as the threshold of the gate which encloses the highest range, which is C again.

$$\text{Weight}_{\text{max}} = C$$

3.2 The Minnick Method

In [Min61], R.C. Minnick presents a design method for symmetric functions. Like the Basic Method this is also a method with two levels of logic gates. The big difference with the Basic method is that Minnick uses weights between the first and second layer of the circuits. As will be shown, this gives an significant drop in elements used. The Minnick also uses only the positive elements, thus it creates simpler circuits.

3.2.1 Design of a Minnick $(7 \mid 3)$ Counter

To illustrate the method we start right of with the $(7 \mid 3)$ counter, for a $(3 \mid 2)$ has a too short truth table to clearly explain what is going on. Simply said, this method is a threshold-shifter. That is, the v is also connected directly to the second layer output device and the threshold of the output device is influenced by the first layer input devices to get a correct output.

It is advisable to look on the circuit in Figure 3.5 when reading the following explanation. Going down Table 3.5, the first time the output bit turns one is when v turns one. Only this time this is not signalled by an first layer device, but directly in the summing layer. Therefore we chose the threshold of the summing device equal to one. To get the output turn zero at $v = 2$ an $2^+_=$ is used to increase the output threshold to three. And indeed arriving at three the output turns one. A $4^+_=$ signals the end of it and increases the threshold to 5. Arriving at five the output turns one. At six an $6^+_=$ is used to increase the threshold to seven.

The foregoing paragraph reveals one disturbing property of Minnick counters: spikes are introduced. Because when we arrive at for example four, the apparent threshold of the output gate is three. The threshold becomes five after one gate delay of the responsible first layer gate. Thus this method is not applicable to designs who are sensitive to such spikes. The spikes only occur when counting up, not when counting down.

Note that for the next output bit, only an output gate is needed, because the first layer devices that it needs can all be found in the y_0 circuit. The circuit can also be described in the known formulas as has been done below:

$$\begin{aligned} y_0 &= \text{sgn}[v - 1 - 2 \cdot 2^+_= - 2 \cdot 4^+_= - 2 \cdot 6^+_=] \\ y_1 &= \text{sgn}[v - 2 - 4 \cdot 4^+_=] \\ y_2 &= \text{sgn}[v - 4] = 4^+_= \end{aligned}$$

v	y_0	Triggered devices			Apparent threshold of g_1
0	0				1
1	1				1
2	0	$2^{+}_{=}$			3
3	1	$2^{+}_{=}$			3
4	0	$2^{+}_{=}$	$4^{+}_{=}$		5
5	1	$2^{+}_{=}$	$4^{+}_{=}$		5
6	0	$2^{+}_{=}$	$4^{+}_{=}$	$6^{+}_{=}$	7
7	1	$2^{+}_{=}$	$4^{+}_{=}$	$6^{+}_{=}$	7

Table 3.5: Truth Table of y_0 of a Minnick $(7 \mid 3)$ Counter

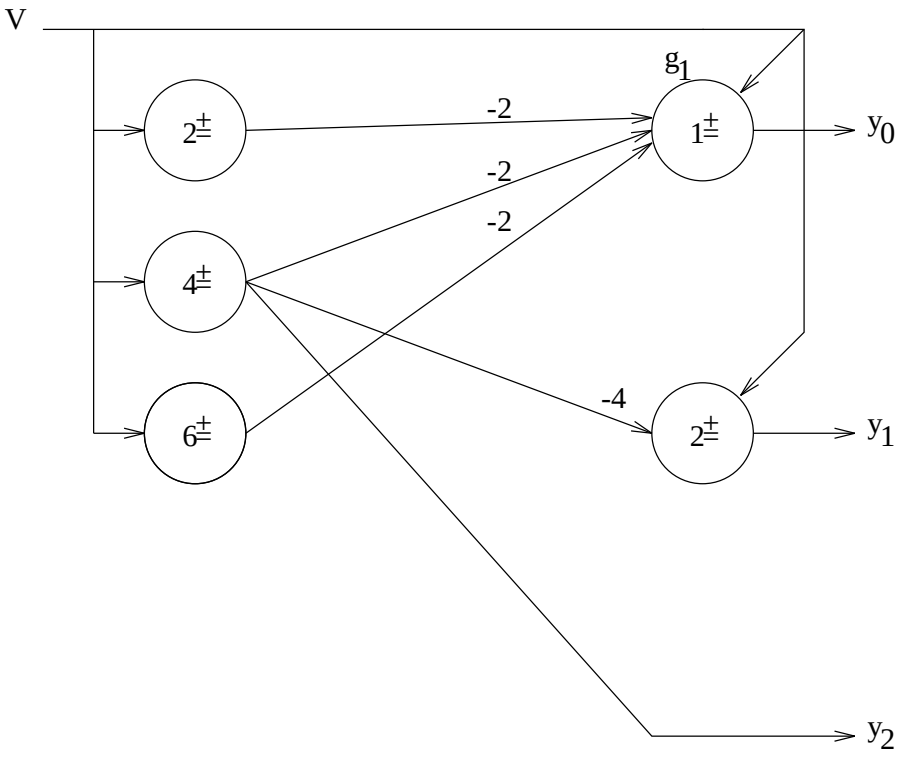

Figure 3.5: Circuit of the Minnick $(7 \mid 3)$ Counter

The v in the formulas stems from the fact that the inputs are also connected to the second layer of gates. The product factors indicate that weights are being used between first and second level gates.

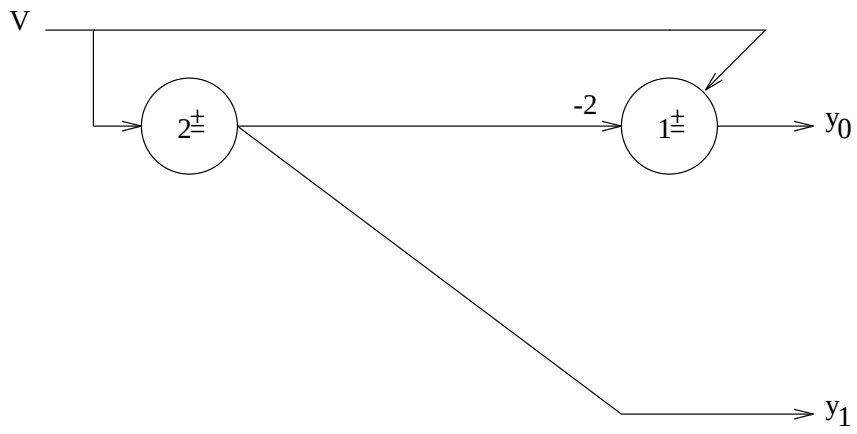

Figure 3.6: Circuit of the Minnick $(3 \mid 2)$ Counter

3.2.2 A $(3 \mid 2)$ Counter

To have the same examples for all three design methods, we show the Minnick $(3 \mid 2)$ counter also. Looking at the truth table in Table 3.1 we can extract the ranges, and write down the equations immediately. Figure 3.6 shows the result in a circuit.

$$\begin{aligned} y_0 &= \text{sgn}[v - 1 - 2 \cdot 2^{+}_{=}] \\ y_1 &= \text{sgn}[v - 2] = 2^{+}_{=} \end{aligned}$$

3.2.3 A $(15 \mid 4)$ Counter

Looking at the truth table in Tables 3.3 and 3.4 we can extract the ranges, and write down the equations immediately. Figure 3.7 shows the result in a circuit.

$$\begin{aligned} y_0 &= \text{sgn}[v - 1 - 2 \cdot 2^{+}_{=} - 2 \cdot 4^{+}_{=} - 2 \cdot 6^{+}_{=} - 2 \cdot 8^{+}_{=} - 2 \cdot 10^{+}_{=} - 2 \cdot 12^{+}_{=}] \\ y_1 &= \text{sgn}[v - 2 - 4 \cdot 4^{+}_{=} - 4 \cdot 8^{+}_{=} - 4 \cdot 12^{+}_{=}] \\ y_2 &= \text{sgn}[v - 4 - 8 \cdot 8^{+}_{=}] \\ y_3 &= \text{sgn}[v - 8] = 8^{+}_{=} \end{aligned}$$

3.2.4 The General Case of the Minnick Counters

Mathematically, the Minnick method is stated the following way, with the notation as in Equation 2.5:

Theorem 3.3 *An output bit y_i of the counter can be constructed in the Minnick method like:*

$$y_i = \textit{sgn}[v - c_{i,0} - c_{i,1} \cdot (S_{i,1} + 1)^{+}_{=} - c_{i,2} \cdot (S_{i,2} + 1)^{+}_{=} - \cdots - c_{i,r_i} \cdot (S_{i,r_i} + 1)^{+}_{=}] \quad (3.2)$$

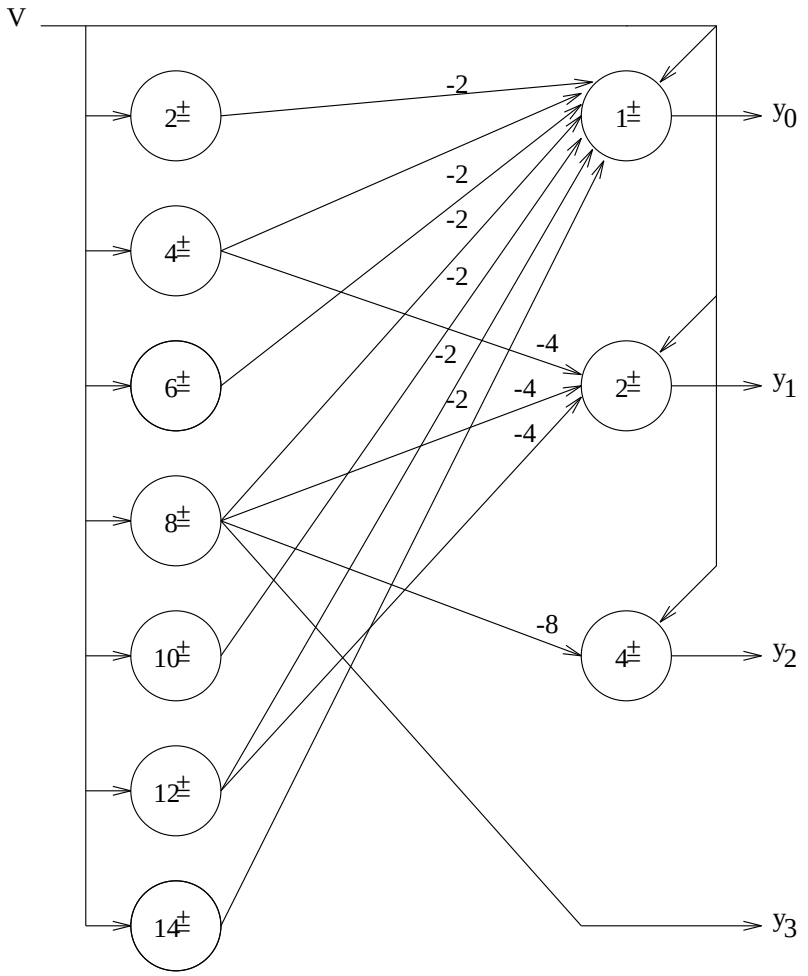

Figure 3.7: Circuit of the Minnick $(15 \mid 4)$ Counter

Where:

$$\begin{aligned} c_{i,0} &= s_{i,0}, c_{i,j} = s_{i,j+1} - s_{i,j}, j = 1, 2, \cdots, r-1 \\ c_{i,r} &= \begin{cases} n + 1 - q_{i,r} & \text{if } S_{i,r} \neq n \\ 0 & \text{if } S_{i,r} = n \end{cases} \end{aligned} \tag{3.3}$$

Proof: As stated, this design method comes from R.C. Minnick in [Min61]. In this article an mathematical proof is given, therefore we only explain the Theorem here.

□

The functioning of this circuit is a little bit harder to get than the circuit with the basic method. Lets assume a v for which y_i is true. Then there is a j for which $s_{i,j} \leq v \leq S_{i,j}$. Because the intervals do not overlap (for if they were, we'd have them redefined to get them non-overlapped) we have $S_{i,j-1} < s_{i,j}$. It now follows that the first-level circuits $(S_{i,k} + 1)^{+}_{=}$ are true only for $0 \leq k \leq j-1$. Taking this into account, Equation 3.2 can be rewritten for this case:

$$y_i = \text{sgn}[v - \sum_{k=0}^{j-1} c_{i,k}] = \text{sgn}[v - s_{i,j}]$$

And because $v \geq s_{i,j}$ of the inputs is true, y_i is true as required. Now is is very easy to show that the function is false for $S_{i,j-1} \leq v \leq s_{i,j}$. This follows immediately because the above simplification of Equation 3.2 remains the same while $v < s_{i,j}$, so the device won't be triggered.

Note that in the saturated case one device less is needed, as follows from the coefficients in Equation 3.3.

3.2.5 Cost and Delay of Minnick Counters

In this section we calculate the cost and delay parameters as explained in the introduction to this chapter for the counters designed with the Minnick Method.

Like we did with the Basic Method, for the number of connections for higher order counters we assume that the higher order inputs are created by using the proper input weights. This implies that the total number of inputs coming into the circuit as a whole equals the sum of the p_i's. On the other side, for the determination of the number of output bits the maximum value of v, which is the capacity as calculated in Equation 2.2, is important.

Circuits designed in the Minnick Method are always organized in two levels of threshold gates. In the derivation of the formulas we will therefore often calculate the cost an delay parameters separately for the two levels and add them to get the formulas for the whole circuit.

Gates

The sizes of the circuit are easily found. One of the main advantages of this design method is that the least significant output bit has all the needed first layer threshold devices which are used also for the other output bits. The second level threshold gate connected to a certain output bit itself takes care of the lower bound of the first range, and the rest is done with first layer devices.

The number of first level gates is equal to the number of upper-bounds in the creation of y_0. When the counter is saturated one is subtracted because like with the Basic Method this gate would always be triggered. The number of upper bounds is calculated in Theorem 2.1. This gives:

$$\text{Number of Gates in the first level} = 2^{q-1} - 1$$

Like with the Basic method, every single output bit uses a second level gate to form it's desired output function, with the exception of the most significant bit y_{q-1} for its second level gate degenerates to a redundant one-input $1^{+}_{=}$, as with the Basic method. Hence the number of gates in the second level $= q - 1$.

Adding the two cases gives:

$$\begin{aligned} \text{Gates} &= \text{First level gates} + \text{Second level gates} \\ &= 2^{q-1} + q - 2 \end{aligned}$$

Cost

The number of inputs in the circuit is calculated by splitting the circuit in two parts. The inputs from outside who enter all gates, and the inputs of the second level gates who come from the first level gates. The $P = \sum_{k=0}^{i} p_k$ inputs from outside the circuit enter all gates. This gives rise to a number of inputs of:

$$\text{Cost resulting from outside inputs} = P \cdot (2^{q-1} + q - 2)$$

In addition to that, the second level gates also receive input from the first level. A second level gate is connected to all first level gates who define the upper boundaries of its ranges. Which is, for the saturated counters, equal to $2^{q-1-i} - 1$.

$$\begin{aligned} \text{Cost resulting from internal connections} &= \textstyle\sum_{i=0}^{q-2}(2^{q-i-1} - 1) \\ &= \textstyle\sum_{i=1}^{q-1}(2^{i} - 1) \\ &= \textstyle\sum_{i=1}^{q-1}(2^{i}) - q \\ &= \textstyle\sum_{i=0}^{q-1}(2^{i}) - 2^0 - q \\ &\Rightarrow 2^q - q - 1 \end{aligned}$$

Adding the two equations the desired result follows.

$$\text{Cost} = P(2^{q-1} + q - 2) + 2^q - q - 1$$

Delay

The delay of the Minnick method counters is always two, as this is a two layer method. In some cases, for example with carry propagation, it is worthwhile to know that the most significant bit needs only one calculation delay.

$$\begin{aligned} \text{Delay of } y_0, y_1, \cdots, y_{q-2} &= 2 \\ \text{Delay of } y_{q-1} &= 1 \end{aligned}$$

Fan-in $_{\max}$

The highest Fan-in occurs at the output gate of the least significant bit. The Fan-in consists of the P inputs from outside inputs and the $2q-1-1$ inputs from the first level of gates.

$$\text{Fan-in}_{\max} = P + 2^{q-1} - 1$$

Weight $_{\max}$

The maximum weight occurs at the first level in the gate which detects the highest range. We can determine this threshold by looking at the least significant output because it determines all first level devices. In the notation of Theorem 3.3 this is for saturated counters the $(S_{r-1}+1)^{+}_{=}$ gate. According to Theorem 2.1 with substituting the capacity c for p to incorporate higher order counters this is:

$$\begin{aligned} S_{r-1} \text{ of } y_0 &= 2^0 + (\tfrac{c+1}{2^1} - 1 - 1)2^1 \\ &= 1 + (\tfrac{c+1-4}{2})2 \\ &= 1 + c + 1 - 4 \\ &= c - 2 \end{aligned}$$

Adding one to the threshold because with Minnick the devices are $(S_{r-1}+1)^{+}_{=}$ we get:

$$\begin{aligned} \text{Weight}_{\max} &= c - 1 \\ &= 2^q - 2 \end{aligned}$$

3.3 The Kautz Method

In [Kau61], W.H. Kautz shows another method for creating counting functions. Even fewer devices are needed because he uses multiple levels, regrettedly getting an larger settling-time. One drawback of this method is that the ranges of the truth table are somewhat restricted, so its use for a general function is more difficult. However, it is especially well suited for the counters. That's why the first example is the $(7 \mid 3)$ counter, because the capabilities of the scheme become clear at once.

Consider the three-element network shown in Figure 3.8. The output of the three elements are shown plotted in Figure 3.9.

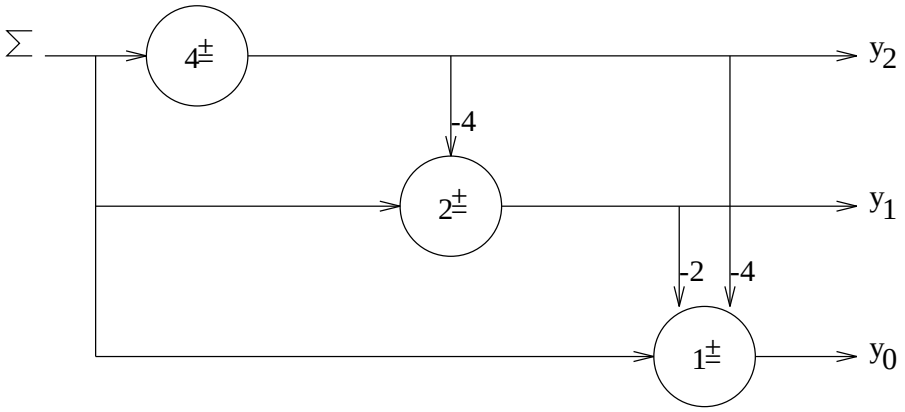

Figure 3.8: Circuit of the Kautz $(7 \mid 3)$ Counter

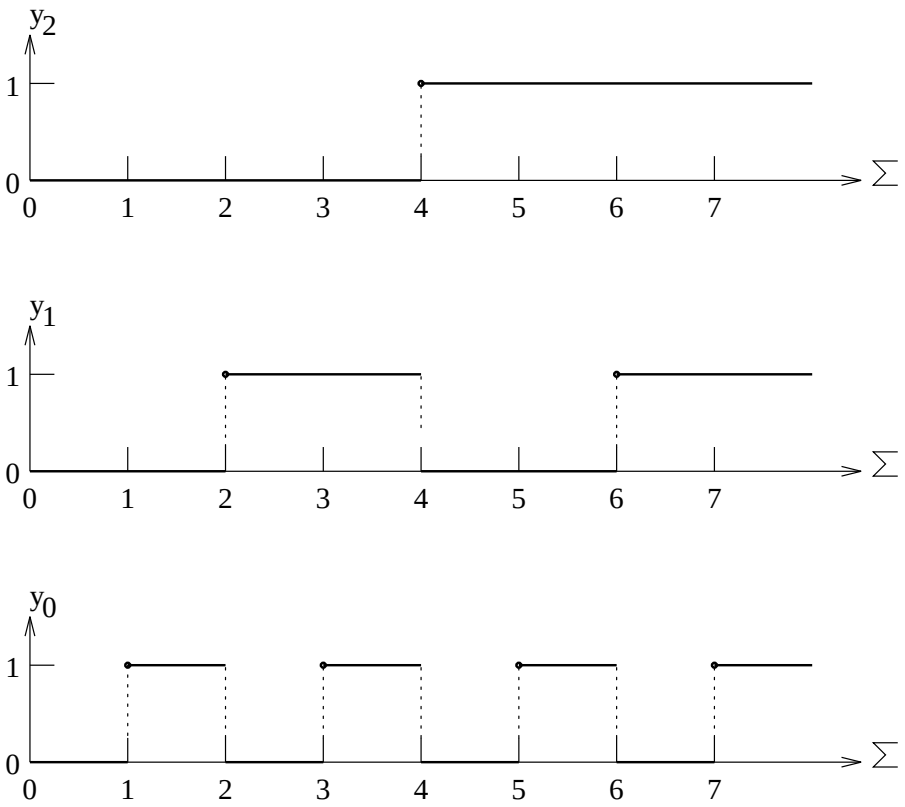

Figure 3.9: Element-Output Functions of the Kautz $(7 \mid 3)$ Counter

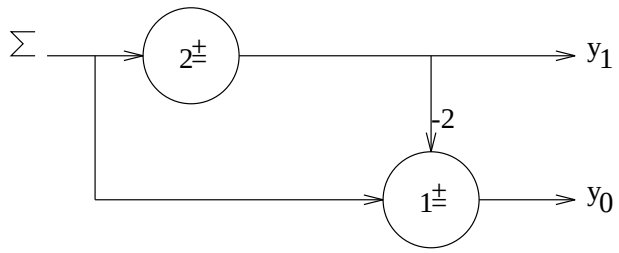

Figure 3.10: Circuit of the Kautz $(3 \mid 2)$ Counter

It operates in a rather straightforward way, a bit like the Minnick $(7 \mid 3)$-counter. The first element is a simple $4^{+}_{=}$, and so it generates y_2. The second element is simple $2^{+}_{=}$ as long as $v < 4$. When $v \geq 4$ however, y_2 becomes one and the apparent threshold of the second element increases to $2 + 6 = 6$. Thus y_1 has two positive transitions instead of one. With y_0 the number of positive transitions is again doubled. It has threshold 1 when $y_2 = y_1 = 0$, this threshold increases to $1 + 2 = 3$ when $y_2 = 0, y_1 = 1$. It increases again to $1 + 4 = 5$ when $y_2 = 1, y_1 = 0$; finally reaching $1 + 2 + 4 = 7$ then $y_2 = y_1 = 1$. Thus the circuit of Figure 3.8 realizes the wanted $(7 \mid 3)$ counter. And the beautiful thing is that you only design for the least significant bit, and the rest comes in for free.

Out of the last explanation it follows logically that this network may be expanded to get 8 positive transitions from the output of a fourth element, and so forth and so on.

3.3.1 A Kautz $(3 \mid 2)$ Counter

For reasons of completeness, Figure 3.10 shows the Kautz $(3 \mid 2)$ counter. This figure is exactly the same as the Minnick $(3 \mid 2)$ counter of Figure 3.6 on page 30.

3.3.2 A Kautz $(15 \mid 4)$ Counter

And now for the very last figure in our counters example cyclus: the Kautz $(15 \mid 4)$ counter of Figure 3.11.

3.3.3 The general Case of the Kautz' Counters

As this is a multi-layer method, we cannot give a simple formula as before. In [Kau61] a notation is suggested, which will provide valuable insight. Figure 3.12 shows how all thresholds and weights are called.

From [Kau61] we find Equation 3.4 that gives us the correct values to get to a counter.

$$T_{jk} = 2^{r-j}, j \leq k, j, k = 1, 2, \cdots, r. \tag{3.4}$$

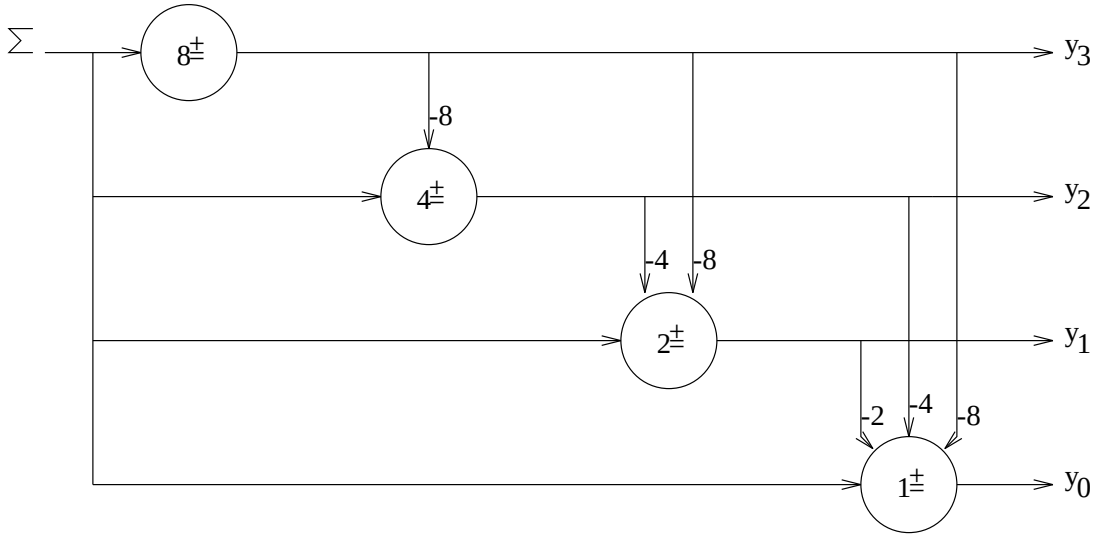

Figure 3.11: Circuit of the Kautz $(15 \mid 4)$ Counter

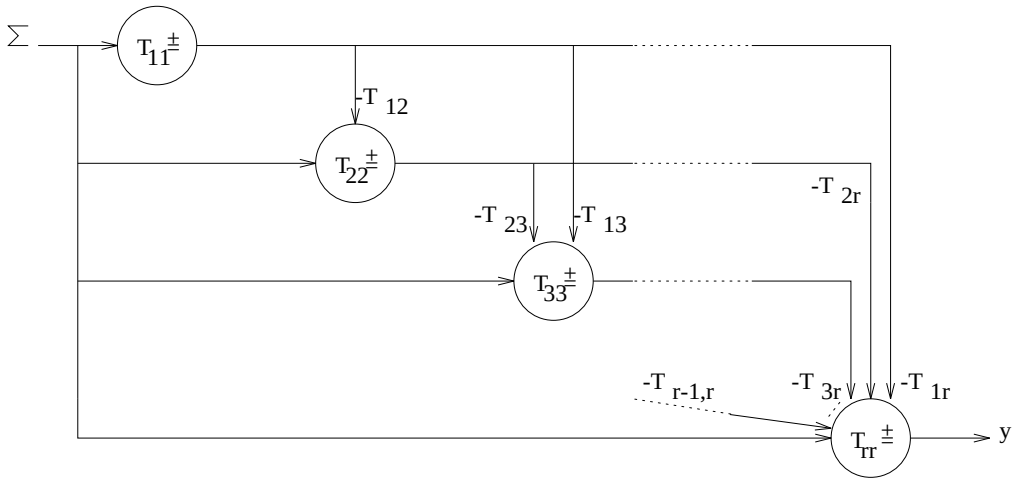

Figure 3.12: General Kautz-Network

3.3.4 Cost and delay of Kautz Counters

The cost of the Kautz counters is computed by the following.

Gates

With the Kautz method every output bit is connected to a gate. Thus the number of gates is equal to the number of output bits.

$$\text{Gates} = q$$

Cost

The number of inputs in the circuit is calculated by splitting the circuit in two parts. The inputs from outside who enter all gates, and the inputs of lower significant gates who come from higher significant gates. The $P = \sum_{k=0}^{i} p_k$ inputs from outside the circuit enter all gates. This gives rise to a number of inputs of:

$$\text{Cost resulting from outside inputs} = Pq$$

Looking again at Figure 3.12 we see that the outputs of the threshold devices connect to all lower significant devices, adding to Cost resulting from internal connections $= \sum_{i=1}^{q-1} i = \frac{1}{2}(q-1)q$ connections.

The sum of the two equations give for the total number of inputs:

$$\text{Cost} = Pq + \frac{1}{2}(q-1)q \tag{3.5}$$

Delay

The delay of a circuit is defined as the maximum number of times a signal has to pass a gate. When all gates are assumed to have the same input to output delay it is expressed in number of gate delays. With Kautz counters, compare Figure 3.12, the highest output bit is formed with only one gate, so it has a delay of one. The result of this is needed for the next output bit, it has to wait and then calculate so it has two, and so forth. The maximum delay occurs with y_0 and is equal to the number of output bits. In general the delay for the bit y_i is given by:

$$\text{Delay of } y_i = q - i \tag{3.6}$$

Fan-in $_{\text{max}}$

The highest Fan-in occurs at the output gate of the least significant bit. The Fan-in consists of the P inputs from outside inputs and the $q-1$ inputs from the other gates.

$$\text{Fan-in}_{\text{max}} = P + q - 1$$

Weight $_{\max}$

The threshold (which we also regard as a weight because it has to be implemented in likewise ways) of the least significant output gate is 2^0. This threshold doubles going to higher gates and amounts to 2^i. Thus the highest threshold is at the most significant output gate and is:

$$\text{Weight}_{\max} = 2^{q-1}$$

3.4 Cutting Unnecessary Lines

Some formulas change if we cut the unnecessary input lines. That is, not feeding input columns to lower order output bits. The basic method receives considerable reduction. Although it is not easy to capture this in simple formulas. This can be illustrated with a $(2,3 \mid 3)$ counter, see Figure 3.13.

This counter of-course can be build in the same way as a $(7 \mid 3)$ counter. So the circuit looks a lot like a $(7 \mid 3)$ counter. We notice first the reduced number of gates. The least significant input column v_0 counts up to and including 3, so there is no need to check for more. The v_1 is connected only to the outputs of equal and higher signification. Comparing to the standard $(7 \mid 3)$ counter this way saves both gates and input lines. If we had built an $(2,4 \mid 4)$ counter, the y_0 output would also need an $3^{-}_{=}$ which we could reuse from the y_1. So every case should be separately considered. For Minnick counters a similar argument can be made. Unfortunately, for Kautz' counters this optimization is not applicable. This is because in Kautz' counters, the result of higher order bits is used in calculating lower order bits.

3.5 Comparison

As can be seen in the foregoing discussions on the three design methods, they exhibit much differences in speed and cost. To make a clear comparison this section lists them in an orderly manner close to each other, together with some figures.

In Table 3.6 a clear summary of the equations derived in the previous sections is given. The Tables 3.7, 3.8 and 3.9 show the results of this equations for the five smallest saturated counters.

Based on this comparison, we see that the Kautz method is the most economical in terms of cost. Therefore we will continue to use this method in the chapters to follow as our main constrain of the investigation is to reduce the area of the network.

3.6 Conclusions

We have seen that parallel counters can be efficiently build using threshold logic. The Basic and Minnick method provide the output in two gate delays. The Kautz method is

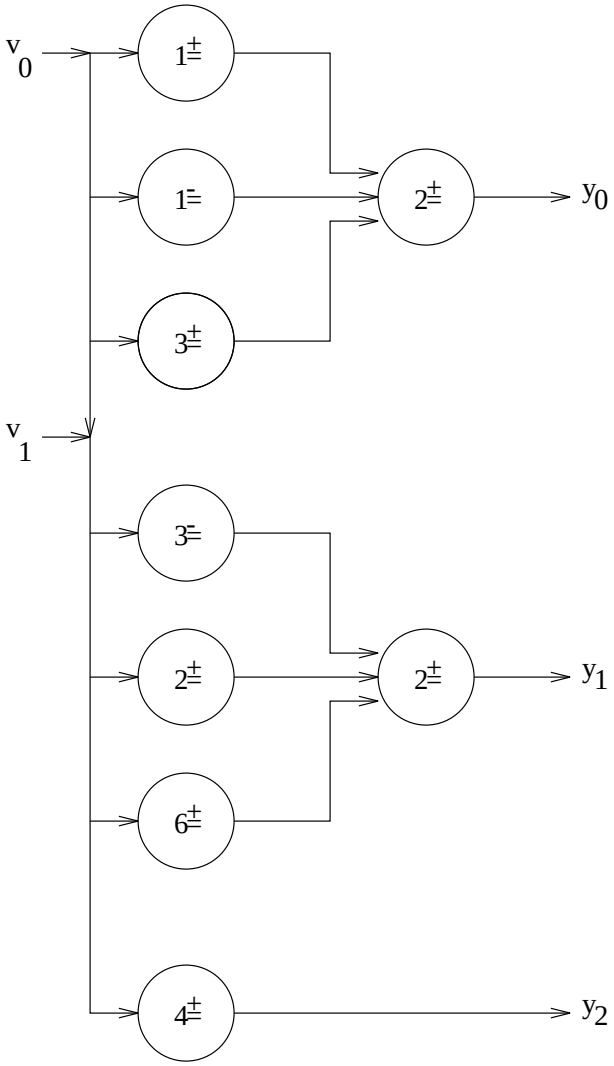

Figure 3.13: $(2,3 \mid 3)$ Counter with Reduced Input Lines

Design	Gates	Cost	D.	Fan-in $_{max}$	Weight $_{max}$
Basic	$q + 3(2^{q-1} - 1)$	$P(3 \cdot 2^{q-1} - 2) + 2^{q+1} - q - 3$	2	C	C
Minnick	$2^{q-1} + q - 2$	$P(2^{q-1} + q - 2) + 2^q - q - 1$	2	$P + 2^{q-1} - 1$	$2^q - 2$
Kautz	q	$Pq + \frac{1}{2}(q-1)q$	q	$P + (q-1)$	2^{q-1}

Table 3.6: Formulas for Comparison of the Different Design Methods

Gates	$(3 \mid 2)$	$(7 \mid 3)$	$(15 \mid 4)$	$(31 \mid 5)$	$(63 \mid 6)$
Basic	5	12	25	50	99
Minnick	2	5	10	19	36
Kautz	2	3	4	5	6

Table 3.7: Number of Gates Used by the Different Designs

Cost	$(3 \mid 2)$	$(7 \mid 3)$	$(15 \mid 4)$	$(31 \mid 5)$	$(63 \mid 6)$
Basic	15	80	355	1482	6041
Minnick	7	39	161	615	2325
Kautz	7	24	66	165	393

Table 3.8: Cost of the Different Designs

Delay	$(3 \mid 2)$	$(7 \mid 3)$	$(15 \mid 4)$	$(31 \mid 5)$	$(63 \mid 6)$
Basic	2	2	2	2	2
Minnick	2	2	2	2	2
Kautz	2	3	4	5	6

Table 3.9: Delay Required by the Different Designs

Fan-in $_{max}$	$(3 \mid 2)$	$(7 \mid 3)$	$(15 \mid 4)$	$(31 \mid 5)$	$(63 \mid 6)$
Basic	3	7	15	31	63
Minnick	4	10	22	46	94
Kautz	4	9	18	35	68

Table 3.10: Maximum Fan-in Required by the Different Designs

Weight $_{max}$	$(3 \mid 2)$	$(7 \mid 3)$	$(15 \mid 4)$	$(31 \mid 5)$	$(63 \mid 6)$
Basic	3	7	15	31	63
Minnick	2	6	14	30	62
Kautz	2	4	8	16	32

Table 3.11: Maximum Weight Required by the Different Designs

the slowest method providing the output one bit at a time, but is very cheap in gate count and connection density. An important conclusion in this chapter is that we establish that the Kautz circuits for symmetric functions can be used to build inexpensive counting devices.

Given that the primary concern of this thesis is the area we will use such counting devices as the basic blocks to produce higher level reductions.

The next chapter shows us how to combine the parallel counters to get bigger adders of larger input words.

Multiple Addition 4

Multiple addition is the adding together an amount of numbers. This can range from the addition of two numbers to the addition of a whole matrix of numbers. An efficient implementation of multiple addition is crucial in the design of multipliers and some array functions. The connection with multipliers will be elaborated in Chapter 5.

In essence multiple adders are parallel counters. In can be noted also that simple **0**-order parallel counters are in fact 1-bit multiple adders, as shown in Chapter 2. Even though counters can be viewed as multi-operand adders, we treat this adders separately from the counters because they are large thus we need to connect several parallel counters to form a multiple adder.

In this chapter we will show how to connect counters to form multiple adders. We primarily concern with the smallest area multi-operand adders. Thus we will assume that counters are designed using the Kautz method. We will use various order counters, and calculate the differences in the cost and delay parameters.

All formulas and numbers in this chapter are based on the following assumptions.

- We will be looking at $(2n-1) \times n$ matrices only. This is because this chapter is meant as a large introduction to the next chapter on multipliers. As will be shown in the next chapter, the multiplication process uses the reduction of a $(2n-1) \times n$ matrix. Using the same in here, we will be able to use formulas derived in this chapter for the next chapter.

- We will restrict ourselves to matrix sizes corresponding to the most used dimensions of operands, i.e. $n \in \{8, 16, 32, 64, 128\}$. This also deprives us from the

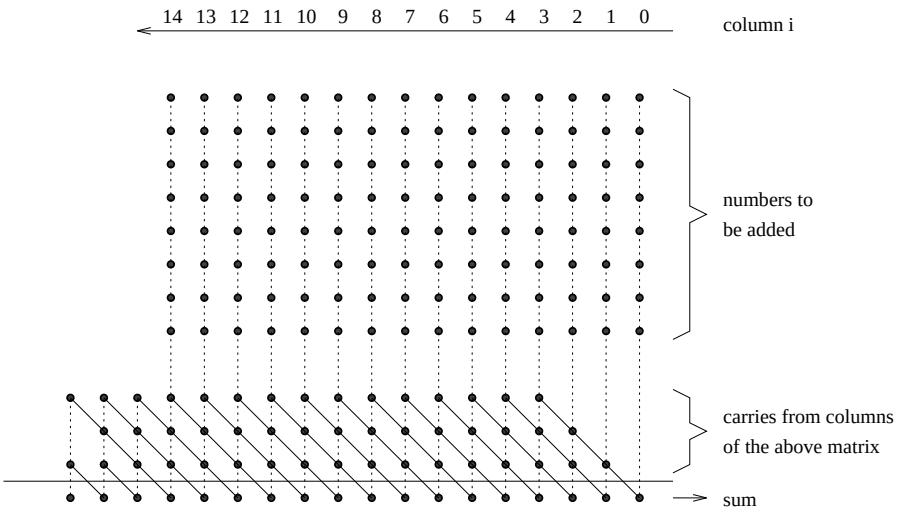

Figure 4.1: Addition of 8 15-bit Numbers with **0**-Order Counters. Dotted Lines Connect Inputs of a Counter, Full Lines the Outputs

need to use floor's and ceil's around the log's because the operand lengths are powers of two and thus give integer answers (we could alternatively state that $n \in \{2^x \mid x \in \{3, 4, 5, 6, 7\}\}$).

- All calculations will be based on Kautz counters, as we are trying to get the least expensive implementations.

4.1 Algorithm

In Figure 4.1, the addition of 8 15-bit numbers is accomplished by means of **0**-order counters. This means that every column of the matrix is input for a separate counter. We start on the right in the column $i = 0$. This column contains 8 bits, so we need 4 output bits (see Equation 2.3). The least significant output bit is part of the final sum, the others are carries to the columns 1 to 3, depending on their weight value.

In column $i = 1$ we now have 9 bits: eight of them belong to the original matrix, the ninth is a carry from column $i = 0$. These nine bits will be applied to a $(9 \mid 4)$ counter. Again, the least significant bit is part of the final sum, the others are recorded in following counters. From column $i = 3$ onwards $(11 \mid 4)$ counters are used. In columns $i = 15, 16, 17$ we see a "tail" which need calculation to. In this $n = 8$ case two additional $(3 \mid 2)$ and one $(2 \mid 1)$ counters are needed for this.

Another way of saying the above is that we have composed a BA(15,8,18) counter out of 15 $(11 \mid 4)$ and smaller counters and some extra. In Figure 4.2 this is made clear by drawing the stages underneath each other.

In the previous example we used **0**-order counters. We can vary the order of the counters, that is the number of input columns, to get different numbers on cost and

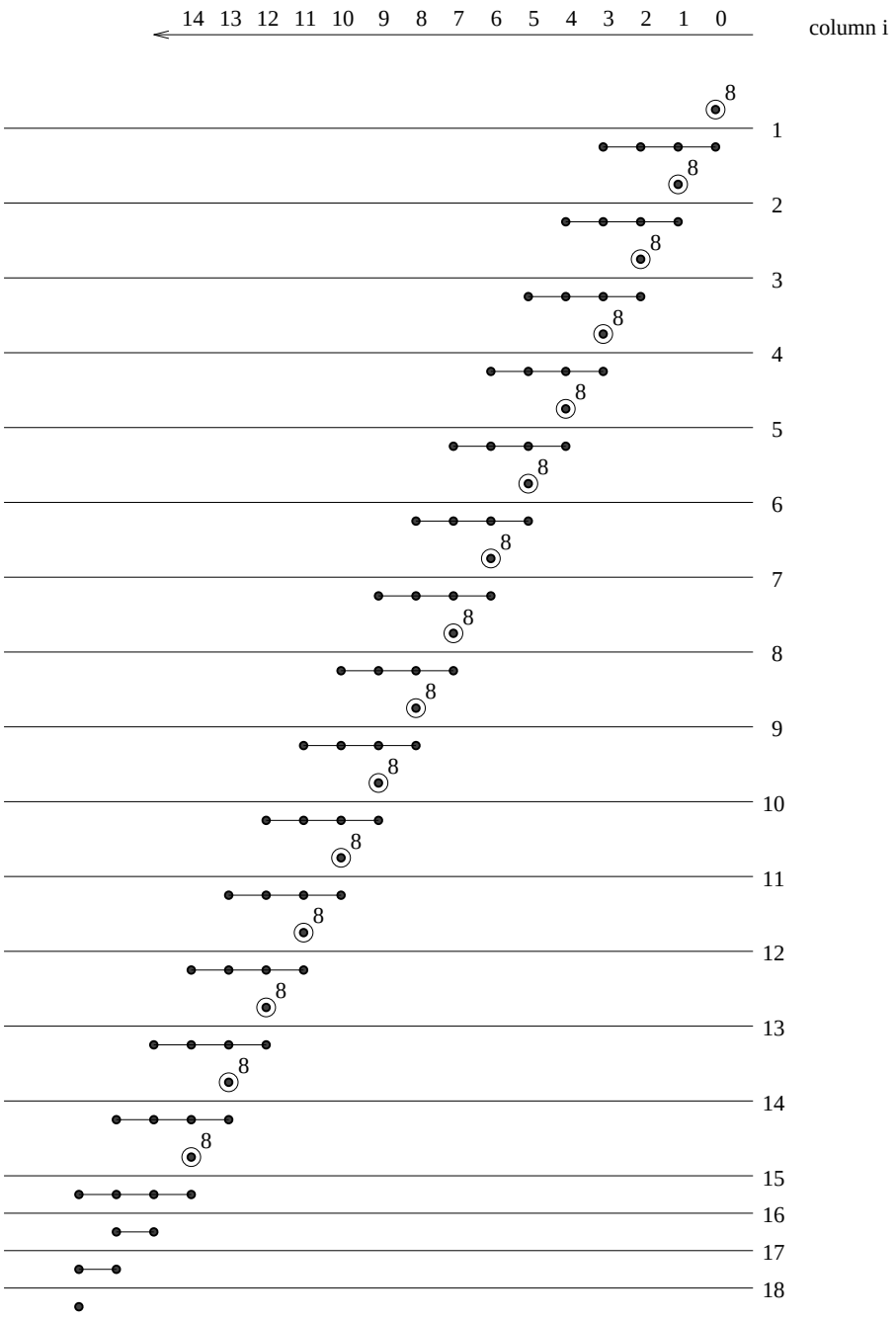

Figure 4.2: Addition of 8 15-bit Numbers with **0**-Order Counters. (The numbers next to the horizontal lines denote the stage in the adding process.)

performance.

4.2 Cost Calculation

For four distinct cases the formulas and results on the cost and performance parameters are given.

0-order addition

The tail in columns $i = 15, 16, 17$ comes from the fact that we consistently used **0**-order counters. We do not take this tail into the calculations because it would complicate the calculations unnecessarily and there is a clearer and less expensive way to do it, shown in Figure 4.3. The outputs of the column $i = 14$ counter are all used as part of the final sum, and the carry bits from columns $i = 12$ and $i = 13$ are fed directly (taking care of proper weights) into the column $i = 14$ counter. Because we base the calculations on the biggest counter used, in columns $i = 0, 1, 2$ we overprice a little which makes up for the extra lines to column $i = 14$ thus out formulas will still be right.

There are n numbers to be added, divided into $2n - 1$ columns. These n numbers enter counters with 1 sum bit and $\log n$ carries. The least significant output bit is part of the final sum, the others are recorded in other columns. Thus, the maximum height of the columns now is $n + \log n$ bits. Now we should calculate again the number of output bits and the matrix height and so forth. Because our choices for n leave the counters with an extreme over capacity of $n - 1$ we will not need extra output bits. We will also give a formal proof of this fact in Theorem 4.1.

If we take all counters the maximum height, this gives us $2n - 1$ counters with $n + \log n$ inputs and $1 + \log n$ outputs. This assumption will give us an upper bound for the formulas to follow. A Kautz counter has the maximum fan-in at its least significant output bit, consisting of the $n + \log n$ inputs, plus the threshold influencing connections coming from higher order gates of the same counter (see section 3.3.4):

$$\text{Fan-in } \leq n + 2 \cdot \log n \tag{4.1}$$

Using Equation 3.3.4, the total number of gates used for the counters in columns 0 to $2n - 2$ follows easily.

$$\text{Number of gates} \leq (2n - 1) \cdot (1 + \log n) \tag{4.2}$$

With the aid of Equation 3.5 we calculate the cost of one counter as $(n + \log n) \cdot (1 + \log n) + \frac{1}{2}(1 + \log n) \log n$. Simplified and multiplied by the number of counters this gives:

$$\text{Cost} \leq (2n - 1) \cdot (n + \frac{3}{2} \log n) \cdot (1 + \log n) \tag{4.3}$$

The delay of a Kautz counter is different for each output. As we have shown in Equation 3.6 is the most significant output bit calculated first, and each lower bit takes one gate delay more than its predecessor. Thus, a counter has to wait for all outputs of

its predecessor, except for the least significant bit which is output directly. This gives us $\log n$ gate delay per counter. At the very end, we have to add one gate delay for we must wait for all bits to get to the final sum.

$$\text{Gate delay} \leq (2n-1) \cdot \log n + 1 \tag{4.4}$$

Looking at a single counter, the maximum weight is the threshold of the most significant output bit. This threshold depends only on the number of output bits, being 2^q. As stated earlier in this section, the number of output bits will not grow because of the incoming carries and stay $1 + \log n$. This is the subject of the theorem to follow.

Theorem 4.1 *The number of output bits of a counter with $n + \log n$ inputs is not more than the number of output bits of a counter with n inputs, with $n \in \{2^x \mid x \in N\}$. That number of output bits is $1 + \log n$.*

Proof: The number of output bits of a $n + logn$ input counter is, using the second variant of Equation 2.3,

$$\begin{aligned} \lceil \log(n + \log(n) + 1) \rceil &\leq \lceil \log(n + n - 1 + 1) \rceil \\ &= \lceil \log(2n) \rceil = \lceil \log(n) + \log(2) \rceil \\ &= \lceil \log(n) + 1 \rceil = 1 + logn \end{aligned}$$

In which we use the fact that $\log a \leq a - 1$ and that $n \in \{2^x \mid x \in N\}$.

□

As a conclusion to the theorem above, the most significant output bit has a threshold of $2^{\log n}$ which simplifies to:

$$\text{Weight}_{\text{max}} = n \tag{4.5}$$

Table 4.1 indicates the values for these five parameters for the most used dimensions of n.

As already mentioned, because we use only $n \in \{2^x \mid x \in N\}$ the counters have an extreme over capacity of $n - 1$. So it promises worthwhile to try and saturate the counters as much as possible. Thus we will enlarge the input count from $n + \log n$ to the maximum possible with $1 + \log n$ outputs, which amounts to $2^{1+\log n} - 1 = 2n - 1$ Figure 4.3 shows the result of this approach. Column $i = 0$ contains n numbers. We can redirect input from column $i = 1$ to column $i = 0$ and note that we replicate that redirected input twice to create double the weight. For the other columns we follow the same process. In the example of Figure 4.3, starting in column $i = 3$ the counters stabilize to have 7 inputs of their original column and 4 from the column next to it.

What is implied is the fact that the number of gates stays the same. So this won't be the big saving. The delay also stays approximately the same. Also, the cost does not decrease, because an $(15 \mid 4)$ counter costs more than an $(11 \mid 4)$ counter. We will not investigate this further. Instead of this we will try to use counters which have an input width of 2, the **1**-order counters.

n	Cost	Gates	Fan-in $_{max}$	Delay	Weight $_{max}$
8	750	60	14	46	8
16	3410	155	24	125	16
32	14931	378	42	316	32
64	64897	889	76	763	64
128	282540	2040	142	1786	128

Table 4.1: Cost Parameters for Multiple Adders, Using Kautz **0**-Order Counters. This Table Corresponds to Figure 4.1

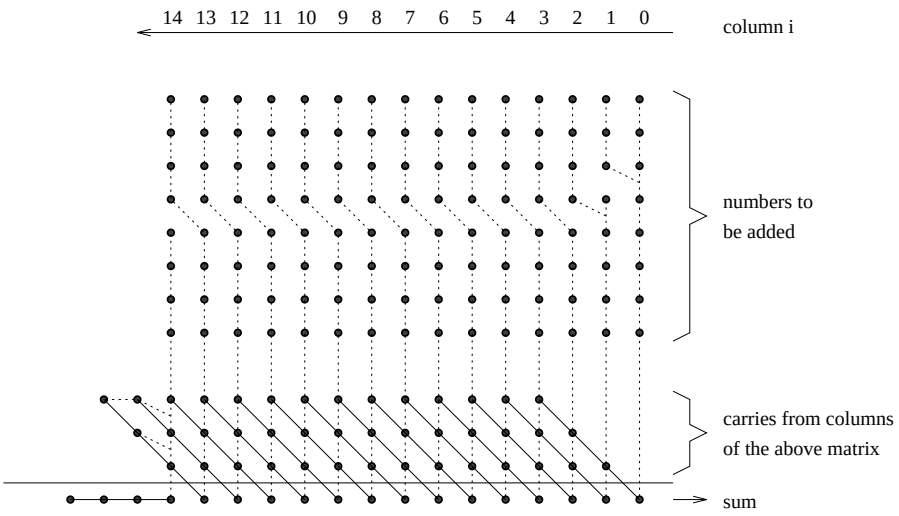

Figure 4.3: Addition of 8 15-bit Numbers with Saturated **0**-Order Counters.

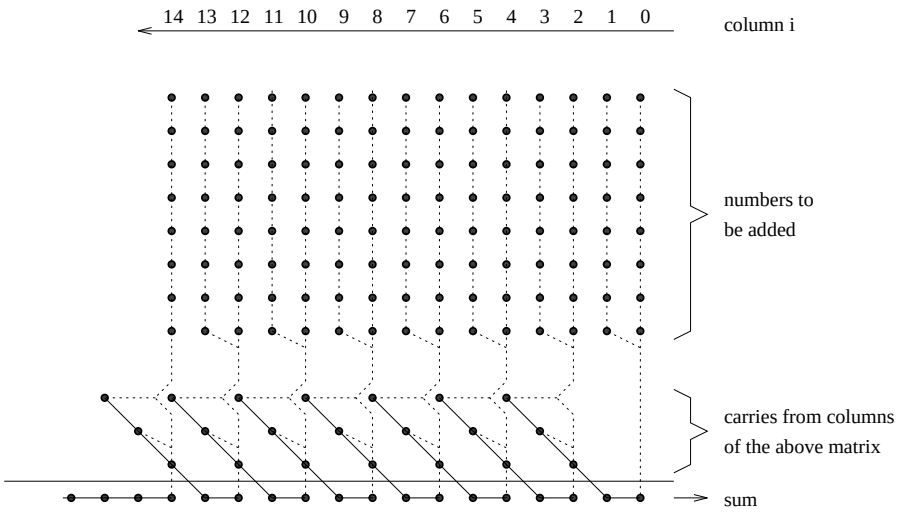

Figure 4.4: Addition of 8 15-bit Numbers with **1**-Order Counters.

n	Cost	Gates	Fan-in $_{max}$	Delay	Weight $_{max}$
8	840	40	23	26	16
16	3696	96	41	66	32
32	16128	224	75	162	64
64	70400	512	141	386	128
128	307584	1152	271	898	256

Table 4.2: Cost Parameters for Multiple Adders, using Kautz **1**-Order Counters. This Table Corresponds to Figure 4.4

1-order addition

Figure 4.4 shows the approach if we take two columns at a time. Also we have made sure that all outputs of a certain column enter the counter direct next to it to facilitate a more clear explanation of the calculations.

Each of the counters (with the exception of the leftmost, which is smaller in most cases, hence all the less or equal signs) has $p = 2n$ inputs if not counting the carries. The maximum input count per counter if not counting the carries is $c = 2^1 n + 2^0 n = 3n$ which calls for $q = 1 + \lfloor \log 3n \rfloor = 1 + \lfloor \log 3 + \log n \rfloor = 2 + \log n$ output bits. Of the $2 + \log n$ output bits, $\log n$ are carried over to the counter directly to the left. Now the number of inputs per counter rise to $p \leq 2n + \log n$. The maximum input count raises with the maximum amount the $\log n$ carry bits can represent which is $2^{\log n} - 1$, to $c \leq 3n + 2^{\log n} - 1 = 4n - 1$. Now we need $\lceil \log(4n - 1 + 1) \rceil = \lceil \log 4n \rceil = \lceil \log 4 + \log n \rceil = 2 + \log n$ output bits. Thus the carries do not call for extra output bits.

If we take all counters to be equal to the biggest available, we have $p \leq 2n + \log n$

inputs per counter. The least significant bit of a counter receives $1 + \log n$ inputs from other gates within the counter. The sum of these two values gives the maximum fan-in:

$$\text{Fan-in } \leq 2n + 2\log(n) + 1 \tag{4.6}$$

To continue we first calculate the number of counters:

$$\text{Number of counters} = \left\lceil \frac{2n-1}{2} \right\rceil = \left\lceil n - \frac{1}{2} \right\rceil = n$$

With n counters and $2 + \log n$ output bits per counter, the Kautz method gives:

$$\text{Gates} \leq n \cdot (2 + \log n) \tag{4.7}$$

The cost consists of the added cost of n counters. The counters each have $2n + \log n$ inputs. The internal connections of these counters with $2 + \log n$ output bits amount to $\frac{1}{2}(2 + \log n)(1 + \log n)$. This leads to:

$$\text{Cost} \leq n \cdot \left((2n + \log n)(2 + \log n) + \frac{1}{2}(2 + \log n)(1 + \log n) \right) \tag{4.8}$$

Each counter has to wait for $\log n$ output bits from its neighbor, and in the end an extra delay of 2 is needed to complete the last counter:

$$\text{Gate delay} \leq n\log(n) + 2 \tag{4.9}$$

The maximum weight is $2^{1+\log n}$ which simplifies to:

$$\text{Weight}_{\text{max}} = 2n \tag{4.10}$$

Table 4.2 indicates the values for these five parameters for a few sizes of n.

Logarithmic Order Addition

Figure 4.5 shows the approach if we take $\log n$ columns at a time. Each of the counters has $p = n \log n$ inputs not counting the carries. The input count per counter, not counting the carries is $c = n(2^{\log n} - 1) = n(n-1) \leq n^2$ which calls for $q = \lceil \log(n(2^{\log n} - 1) + 1 \rceil = \lceil \log(n(n-1) + 1) \rceil = \lceil \log(n^2 - (n-1) \rceil \leq \lceil \log n^2 \rceil = \lceil 2 \log n \rceil = 2 \log n$ output bits. Thus the carries of a counter only influence the counter direct to the left, and it is with an extra row of inputs. To make sure we still make it with the number of output bits: $c = (n+1)(2^{\log n} - 1) = (n+1)(n-1) = n^2 - 1$. For this we need $q = \lceil \log(n^2 - 1 + 1) \rceil = \lceil \log n^2 \rceil = 2 \log n$ output bits, thus this stays the same.

The algorithm is shown in some more detail in Figure 4.6. The formulas below follow in the same way as in the foregoing.

If we take all counters to be equal to the biggest available, we have $(n+1)\log n$ inputs per counter. The least significant bit of a counter receives $2\log(n) - 1$ inputs

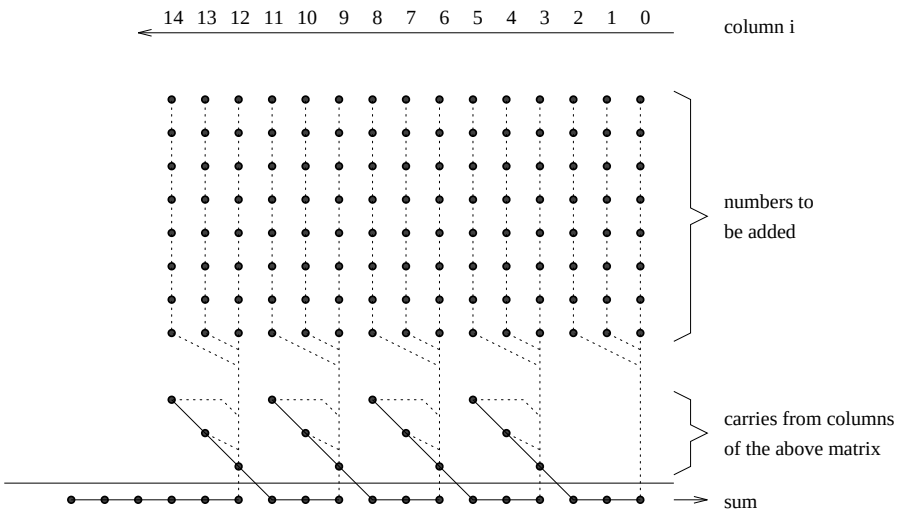

Figure 4.5: Addition of 8 15-bit Numbers with **Logarithmic**-Order Counters.

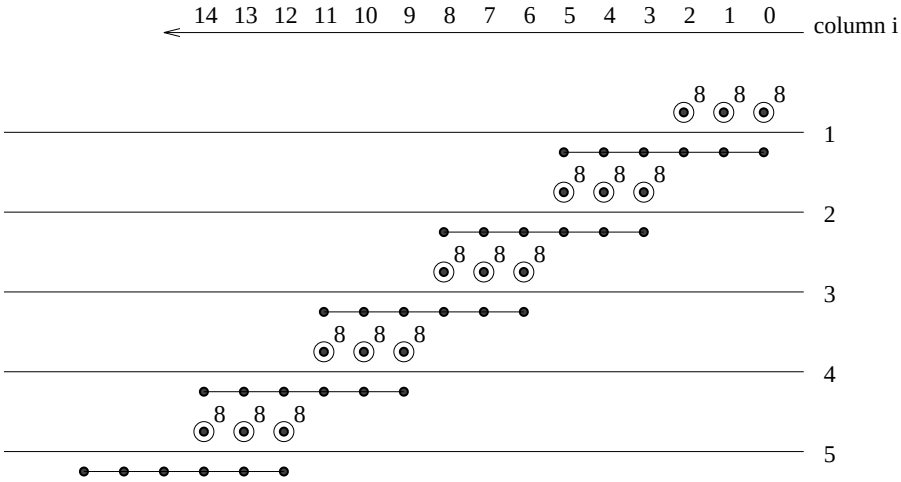

Figure 4.6: Addition of 8 15-bit Numbers with **Logarithmic**-Order Counters. (The numbers next to the horizontal lines denote the stage in the adding process.)

n	Cost	Gates	Fan-in $_{max}$	Delay	Weight $_{max}$
8	1062	36	32	21	32
16	5005	70	75	39	128
32	23052	136	174	73	512
64	105203	266	401	139	2048
128	476578	524	916	269	8192

Table 4.3: Cost Parameters for Multiple Adders, using Kautz **Logarithmic**-Order Counters. This Table Corresponds to Figure 4.5

from other gates within the counter. The sum of these two values gives the maximum fan-in:

$$\begin{aligned} \text{Fan-in}_{\max} &\leq (n+1)\log n + 2\log(n) - 1 \\ &= (n+3)\log n - 1 \end{aligned}$$

We calculate the number of counters:

$$\begin{aligned} \text{Number of counters} &= \left\lceil \frac{2n-1}{\log n} \right\rceil \\ &< \left(\frac{2n-1}{\log n} + 1 \right) \end{aligned}$$

Per counter we have $2\log n$ gates. Multiplied by the number of counters we have:

$$\begin{aligned} \text{Number of gates} &\leq \left\lceil \frac{2n-1}{\log n} \right\rceil \cdot 2\log n \\ &< \left(\frac{2n-1}{\log n} + 1 \right) \cdot 2\log n \\ &= \left(\frac{2n-1+\log n}{\log n} \right) \cdot 2\log n \\ &= 4n + 2\log n - 2 \end{aligned}$$

The cost per counter calculates as the number of inputs times the number of output gates plus the number of internal connections:

$$\begin{aligned} \text{Cost} &\leq \left\lceil \frac{2n-1}{\log n} \right\rceil \cdot \left((n+1)\log(n) 2\log(n) + \frac{1}{2} 2\log(n)(2\log n - 1) \right) \\ &< \left(\frac{2n-1}{\log n} + 1 \right) \cdot \left(2(n+1)\log^2 n + \log n(2\log n - 1) \right) \\ &= \left(\frac{2n-1+\log n}{\log n} \right) \cdot \left(2(n+1)\log^2 n + \log n(2\log n - 1) \right) \\ &= (2n-1+\log n) \cdot (2(n+1)\log(n) + 2\log(n) - 1) \\ &= (2n-1+\log n) \cdot (2n\log(n) + 4\log(n) - 1) \end{aligned}$$

Each counter has to wait for $\log n$ output bits from its neighbor, and in the end an extra delay of $\log n$ is needed to complete the last counter:

$$\begin{aligned} \text{Delay} &\leq \left\lceil \frac{2n-1}{\log n} \right\rceil \cdot \log n + \log n \\ &< \left(\frac{2n-1}{\log n} + 1 \right) \cdot \log n + \log n \end{aligned}$$

$$= \left(\frac{2n - 1 + \log n}{\log n}\right) \cdot \log n + \log n$$
$$= 2n - 1 + 2\log n$$

The maximum weight is:

$$\begin{aligned} \text{Weight}_{\text{max}} &= 2^{2\log n - 1} \\ &= \frac{1}{2}(2^{\log n})^2 \\ &= \frac{1}{2}n^2 \end{aligned}$$

Table 4.3 indicates the values of these four parameters for a few sizes of n.

4.3 Conclusions

In this chapter with the major concern being the smallest possible area constrained with all other parameters related, we investigated multi-operand addition. We have shown that using Kautz counters very small area is required for the design of large multi-operand additions when the other parameters are left unrestricted. That is we have established somehow a lower bound for an implementation using Kautz counters.

In the chapter to follow we will investigate, with the same assumptions, multiplication. We will assume that the multiplication is created using both signed and unsigned numbers.

5 Multipliers

In this chapter we investigate the use of threshold gates with the multiplication of binary signed and unsigned numbers. The major concern again for the chapter is minimal area with the other parameters unrestricted. First the general binary multiplication process will be explained. Consequently we will discuss in general an algorithm that eliminates negative elements in the multiplication matrix. The suggestion being that treating unsigned multiplication is enough to also treat most multiplier designs. Finally, various fast parallel and cheap serial-parallel algorithms are described and compared on the subject of circuit cost and calculation time.

5.1 Multiplication Algorithm

Calculating the product of two binary numbers consists of two separate steps. First temporary results are formed by multiplying every single bit of the multiplier *A* with the multiplicand *B*. Next these temporary results are added to form the product. The temporary results are called *partial products*. They are grouped down in a matrix, the *partial product matrix*. See Figure 5.1 for an example multiplication of $1001_{bin} \cdot 1010_{bin}$ which is the binary equivalent of the decimal $9 \cdot 10$.

Formally, let *A* be a n bit multiplier and *B* the m bit multiplicand. A and B are non-negative integer numbers. The product $A \cdot B$ is the $n + m$ bit weighted sum of all partial products $a_i b_j$ (see Figure 5.2):

$$A = a_{n-1}a_{n-2}\cdots a_1 a_0 \qquad (a_i \in \{0,1\}) \tag{5.1}$$

$$
\begin{array}{cccccccc}
 & & & & 1 & 0 & 1 & 0 & \\
 & & & & 1 & 0 & 0 & 1 & \times \\
\hline
 & & & & 1 & 0 & 1 & 0 & \\
 & & & 0 & 0 & 0 & 0 & & \\
 & & 0 & 0 & 0 & 0 & & & \\
 & 1 & 0 & 1 & 0 & & & & + \\
\hline
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 &
\end{array}
$$

Figure 5.1: Example of Binary Multiplication

$$
\begin{array}{cccccccc}
 & & & & b_3 & b_2 & b_1 & b_0 & \\
 & & & & a_3 & a_2 & a_1 & a_0 & \times \\
\hline
 & & & & a_0b_3 & a_0b_2 & a_0b_1 & a_0b_0 & \\
 & & & a_1b_3 & a_1b_2 & a_1b_1 & a_1b_0 & & \\
 & & a_2b_3 & a_2b_2 & a_2b_1 & a_2b_0 & & & \\
 & a_3b_3 & a_3b_2 & a_3b_1 & a_3b_0 & & & & + \\
\hline
\sigma_7 & \sigma_6 & \sigma_5 & \sigma_4 & \sigma_3 & \sigma 2 & \sigma_1 & \sigma_0 &
\end{array}
$$

Figure 5.2: General Binary Multiplication

$$
\begin{aligned}
 & = \sum_{i=0}^{n-1} a_i \cdot 2^i \\
\\
B &= b_{m-1}b_{m-2}\cdots b_1b_0 \qquad (b_j \in \{0,1\}) \qquad (5.2) \\
 &= \sum_{j=0}^{m-1} b_j \cdot 2^j \\
\\
A \cdot B &= \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} a_ib_j \cdot 2^{i+j} \qquad (5.3)
\end{aligned}
$$

In straightforward iterative multiplication a partial product is calculated and directly added to an accumulator register. This takes a long time largely because of the carries that arise as the multiplier adds the successive partial products. Various techniques exist to speed up this process. In this chapter we create the entire partial product matrix at the same time and add ALL partial products at the same time on a column-by-column basis. This is called bit-oriented addition or column-oriented addition. To attempt to reduce the area we also consider on serial-parallel multipliers which differ in that the columns are not added all at the same time but using some iterative scheme one after the other.

For ease of notation and compactness, the Dadda-like dot notation is used throughout this chapter. For example in Figure 5.3 diagrammatically the multiplication of two unsigned 16-bit numbers is shown. Of this only the significant bits are denoted, the

zeros to the left and right of the matrix are not shown.

Given that the most frequently used multipliers are the $8 \times 8, 16 \times 16, 32 \times 32$ and 64×64 and because they are all powers of 2 they can be expressed as $2^x \times 2^x, x \in \{3, 4, 5, 6\}$ or $x \in N$ in general. In this chapter they are called the *standard multipliers*. We will emphasize on them because of the powers of 2 the forthcoming formulas who depend heavily on base 2 logarithms appear to be much simpler than in the general case.

5.2 Unsigned versus Signed Multiplications

Signed operands involve potentially sign bits that may be negative. Also, the most frequently used multiplication algorithms, the multi-bit overlapped scanning algorithm, may introduce negative elements, dots in our representation, in the multiplication matrix.

While threshold logic can be used to accommodate element addition/subtraction, there are algorithms that entirely avoid the necessity. We discuss brief such algorithms for multi-bit overlapped scanning. The discussion is from [VSH89, VSS91].

5.3 The algorithms

First we note that sign magnitude numbers, use of sign operand representations, can be simply viewed as two's complement numbers, the other most frequently used signed notation. Simply stated two sign magnitude numbers, X and Y can be written as:

$$Y = -y_0 + \sum_{i=1}^{n-1} y_i 2^{-i}, \qquad X = -x_0 + \sum_{i=1}^{q-1} x_i 2^{-i}$$

and the product is:

$$P = X \cdot Y = -p_0 + \sum_{i=1}^{W_p - 1} p_i 2^{-i} \qquad \text{with } x_0 = y_0 = p_0 = 0$$

In an s-bit overlapped scanning algorithm, the multiplication in general can be described by the following. (The discussion is from [VSH89].)

The multiplier is divided into s-bit contiguous groups. Except for the group containing the most significant bit of the multiplier, the most significant bit of each group is the same as the least significant bit of the previous group. After the division of the multiplier into the previously described groups, each group is scanned once implying that the least significant bit of each group except the group containing the least significant bit of the multiplier is scanned twice. The scanning process can be started at either end of the multiplier. In correspondence to every group scanned, depending on the bit pattern and the position of the group, a row involving the multiplicand X is created for the multiplication matrix which is referred to as a partial product (PP). Finally, the matrix containing all the partial products is added with a convenient scheme to produce the final

product. In this section, the previously described multiplication process and the actions to be followed for the design of hardware multipliers is described.

Assume the multiplication of two fractional numbers in two's complement notation with s-bit overlapped scanning such that $n > s$ where n is the length of the multiplier. Let m be equal to $\lfloor (n-1)/(s-1) \rfloor$ which defines the number of groups of the multiplier to be equal to $m+1$ The implication of grouping the multiplier in $m+1$ groups is the following.

1) There will be at least one zero at the least significant end. This is because either $r = 0$, which implies $s-1$ zeros added, or if $r > 0$, at least one zero will be added because r can be at maximum $s-2$.

2) There will be $m+1$ partial product terms. The ith partial product has a coefficient $W_{(i)}$ as follows:

$$W_{(i)} = -Y_{(s-1)(i-1)}2^{(s-2)} + Y_{(s-1)i} + \sum_{j=1}^{s-2} Y_{[(s-1)(i-1)+j]}2^{-[j+1-(s-1)]}$$

Then,

$$\chi * \mathbf{\Psi} = \sum_{i=1}^{m+1} XW_{(i)}2^{-[i(s-1)-1]}$$

Where $W_{(i)}$ is the coefficient of the ith scan, $XW_{(i)}$ is the multiple of ith scan, and $XW_{(i)}2^{-[i(s-1)-1]}$ is the partial product of the ith scan.

Given that $W_{(i)}$ can have a negative value negative elements may be introduced to the multiplication matrix. An algorithm has been proposed in [VSS91] that excludes the negative elements. The algorithm operates as follows [VSS91]:

Assume an S-bit overlapped scanning and let the width of X and Y be, respectively, q and n including the sign bit forced to zero, then:

1. Determine the number of possible significant bits in a multiple of each partial product and the number of the partial product terms to be respectively $q-1+S-2$ and $M+1$ with $M = \left\lfloor \frac{n-1}{S-1} \right\rfloor$.

2. Place the partial product terms into a matrix by shifting the $(i+1)$th partial product $S-1$ bits to the right of the ith partial product.

3. Allow for $S-1$ significant bits of encode to the right of every term except the last one (this is because the encode for this term is always zeros since these bits are based on the sign of the next partial product term which there is none). Allow for $S-1$ significant bits to the left of every term except the first one (which needs S bits) for the encoding of the sign extensions.

4. Place 1's and 0's in the bit positions where they are always given into the matrix. The encoding to the right of the multiple has $S-2$ zeros to the most significant portion of this encode. Also, the encoding to the left has $S-2$ ones in the most significant portion of this encode.

For more details and proof, the interested reader is referred to [VSS91].

Many computer architectures, e.g., S/370 [IBMb] and S/370 XA [IBMa], define multiplication of floating-point numbers in sign-magnitude fractional notation and of fixed-point numbers in two's complement non fractional notation. Since both types of multiplication are architecturally defined, it is required to design a computer system that will accommodate both notations. While there are a number of algorithms to achieve both multiplications, see for example [GH86], they usually require extra delay and/or hardware. The algorithm presented previously [VSS91] can be modified to accommodate both notations. First of all it is noted that the integer number (two's complement notations usually involve integer numbers) can be written as follows:

Given that a two's complement integer number Z of length n is equal to

$$\begin{aligned} Z &= -Z_{n-1}2^{n-1} + \textstyle\sum_{i=0}^{n-2} Z_i 2^i \\ &= -Z_{n-1}2^{n-1} + \textstyle\sum_{i=1}^{n-1} Z_{n-1-i} 2^{n-1-i} \\ &= 2^{n-1}\left[-Z_{n-1} + \textstyle\sum_{i=1}^{n-1} Z_{n-i-1} 2^{-i}\right] \end{aligned}$$

It can be stated that an integer two's complement number can be viewed as a fractional number with appropriate radix point shifting and renaming of the bit positions. In essence it can be stated that integer and fractional number conventions are essentially equivalent and easily converted one to the other.

The previous discussion indicates that the multiplication algorithm for sign-magnitude numbers can be easily adapted to compute two's complement multiplications with three minor modifications (see for more details notation etcetera [VSS91]), namely:

1. The one's complementation and "hot 1" addition for the required two's complement subtraction should reflect a negative coefficient rather that a negative partial product.

2. Sign extend the multiplier, Y, by at least $S-1$ bits and to the length n of the sign-magnitude notation.

3. Adjust the control of the left sign encode th be based off $A_i = \Gamma_i \oplus X_0$ rather than just $A_i = \Gamma_i$, Γ_i being the sign of the ith coefficient W_i.

The consequence of the previous discussion is that treating unsigned numbers will also treat signed numbers using the most commonly used notations and algorithms with no restriction for the threshold gates.

5.4 Parallel Multipliers

In this section we assume unsigned notations and accomplish the second stage in the multiplication process, the addition of the partial product matrix, with threshold gates. In essence the addition of the partial product matrix is just a multiple addition like in

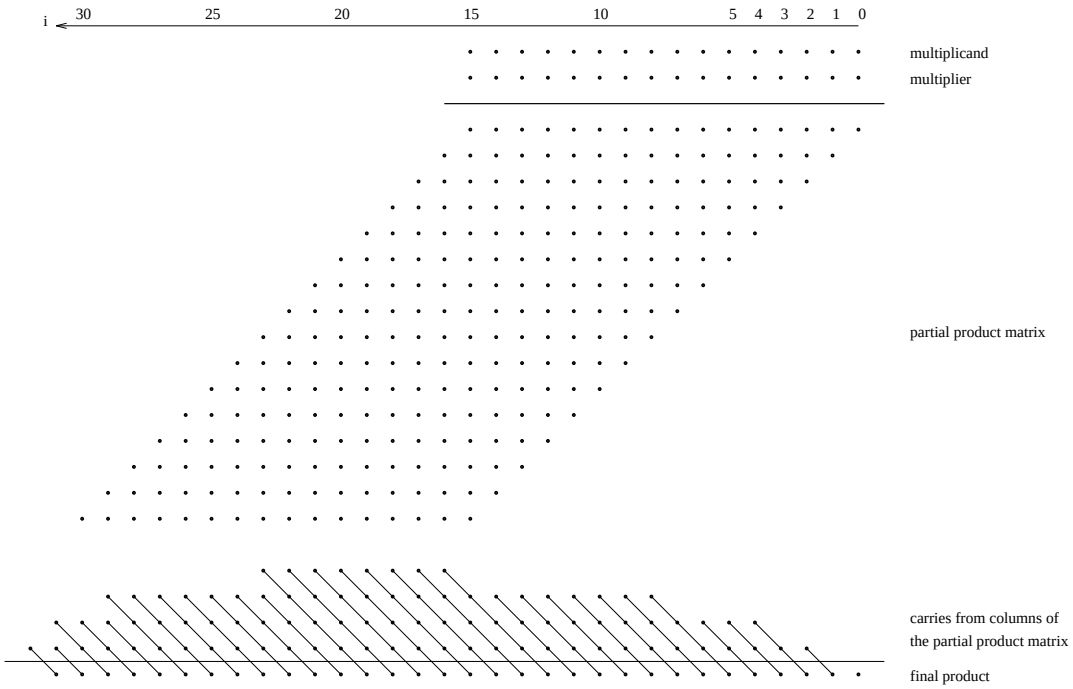

Figure 5.3: 16x16 bit multiplication in a single stage using parallel counters. Carries are propagated through the counters

Chapter 4. The difference lies in the fact that almost half of the matrix is always equal to zero due to the nature of the multiplication process.

It should also become clear why we used the addition of n numbers of $(2n-1)$ bits in Chapter 4. The partial product matrix of an $n \times n$ multiplier precisely fits in such an adder. The difference is that in an multiplier, about half of the matrix is always zero, so we can cut down on cost. An upper bound on cost and delay is thus given by the same formulas as in Chapter 4.

We will not consider the forming of the partial product matrix. As described, this can be accomplished with n^2 two input gates thus with a cost of $2n^2$. It will take a delay of one. This numbers are not incorporated in the various tables in this chapter.

5.4.1 Cost Calculation

Like we did with the Multiple Adders, we will take a close look at reduction schemes with various order counters.

0-order reduction

In Figure 5.3 is depicted the reduction of an 16x16 bit multiplier with **0**-order counters. The delay and cost parameters of this multiplier appear, as we will see, not direct

calculatable in an exact way. Using an algorithm we can obtain the exact answers. We are concerned only about the cost of the reduction of the partial product matrix, so the hardware to create the partial products is not taken into account in the calculations.

As already said, we have no exact expression on cost and delay. When we don't want to make use of the algorithm we can use the upper bound formulas from Chapter 4. The derivation of the algorithm to calculate the cost and delay parameters is given below.

We start by expressing the height of the columns in some way. The height of the partial product part of the matrix is given by:

$$h_i = \begin{cases} i+1, & \text{for } i = 0, \cdots, n-1 \\ 2n-1-i, & \text{for } i = n, \cdots, 2n-2 \end{cases} \tag{5.4}$$

To get the height of the carry part of the matrix, we start of by noting the number of output bits q_i of the i'th counter c_i. It goes with the aid of equation 5.4 and the by now very well known formula $q = \lceil \log(p+1) \rceil$:

$$q_i = \begin{cases} \lceil \log(i+2) \rceil, & \text{for } i = 0, \cdots, n-1 \\ \lceil \log(2n-i) \rceil, & \text{for } i = n, \cdots, 2n-2 \end{cases} \tag{5.5}$$

What remains to be done is to add all the carry bits which go into a certain column and the height of the partial product matrix part of that particular column, which gives us the number of input bits for that row, and thus we can calculate the cost.

Regrettedly, this figures cannot be calculated for all i's at the same time. Because carries are carried over to other rows, it is not possible in advance to tell the height of a certain row if the number of carries received from other rows is not known. Thus we need an iterative process like the following:

```
h_{0..2n-1}                    ; are initialized according to Equation 5.4.
for (i = 0, i < 2 * n, i++)    ; Iterate through all columns.
   q_i = ⌈log h_i + 1⌉         ; calculate number of output bits.
   for (j = 1, j < q_i, j++)
      h_{i+j} = h_{i+j} + 1    ; Add carries to respective columns.
   end
end
```

What we have got now is the number of input bits of the $2n$ counters, represented in the $h_{0..2n-1}$. With the number of output bits given by $q = \lceil \log(p+1) \rceil$ and equation 3.5 we have the cost in terms of number of interconnections. The sum of all (also intermediate) output bits gives the cost in number of gates. Table 5.1 gives the results.

Logarithmic Order Reduction

In Figure 5.4 we see the reduction of a 16x16 bit multiplier with **logarithmic**-order counters. To come to the delay and cost of this type of partial product matrix reductors,

Size	Gates	Inputs	Delay
8x8	47	386	33
16x16	119	1646	89
32x32	289	6886	227
64x64	685	29140	559
128x128	1595	125250	1341

Table 5.1: Cost and Delay of Partial Product Matrix Reduction Using Kautz **0**-Order Counters Calculated with the Algorithm

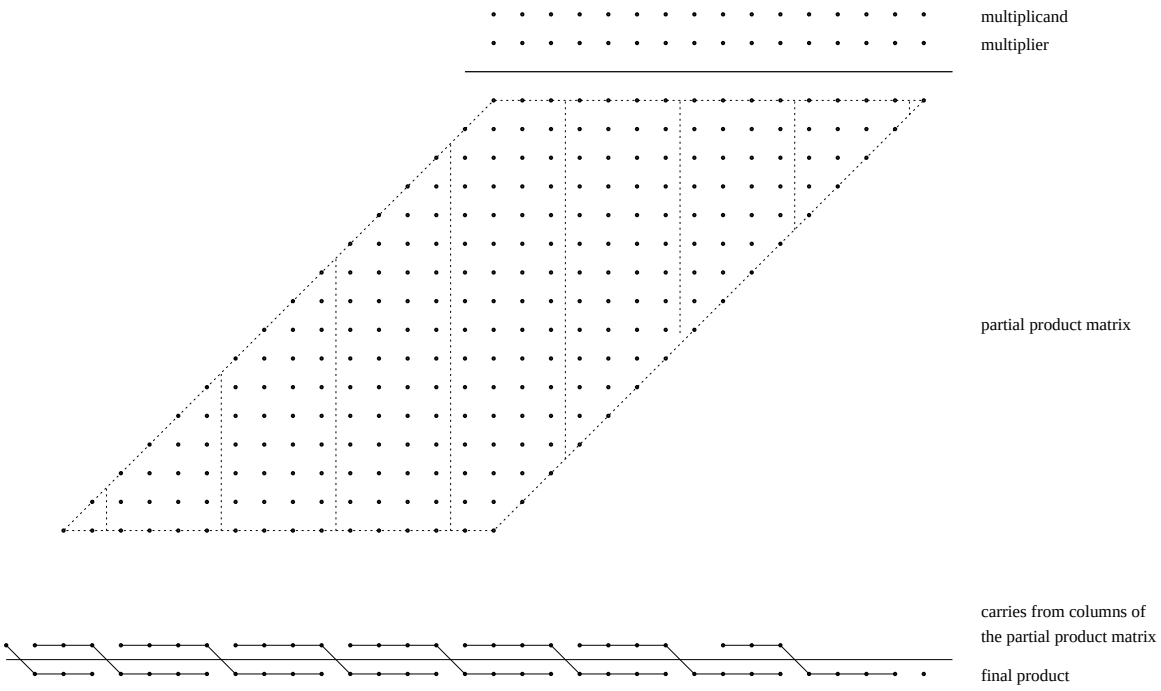

Figure 5.4: Logarithmic Partitioning of the 16x16 Bit Multiplication Matrix

Size	Gates	Inputs	Delay
8x8	26	475	14
16x16	57	2367	30
32x32	115	10827	57
64x64	234	50225	113
128x128	475	229675	227

Table 5.2: Cost and Delay of Partial Product Matrix Reduction Using Kautz Logarithmic Order Counters Calculated with the Algorithm

again some calculations must be made. What we effectively need is the amount of counters needed and per counters the number of input bits (the p_i) and the number of output bits (the q_i) which follows from the maximum input count as given in Equations 2.2 and 2.3. We will enhance the already known algorithm of the **0**-order reduction to give precise results.

The p_i is a sum of h_i's as given in Equation 5.4:

$$p_i = \sum_{j=(i-1)\log(n)+1}^{i\log(n)} h_j \tag{5.6}$$

which gives for $i = 0, \cdots, n-1$

A difficulty is that there is not in general a counter boundary at the exact symmetry point. Also at the far left of the matrix, a few columns remain who don't comprise a full $\log n$ counter. Thus these sums are not easily to express in a closed form.

The enhanced algorithm is:

```
h_{0..2n-1}                          are initialized according to Equation 5.4.
for (i = 0, i < 2 * n, i+ = log n)   ; Iterate through all columns.
   for (j = 0, j < b, j + +)         ; Iterate through one counter.
      p_i+ = h_{i+j}
      mc_i+ = 2^j h_{i+j}
   end
   q_i = ⌈log mc_i + 1⌉              ; calculate number of output bits.
   for (j = 1, j < q_i, j + +)
      h_{i+j} = h_{i+j} + 1          ; Add carries to respective columns.
   end
end
```

Table 5.2 shows the cost and delay of the logarithmic order ppm-reductors.

Size	Gates	Inputs	Delay
8x8	16	1144	16
16x16	32	8688	32
32x32	64	67552	64
64x64	128	532416	128
128x128	256	4226944	256

Table 5.3: Cost and delay of partial product matrix reduction using Kautz **2n**-order counters

2n-Order Reduction

Having looked at the highest degree of division of the partial product matrix, all columns apart in the **0**-order reduction, we now look at the lowest degree of division. Thus we use just one single Kautz' counter that is connected to the whole of the matrix.

This counter has n^2 inputs and $2n$ outputs. So the delay and number of gates are $2n$. The number of inputs according to Equation 3.5 is:

$$\text{Cost} = 2n^3 + \frac{1}{2}(2n-1)2n \tag{5.7}$$

Table 5.3 shows some results with this type of counters.

5.5 Serial-Parallel Multipliers

To reduce the cost of a multiplier, we can use an iterative process known as the serial-parallel multiplier. It determines in parallel just a part of the partial product matrix at a time. This part is then added to an accumulator. When all of the parts are done, the product is known.

In general this scheme is followed in a row-by-row basis. So the 'part' then is a row of the matrix which is very easily calculatable by multiplying one bit of the multiplier and the whole multiplicand. Which is , in the binary case, the same as adding the multiplicand to the accumulator if the multiplier bit is 'one' and adding nothing if it is 'zero'. After an addition a shift right is performed to keep the columns with the same weights together.

This scheme is much used in software implementation of multiplication, when no hardware multiplicator is available, and this is called as the add-and-shift algorithm.

In this section we consider by a serial-parallel multiplier every scheme which adds in parallel a part of the partial product matrix, adds the result to a accumulator and goes on to the next part. Note carefully that the dividing of the matrix is similar a problem as in the case of the parallel multipliers of Section 5.4. The only difference in here is

that we add all the parts with the same counter and the parts come in serially. Thus the approach is similar to the approach shown in Section 5.4.

5.5.1 Algorithm

In this column-by-column approach, as opposed to the row-by-row approach, we cannot multiply one word by a single bit of the other. Referring to Figure 5.2 we see that a column is formed with various indexes who's sum is equal to the column number.

In [Swa73] an algorithm for a single column is presented. as it is explained below, and later on a generalization is made for more-column counters.

We rewrite the Equation 5.3 for the magnitude of a binary multiplication while taking $m = n$ to:

$$A \cdot B = \sum_{i=0}^{n-1} a_i 2^i \sum_{j=0}^{n-1} b_j 2^j$$

and rewrite:

$$A \cdot B = \sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} b_j 2^j a_i \right) 2^i \tag{5.8}$$

The expression between the braces in Equation 5.8 is equal to the multiplication of one bit of the multiplier A with the entire multiplicand B. This is the basis of the row-by-row add-and-shift approach.

To come to the column-by-column approach, we rewrite again.

$$\begin{aligned} A \cdot B \quad &= \quad \sum_{i=0}^{2n-2} \left(\sum_{k=0}^{i} a_k b_{i-k} \right) 2^i \\ &\text{where} \quad a_i, b_i = 0, \qquad \text{for } i > n-1 \end{aligned}$$

The column based approach shows even clearer if we define s_i as being the sum of the i'th column of the matrix.

$$\begin{aligned} A \cdot B &= \sum_{i=0}^{2n-2} s_i 2^i \\ s_i &= \sum_{k=0}^{i} a_k b_{i-k} \end{aligned}$$

Note that s_i is a word of $\lceil \log i \rceil$ wide and, like in the parallel case, the least significant bit goes in the result register and the rest has to be added to the proper columns.

The algorithm described above can be implemented as shown in Figure 5.5, this picture is taken from [Swa73]. The big idea behind it is that the least significant bit of the multiplier is first shifted into the visible window opposing the most significant bit of the multiplicand.

In the first cycle, the least significant bits of multiplier and multiplicand are multiplied in an AND gate. The rest of the AND gates output a zero because the multiplier shift

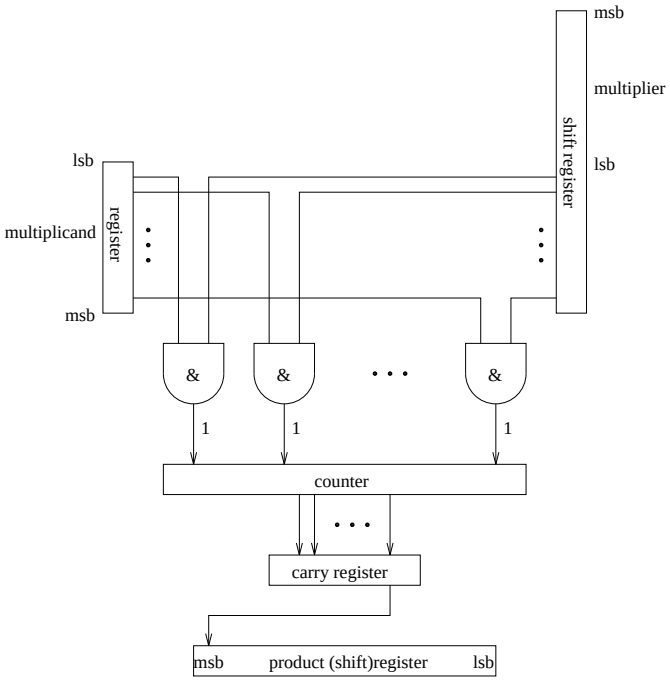

Figure 5.5: Circuit of the serial-parallel multiplier counting one column at a time

register is padded with zeros. The counter counts the result of the AND, and adds it to the output shift register. The multiplier is shifted down by one bit, causing the second columns of the partial product matrix to be generated by the AND-gates. It is again counted by the counter and added to the contents of the result register. Every cycle the least significant bit of the counter is a result bit. The rest are redirected with proper weight into the counter, so taking care of the carries. When $2n$ cycles are completed the product register contains the result.

The algorithm can be extended to facilitate higher order counters to reduce the number of cycles. Define w as the width (order+1) of the counter. Then:

$$A \cdot B = \sum_{i=0}^{\lceil \frac{2n-1}{w} \rceil} \left(\sum_{c=wi}^{wi+w-1} s_c \right) 2^i \tag{5.9}$$

So we add a w columns together to form a partial output. To accomplish this in the circuit we've got to add extra AND-gates to calculate the products. Figure 5.6 shows the circuit in the case of $w = 2$.

5.5.2 Cost Calculation

To calculate the delay of this construction, we have to take the algorithmic aspects into account. The delay of one cycle consists of the AND-gate delay, the counter delay, the delay of the carry register and the shift delay. Note that we take the delay of the

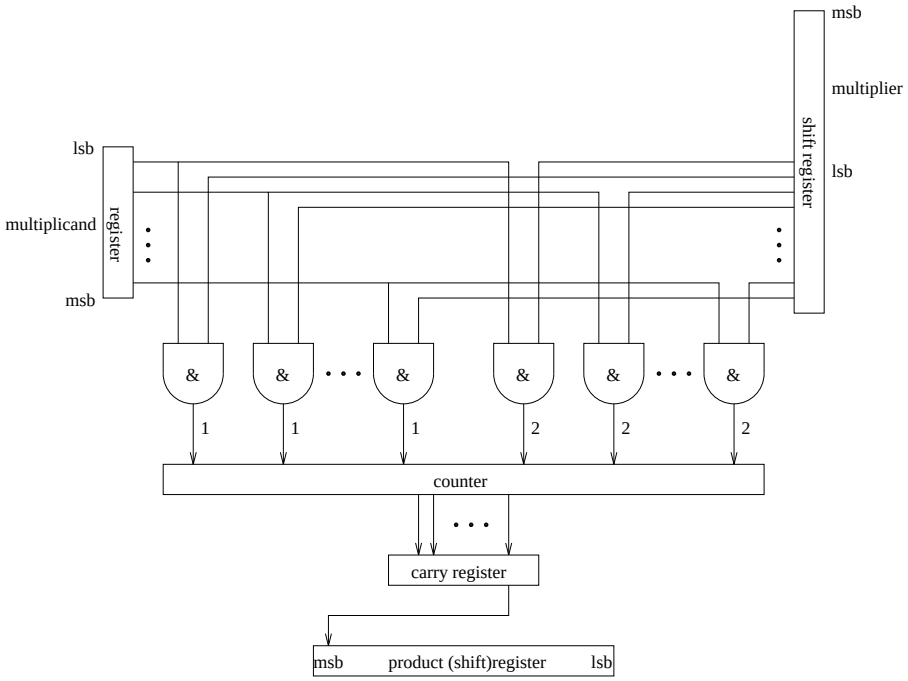

Figure 5.6: Circuit of the serial-parallel multiplier counting two columns at a time

AND-gates in account, what we did not do in the foregoing sections, because in this serial case we have to wait for them every iteration of the algorithm. Equation 5.9 tells us that the algorithm takes $\frac{2n}{w}$ cycles. The delay is:

$$D_{multiplier} = \left\lceil \frac{2n-1}{w} \right\rceil (D_{AND-gate} + D_{counter} + D_{carryregister} + D_{shiftregister})$$

With D for Delay. Taking $D_{carryregister} = 2$, $D_{shiftregister} = 1$ and $D_{AND-gate} = 1$ then

$$D_{multiplier} = \left\lceil \frac{2n-1}{w} \right\rceil (4 + D_{counter})$$

And the cost is equal to the cost of all parts together. However we have only take the counter into account analogous to all other cases we have shown throughout this thesis. The Cost and Delay parameters thus follow easily from Section 4.2 if we bear in mind that we only use one counter, thus do not multiply with the number of counters as is done in Section 4.2.

In Table 5.5.2 the parameters are given based on a **0**-order counter and in Table 5.5.2 the parameters are given based on a **logarithmic**-order counter. For the **0**-order reduction we will need one $(n \mid 1 + \log n)$ counter and for the **Logarithmic**-order one BA$(\log n, n, 2\log n)$ is needed. The Gates and Inputs parameter thus only reflect the cost of the single counter.

For more information on the subject of serial-parallel multipliers the interested reader is referred to [Swa73], on which this section is based.

n	Cost	Gates	Fan-in $_{max}$	Delay	Weight $_{max}$
8	30	4	11	128	8
16	74	5	20	288	16
32	175	6	37	640	32
64	405	7	70	1408	64
128	924	8	135	3072	128

Table 5.4: Cost Parameters of Serial-Parallel Partial Product Matrix Reduction Using Kautz **0**-Order Counters

n	Cost	Gates	Fan-in $_{max}$	Delay	Weight $_{max}$
8	159	6	29	54	32
16	540	8	71	96	128
32	1645	10	169	180	512
64	4674	12	395	342	2048
128	12635	14	909	659	8192

Table 5.5: Cost Parameters of Serial-Parallel Partial Product Matrix Reduction Using Kautz **Logarithmic**-Order Counters

5.6 Conclusions

In this chapter we investigated the area requirements when the other parameters are unrestricted. The major concern has been the smallest area multiplications for unsigned numbers, which also can cover signed number notation at no additional considerations. We have shown that the area required for multiplication is rather small when the other parameters are unrestricted.

In the next chapter we restrict the technology determined parameters such as fan-in and weight requirements and establish the area and delay cost of multi-operand addition which also covers multiplications.

6 Practical Considerations

In the foregoing chapters, we explained how the summation of n numbers can be accomplished by threshold logic. Using various order counters, we accomplished various values for the cost and performance parameters. Clearly, as the order of the counters used increases, the cost, number of gates and delay decreases. On the other hand, the Fan-in $_{\max}$ and Weight $_{\max}$ parameters increase. The primary concern of the previous two chapters was on the investigation of the smallest area requirements of the networks. The other parameters have been left unrestricted. The conclusions reached are indicative of small area networks but give us little understanding of the true capabilities of the threshold gates, except of course giving us the understanding of how small circuits can be at best. In this chapter, we deal with the influence restrictions of parameters have on the size and delay. We primarily concentrate on limiting the gate fan-in requirements. To do this, we will use a hierarchical decomposition of the addition which means that the algorithm is completed in multiple smaller steps. By restricting the fan-in of the gates we also restrict the weight requirements. In essence by restricting one of the parameters imposed by the technology the other is also restricted. We investigate only one possibility of design as an example. Clearly other possibilities also exist and are left as an continuing research topic.

We assume multi-operand addition as its cost includes the multiplication. For our example construction we assume also low cost counters so that the implementation is area "constrained".

The chapter is organized as follows: we first discuss hierarchical addition and block save addition for the reduction. We assume a counter implemented with the Kautz

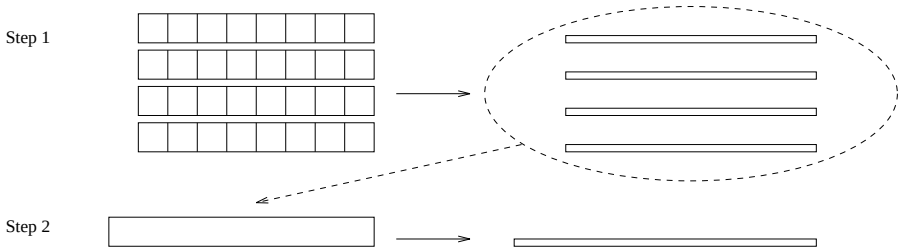

Figure 6.1: Hierarchical Addition using Two Reduction Steps.

method and determine the influence fan-in and weight restrictions have in a number of operand sizes.

6.1 Hierarchical Addition

Hierarchical addition operates as follows: instead of reducing the matrix of n numbers of $2n-1$ bits in one reduction step to one sum number, we break the n numbers down in b_0 matrices of n_0 numbers of $2n-1$ bits. Each block is reduced to one number in one reduction step as explained in the foregoing sections. On the next level, the b_0 resulting numbers are broken down in b_1 blocks of n_1 numbers each (hence $n_i = \lceil n_{i-1}/b_i \rceil$). This process is repeated until the final sum remains. The bit size of the partial results increases with approximately $\log n_i$ bits on every reduction step.

Figure 6.1 shows a two step example of this. The original matrix is divided in four matrices ($b_0 = 4, n_0 = n/4$). The four results are concatenated into one matrix of 4 rows in height. In the second step this matrix is transformed to the final sum ($b_1 = 1, n_1 = 4$).

A disadvantage of this scheme is that the delay may increase if the reduction is not carefully planned. In case each step in the process generates a whole sum and needs the full carry propagation through all counters, thus in the case of **logarithmic**-order counters that will add approximately $2n$ gate delay per each step. To circumvent this we will use a different technique that confines the carry propagation to the last step. The essence of this to not produce the sum but rather a partial sum.

6.2 Hierarchical Block Save Addition

Before going to Hierarchical Block Save Addition, we first have to explain what Block Save Addition is. Then we will extend it to Hierarchical Block Save addition, in the same we we did in the previous subsection.

In **Block Save Addition** [LB91], the adding of n numbers is completed in two steps. In the first step, the sum of n numbers is reduced to the sum of 2 numbers without any carry propagation between the blocks. The second step simply adds these two numbers. The carry propagation can not be avoided, it only is confined to the second

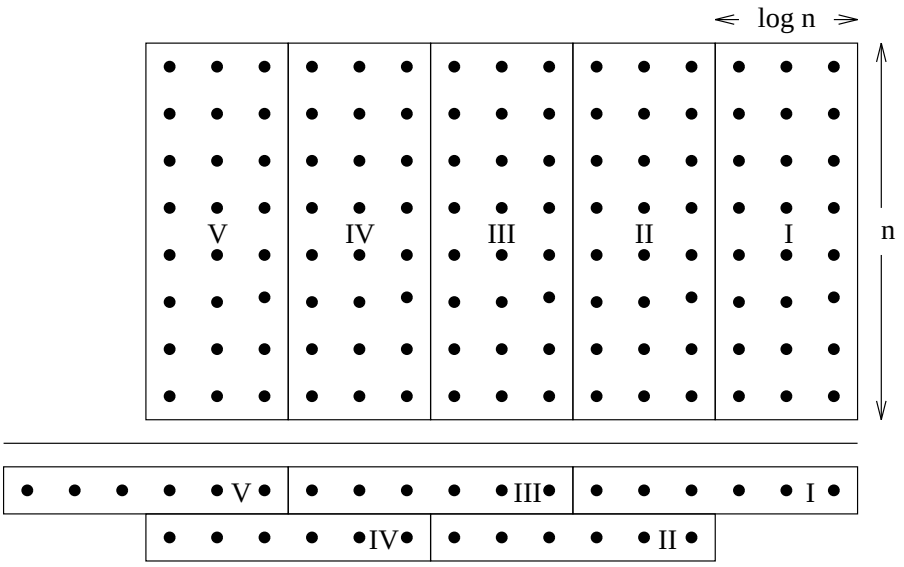

Figure 6.2: Block Save Addition of 8 15-bit Numbers

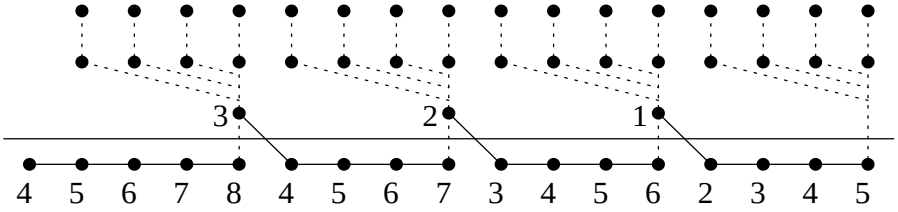

Figure 6.3: Addition of 2 16-bit Numbers, digits indicate the number of gate delays to calculate that output.

step. Figure 6.2 shows the first step of the principle, the inputs of a specific counter are enclosed by a square. The Roman number corresponds with the output bits of that specific counter who are also enclosed by a square.

The n numbers of $2n-1$ bits are divided into columns of $\log n$ bits. As we know, the sum of one such a column is at most $2\log n$ bits wide and thus overlaps only with the sum of the next column. As a consequence, the sums of all even columns do not overlap. The same holds true for the odd columns. The sums of all even columns are concatenated in one partial sum-word and the sums of the odd columns in another partial sum word. This first step takes only the time needed for the completion of one counter. This because all counters can work in parallel and do not have to wait for each other. In our case of Kautz counters, this will take $2\log n$ gate delay for a counter to fully complete all its output bits.

The **addition of the two partial sums** is another interesting subject. Because the sum of two binary numbers is at most twice the maximum of one summand, the sum contains at most one bit more than that summand. This is not influenced by the width of

the summands whatsoever. Consider Figure 6.3, the numbers next to the dots indicate the number of gate delays it takes to calculate that output. The interesting property is that there is at most one bit overlap between adjacent counters. The total delay thus calculates as the number of counters plus the time to finish the leftmost counter. The optimum number of counters with respect to total delay can be found considering the following formula with c the number of counters, q the number of output bits per counter and m the width of the summands:

$$\left.\begin{array}{lcl} \text{Total Delay} & = & c+q-1 \\ q & = & \frac{m}{c}+1 \end{array}\right\} \text{Total Delay} = \frac{m}{c} + c$$

Now we have the Total Delay as a function of summand size and number of counters. For a given summand size (thus m is a constant) we can differentiate the Total Delay with respect to c. We then find the zero point of this function and have our optimum:

$$\frac{\partial \text{Total Delay}}{\partial c} = -\frac{m}{c^2} + 1 \Rightarrow c = \sqrt{m}$$

Which will get us $q = \sqrt{m} + 1$ and a Total Delay of $2\sqrt{m}$.

If we use the Block Addition straight away, we will have the same fan-in and weight problems like before. Therefore we use the hierarchical principle again, the **Hierarchical Block Save Addition**. Just like the normal hierarchical addition we divide the addition in multiple matrices and multiple steps. The big difference is that every sub-matrix is treated as an block adder and generates two output numbers. These two output numbers are concatenated with the output numbers from other blocks and fed into the next stage.

We follow a similar argument as with the normal hierarchical addition. Instead of reducing the matrix of n numbers of $2n-1$ bits in one reduction step to two partial sums, we break the n numbers down in b_0 matrices of n_0 numbers of $2n-1$ bits. Each block is reduced to two number in one reduction step as explained in this sections. On the next level, the $2 \cdot b_0$ resulting numbers are broken down in b_1 blocks of n_1 numbers each (hence $n_i = 2 \cdot \lceil n_{i-1}/b_i \rceil$). This process is repeated until only two numbers remain.

Figure 6.4 shows a three step example of this. The original matrix is divided in two matrices ($b_0 = 2, n_0 = n/2$). The two results of two numbers are concatenated into one matrix of 4 rows in height. In the second step this matrix is transformed to two numbers ($b_1 = 1, n_1 = 4$). And the resulting two numbers are summed in a 2 number adder like we have seen in Figure 6.3.

In Table 6.1 we find some figures based on $(8, 8, 8 \mid 6)$ counters. This counter has an maximal fan-in of 29 and a maximum weight of 32. This weight can be found as the threshold of the most significant output gate. The 2 to 1 reduction is accomplished with $(2, 2, 2, 2 \mid 5)$ counters.

For example, in the case of an $n = 64$ adder, we have 64 numbers of 127 bits to add. With the $(8, 8, 8 \mid 6)$ counters, we will get $\lceil 127/3 \rceil = 43$ columns of counters. And the matrix will be divided into $\lceil 64/8 \rceil$ rows of counters. The 8 rows of counters generate $8 * 2 = 16$ results. This 16 results are reduced to four by 2 rows of counters. Now we

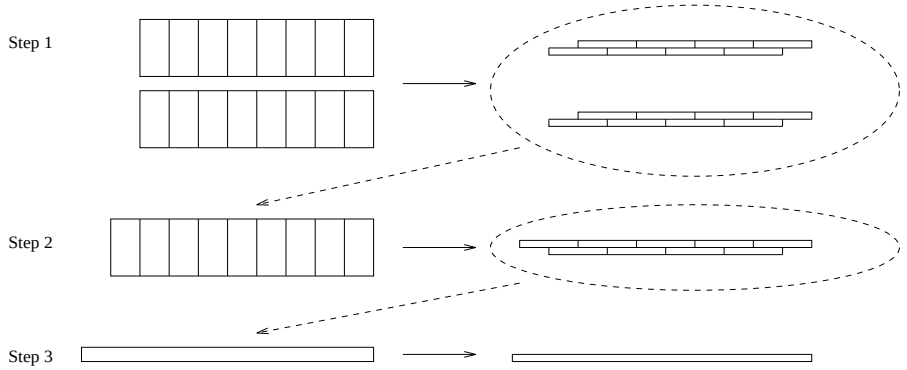

Figure 6.4: Hierarchical Block Save Addition Using Three Reduction Steps, All Carry Handling is Confined to the Last Step

Size	Cost	Gates	Fan-in $_{max}$	Delay	Weight $_{max}$
8x8	1054	55	29	15	32
16x16	4594	245	29	23	32
32x32	17704	721	29	33	32
64x64	72884	3020	29	53	32

Table 6.1: Cost Parameters for Hierarchical Multiple adders, Using Kautz $(8,8,8 \mid 6)$ and Smaller Counters.

use $(4, 4 \mid 4)$ counters to reduce this to two rows and finally reduce this two rows to one using $(2, 2, 2, 2 \mid 5)$ counters.

We note here the following when we also consider the previous two chapters.

- Restricting the fan-in and weights the delay of the implementations decreases substantially.
- Restricting the fan-in and weights the number of gates increases substantially.
- Restricting the fan-in and weights the cost increases substantially.

The major conclusion is that if the size of the operands is small the schemes presented in the previous chapter should be preferable for the implementation if the area cost is of major concern. In case this is not true and in case the operands are large there is no possible choice. Hierarchical reduction is always preferable and most likely imposed by the technology constraints.

6.3 Conclusions

In this chapter we considered some practical aspects of the design of multi-operand additions, and by extension to multiplications. We considered restrictions on parameters imposed by the gate technologies, i.e. gate fan-in and weight requirements, and establish the effect they have with example designs. We have shown that while the delay of the implementation can be improved the opposite effect holds true for the gate and overall cost.

In the final chapter we report a summary of the study and discuss some research topics.

Conclusions and Recommendations

7

7.1 Conclusions

As a conclusion of this study we report a summary of our investigations. We began by considering in Chapter 2 background information on parallel counters needed in the following chapters. We have described a convenient graphical representation which enables us to interpret circuits involving a multiple of parallel counters with ease. Furthermore a mathematical background, which includes the binary functions involved, has been discussed which has been used in the following chapters to evaluate larger circuits built with parallel counters. A valuable item is that a regular input metric like the block addition leads to a simplification of the formulas.

Consequently, in Chapter 3 we have seen that parallel counters can be efficiently build using threshold logic. The Basic and Minnick method provide the output in two gate delays. The Kautz method is the slowest method providing the output one bit at a time, but is very cheap in gate count and connection density. An important conclusion in this chapter is that we establish that the Kautz circuits for symmetric functions can be used to build inexpensive counting devices.

Given that the primary concern of this thesis is the area we used such counting devices as the basic blocks to produce higher level reductions.

In Chapter 4 we have shown how to combine the parallel counters to get bigger adders of larger input words. With the major concern being the smallest possible area constrained with all other parameters related, we investigated multi-operand addition. We have shown that using Kautz counters very small area is required for the design of

large multi-operand additions when the other parameters are left unrestricted. That is we have established somehow a lower bound for an implementation using Kautz counters.

In Chapter 5 we investigated, with the same assumptions as in Chapter 4, multiplication. We assumed that the multiplication is created using both signed and unsigned numbers. we investigated the area requirements when the other parameters are unrestricted. The major concern has been the smallest area multiplications for unsigned numbers, which also can cover signed number notation at no additional considerations. We have shown that the area required for multiplication is rather small when the other parameters are unrestricted.

In Chapter 6 we considered some practical aspects of the design of multi-operand additions, and by extension to multiplications. We considered restrictions on parameters imposed by the gate technologies, i.e. gate fan-in and weight requirements, and establish the effect they have with example designs. We have shown that while the delay of the implementation can be improved the opposite effect holds true for the gate and overall cost.

7.2 Recommendations

While we have investigated a number of issues regarding threshold logic, counting, multi-operand addition and multiplication, a number of issues require further investigations. Most notably, two issues need to be addressed, namely:

- Actually design of arithmetic units using threshold devices.
- More investigation for hierarchical matrix reductions. Possibly constrained further by fan-in and weight gate requirements.

Bibliography

[CL64] C.L. Coates and P.M. Lewis. “DONUT: A Threshold Gate Computer”. *IEEE Transactions on Electronic Computers,* pages 240–247, June 1964.

[Dad65] L. Dadda. “Some schemes for parallel multipliers”. *Alta Frequenza*, 34:349–356, May 1965.

[Dad80] L. Dadda. “Composite Parallel Counters”. *IEEE Transactions on Computers*, C-29(10):942–946, October 1980.

[GH86] S.L. George and J.L. Hefner. “High speed hardware multiplier for fixed floating point operands”. *U.S. Patent 4 594 679, col. 8*, June 10 1986.

[HC73] I.T. Ho and T.C. Chen. “Multiple Addition by Residue Threshold Functions and Their Representation by Array Logic”. *IEEE Transactions on Computers*, C-22(8):762–767, August 1973.

[HHK91] T. Hofmeister, W. Hohberg, and S. Kohling. “Some Notes on Threshold Circuits and Multiplication in Depth 4”. *Information Processing Letters*, Vol. 39, pp. 219-225, 1991.

[IBMa] IBM System/370 Extended Architecture Priciples of Operations, SA22-7085, IBM Corporation (available though IBM branch offices).

[IBMb] IBM System/370 Priciples of Operations, GA22-7000, IBM Corporation (available though IBM branch offices).

[Kau61] W.H. Kautz. "The Realization of Symmetric Switching Functions with Lineair-Input Logical Elements". *IRE Transactions on Electronic Computers*, EC-10:371–378, September 1961.

[LB91] R. Lauwereins and J. Bruck. "Efficient Implementation of a Neural Multiplier". *IBM technical report, RJ 8138*, May 1991.

[Min61] R. Minnick. "Linear Input Logic". *IRE Transactions on Electronic Computers*, EC-10:6–16, March 1961.

[Mur59] S. Muroga. "The Principle of Majority Decision Logic Elements and the Complexity of their Circuits". *International Conference on Information Processing*, Conference Proceedings, June 1959.

[PS90] R. Paturi and M. Saks. "On Threshold Circuits for Parity". *IEEE Symposium in the Foundations of Computer Science*, Conference Proceedings pp. 397-404, October 1990.

[SB90a] K. Y. Siu and J. Bruck. "Neural Computation of Arithmetic Functions". *Proc. IEEE*, 78(10):1669–1675, October 1990.

[SB90b] K.Y. Siu and J. Bruck. "On the Dynamic Range of Linear Threshold Elements". *IBM Technical Report, RJ 7237*, January 1990.

[SO92] T. Shibata and T. Ohmi. "A Functional MOS Transistor Featuring Gate-Level Weighted Sum and Threshold Operations". *IEEE Transactions on Electron Devices*, 39(6):1444–1455, June 1992.

[SO93a] T. Shibata and T. Ohmi. "Neuron MOS Binary-Logic Integrated Circuits- Part I: Design Fundamantals for Soft-Hardware Circuit Implementation". *IEEE Transactions on Electron Devices*, 40(3):570–575, March 1993.

[SO93b] T. Shibata and T. Ohmi. "Neuron MOS Binary-Logic Integrated Circuits- Part II: Simplifying Techniques of Circuit Configration and their Practical Applications". *IEEE Transactions on Electron Devices*, 40(5):974–979, May 1993.

[SRK91] K.Y. Siu, V. Roychowdhury, and T. Kailath. "Depth-Size Tredeoffs for Neural Computation". *IEEE Transactions on Computers*, Vol. 40, No. 12, December 1991.

[Swa73] E.E. Swartzlander jr. "The Quasi-Serial Multiplier". *IEEE Transactions on Computers*, C-22:317–321, April 1973.

[VB94] S. Vassiliadis and K. Bertels. "$O(n)$ Depth-3 Binary Addition". In Conf. Proc., editor, *IEEE 28th Asilomar Conf. on Signals, Systems, and Computers*, page to appear, October 1994.

[VBP94a] S. Vassiliadis, K. Bertels, and G. G. Pechanek. "Feed-Forward ANN for 2-1 Fixed Point ALUs". In Conf. Proc., editor, *IEEE Fourth International Conf. on Microelectronics for Neural Networks and Fuzzy Systems*, pages 156–161, September 1994.

[VBP94b] S. Vassiliadis, K. Bertels, and G. G. Pechanek. "O(n) Depth-2 Binary Addition with Feedforward Neural Nets". In Conf. Proc., editor, *IEEE International Conf. on Neural Networks*, pages 1381–1385, June 1994.

[VSH89] S. Vassiliadis, E. M. Schwarz, and D. J. Hanrahan. "A General Proof for Multi-bit Scanning Multiplications". *IEEE Transactions on Computers*, Vol. 38, No. 2, pp. 172-183, February 1989.

[VSS91] S. Vassiliadis, E. M. Schwarz, and B. M. Sung. "Hard-wired multipliers with encoded partial products,". *IEEE Transactions on Computers*, 40(11):1181–1197, Nov. 1991.

[Wal64] C. S. Wallace. "A suggestion for a fast multiplier,". *IEEE Transactions on Computers*, EC-13:14–17, Feb. 1964.