

**A Thesis Submitted for the Degree of PhD at the University of Warwick**

**Permanent WRAP URL:**

<http://wrap.warwick.ac.uk/135161>

**Copyright and reuse:**

This thesis is made available online and is protected by original copyright.

Please scroll down to view the document itself.

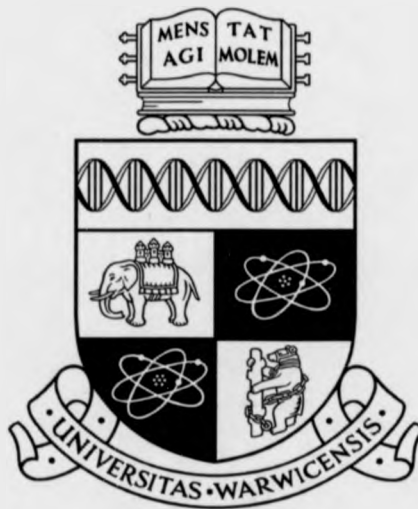
Please refer to the repository record for this item for information to help you to cite it.

Our policy information is available from the repository home page.

For more information, please contact the WRAP Team at: [wrap@warwick.ac.uk](mailto:wrap@warwick.ac.uk)

# **The Implementation and Applications of Multiple-Valued Logic**

Christopher Turrall Clarke



A Thesis submitted for the degree Doctor of Philosophy

The Department of Computer Science  
The University of Warwick,

November 1993

This thesis is dedicated to my mother and father who never just said  
"Because!" when I asked "Why?".



# Table of Contents

Dedication .....	i
Table of Contents .....	ii
List of figures and graphs .....	vii
Acknowledgements .....	xii
Declaration .....	xiii
Summary .....	xiv
Abbreviations .....	xv
1. Introduction .....	1
1.1. The thesis .....	2
1.1.1. Thesis aims .....	2
1.1.2. Thesis layout .....	3
1.2. Current processor technology and uses .....	4
1.2.1. Current trends in computer use .....	4
1.2.2. Multiple-valued logic .....	5
1.2.3. Digital circuits .....	7
1.2.4. Trigonometric functions .....	7
1.2.5. Number systems .....	7
1.2.6. Circuit production processes .....	10
1.3. Summary .....	12
2. Multiple-Valued Logic .....	13
2.1. Introduction .....	13
2.2. Current mode MVL .....	15
2.2.1. Summing nodes .....	18
2.2.2. Current mirrors .....	18
2.2.3. Switched current sources .....	20
2.2.4. Comparators .....	21

2.2.5. Binary logic.....	22
2.2.6. Partitioning current mode circuits.....	22
2.2.7. Current mode adders .....	24
2.2.8. Using current mode logic in networks .....	25
2.2.9. Major applications of current mode MVL.....	27
2.3. Voltage mode MVL .....	28
2.3.1. Power and time in voltage mode MVL .....	28
2.3.2. Basic circuits in voltage mode MVL.....	33
2.3.2.1. The diode connected transistor.....	33
2.3.2.2. The inverter. ....	34
2.3.2.3. Transmission switch.....	37
2.3.2.4. Encoders.....	38
2.3.2.5. Binary Logic.....	42
2.3.3. Applications of voltage mode MVL.....	43
2.4. Charge mode MVL.....	45
2.4.1. CCD MVL circuits .....	45
2.4.2. Switched capacitor MVL circuits.....	48
2.5. MVL memory.....	50
2.5.1. Register memory .....	50
2.5.2. Large scale MVL memories.....	51
2.6. Testing MVL circuits.....	53
2.6.1. Circuits for MVL testability.....	53
2.6.2. MVL test vector generation .....	54
2.7. MVL and VLSI production processes .....	54
2.8. MVL and optical technology .....	57
2.9 MVL versus Binary logic.....	58
2.10. Summary.....	58
3. Elementary Function Evaluation.....	60
3.1. Introduction .....	60

3.2. Nature of the problem.....	61
3.3. Non-iterative algorithms.....	62
3.4. Algorithms with quadratic convergence .....	63
3.5. Algorithms with linear convergence.....	64
3.5.1. The CORDIC algorithm.....	65
3.5.1.1. Origins.....	65
3.5.1.2. Basic CORDIC algorithm .....	65
3.5.1.3. Error and Convergence .....	67
3.5.1.4. Walther's extension .....	70
3.5.1.5. Hybrid Taylor series and CORDIC .....	72
3.5.2. Range extension.....	73
3.5.3. Scale Factor Compensation.....	75
3.5.4. Floating point CORDIC processors .....	76
3.5.4.1. Floating point pre and post processing .....	77
3.5.5. Redundant CORDIC processors .....	78
3.5.5.1. The Variable Scaling Factor Method .....	79
3.5.5.2. The Double Rotation Method .....	80
3.5.5.3. The Correcting Rotation Method .....	81
3.5.5.4. Branching CORDIC Rotation.....	82
3.5.6. Multi-Dimensional systems.....	84
3.5.7. CORDIC Applications.....	84
3.6. Higher Radix Algorithms.....	85
3.7. Summary .....	89
4. A design methodology for MVL circuits .....	91
4.1. Introduction .....	91
4.2. An MVL design methodology .....	92
4.2.1. A worked example: The maximum function.....	94
4.2.2. General function implementation.....	99
4.2.3. Ease of implementation of a function.....	100

4.2.4. Negative Currents.....	103
4.2.5. Multi-stage functions .....	106
4.3. Basic circuit functions for current mode MVL.....	106
4.3.1. The choice of functions .....	107
4.3.2. Encoders as unary functions.....	109
4.3.3. MAX and MIN.....	110
4.3.4. SUM and PRODUCT .....	113
4.4. Current mode MVL libraries .....	118
4.4.1. Abutment.....	119
4.5. MVL circuits in VLSI.....	121
4.5.1. The first test chip.....	121
4.5.2. The second test chip .....	124
4.6. Summary .....	127
5. Function evaluation in many dimensions.....	128
5.1. Introduction .....	128
5.2. CORDIC in more than two dimensions.....	128
5.2.1. 3D CORDIC.....	129
5.2.1.1. Basis of extension to three dimensions .....	129
5.2.1.2. Convergence and accuracy .....	133
5.2.1.3. Initial conditions .....	136
5.2.1.4. Application to scaleless 2D .....	137
5.2.1.5. Redundant numbers.....	138
5.2.1.6. Impact of the 2D Scaleless CORDIC Algorithm....	143
5.2.1.7. Hybrid systems.....	144
5.2.1.8. Higher radix algorithms .....	145
5.2.2. Evaluation of 3 Dimensional CORDIC algorithms.....	149
5.2.2.1. Two versus Three dimensional CORDIC .....	150
5.2.2.2. Three Dimensional Algorithms .....	152
5.3. 3D CORDIC in VLSI .....	158

5.3.1. Basic building blocks.....	160
5.3.2. The angle register pipeline .....	163
5.3.3. The WZ pipeline .....	165
5.3.4. The complete pipeline .....	167
5.4. Summary .....	169
6. Conclusions and Future Work.....	170
6.1. Conclusions.....	170
6.1.1. Direct MVL implementation .....	170
6.1.2. Indirect MVL implementation .....	172
6.1.3. Extensions to the CORDIC algorithm .....	173
6.2. Suggestions for future work .....	173
6.2.1. MVL circuits.....	174
6.2.2. MVL algorithms.....	174
6.2.3. The combination of MVL circuits and algorithms.....	175



# List of figures and graphs

## Figures

2.1. An Integrated Circuit whose area is bound by the pad ring. ....	14
2.2. Kirchoff's current law produces an MVL current mode signal. ....	16
2.3. The effect of connecting a single current mode output to a number of inputs. ....	17
2.4. A simple current mirror. ....	19
2.5. A switched current source. ....	20
2.6. A comparator. ....	21
2.7. A quaternary current mode full adder schematic. ....	22
2.8. Decomposition of a current mode circuit. ....	23
2.9. A Quaternary Full adder showing its DLE decomposition. ....	24
2.10. The two current mode adder types. Carry lines are omitted for clarity. ....	25
2.11. a) An MVL, and b) a binary Wallace tree structure. ....	26
2.12. The switching points for a) binary b) ternary and c) quaternary voltage mode signals. ....	32
2.13. Diode connected a) NMOS, and b) PMOS transistors. ....	33
2.14. An inverter a), and b) its transfer characteristics. ....	34
2.15. A modified inverter a), and b) its transfer characteristics. ....	35
2.16. a) A logical sum, b) a level shifter and c) a permutator derived from an inverter. ....	36
2.17. Inverter outputs ganged together to form a quaternary signal. ....	37
2.18. a) A 'perfect' transmission switch, and b) its logical symbol. ....	38
2.19. Encoders using a) resistive, and b) threshold voltage drop techniques. ....	39
2.20. A transmission switch based quaternary encoder. ....	40
2.21. An encoder based on threshold voltage drops. ....	41

2.22. Multiple power supply encoders. a) Transmission switch based. b) A simple driver. ....	42
2.23. The schematic of a T-gate. ....	43
2.24. A binary adder with a quaternary internal node.....	45
2.25. The movement of charge in CCD circuits. ....	47
2.26. The input of a switched capacitor circuit.....	49
2.27. Two current mode latch circuits. ....	51
2.28 I-V characteristics of a) and RTD b) an RTD in series with a resistor.....	52
2.29. A slowly varying device parameter (Oxide thickness in Angstroms). ....	55
2.30. A VLSI layout of a pair of transistors in series, with common centroid.....	56
3.1. The tangent, and square-root of a number.....	62
3.2. A hybrid ROM-Taylor Series Circuit [Ohhashi85].....	64
3.3. A CORDIC pseudo-rotation.....	66
3.4. The error in a rotation using a set of sub-rotations as in the CORDIC algorithm. ....	69
3.5. Daggett's number representation ....	74
3.6. A floating point CORDIC processor.....	77
3.7. a) A sign evaluation circuit for SBNR ....	78
3.7. b) Cell details for a sign evaluation circuit.....	79
3.8. The double rotation method. ....	80
3.9. The correcting rotation method. ....	81
3.10. The Branching CORDIC Rotation.....	83
3.11. The branching CORDIC algorithm (from [Duprat90]).....	83
3.12. The selection sets a) $\{-3,-2,-1,0,1,2,3\}$ and b) $\{-2,-1,0,1,2\}$ over two iterations.....	86
4.1. The DLE decomposition of a current mode circuit. ....	92
4.2. A maximum function decoder.....	99
4.3. The maximum function encoder. ....	99
4.4. An encoder with a negative output. ....	103

4.5. The encoder for the maximum function using a grounding output.....	104
4.6. The maximum function decoder and logic for use with the grounding output encoders.....	105
4.7. A multi stage function.....	106
4.8. A unary function in a circuit in DLE form.....	109
4.9. A unary function as the output of another multi input function.....	110
4.10. A complemented MAX encoder (simple encoding). ....	112
4.11. A complemented MAX encoder (grounded output encoding).....	113
4.12. The modulo 4 product decoder. ....	115
4.13. Encoders for a modulo 4 product. a) grounded output b) positive current. ...	116
4.14. A positive current modulo 4 product decoder.....	116
4.15. A multiplier cell block diagram. ....	117
4.16. An example schematic of a current mode circuit.....	118
4.17. The perimeter of library cells.....	120
4.18. The first test chip circuit. a) Circuit layout b) floorplan.....	122
4.19. Current mode modulo 4 sum. a) Layout, b) floorplan.....	123
4.20. The second test circuit. ....	125
4.21. The grounded output, 2 input encoder. a) Layout, b) schematic. ....	126
5.1. A vector in three dimensional space.....	130
5.2. Planes of rotation for a polar coordinate system.....	134
5.3. The sum of two vectors on the x-y plane.....	137
5.4. Rotating in 3D to remove the 2D scale factor. ....	138
5.5. A 3D rotation.....	139
5.6. The double rotation method for a 3D rotation.....	139
5.7. Example bit slice of redundant adder layouts for a) A unit scale factor system b) Takagi's suggestion. ....	143
5.8. Graph Legend.....	153
5.9. The fractional area used by the different parts of a) conventional and b) redundant arithmetic CORDIC processors.....	156

5.10. The Floorplan for a pipelined 3D CORDIC circuit (Redundant radix 4).....	159
5.11. Block diagram of a single digit redundant full adder.....	161
5.12. The redundant full adder single digit section.....	162
5.13. Bit register schematic. ....	162
5.14. Single digit register layout.....	163
5.15. A double metal NAND gate layout .....	163
5.16. Radix two angle evaluation circuit.....	164
5.17. The angle register pipeline organisation (bit slice).....	164
5.18. a) A radix two angle register single pipeline stage.b) Circuit block diagram. .	165
5.19. The layout of the WZ pipeline adders .....	166
5.20. a) A radix two stage of the WZ pipeline. b) Circuit block diagram.....	167
5.21. The WZ, and angle register pipelines. ....	168

## Graphs

2.1. The output characteristics of a) an ideal, and b) a typical 2 transistor current mirror.....	19
2.2. The energy used per cycle for bus structures of differing radices, but with the same voltage step between levels.....	30
2.3. The energy used per cycle for bus structures of differing radices, but with the same power supply .....	31
2.4. bus time delay as a function of radix assuming equal CR constants. ....	32
5.1. Gate counts for vector rotations with various required accuracies.....	151
5.2. Execution times for vector rotations with various required accuracies.....	152
5.3. Number of cycles taken in various 3D CORDIC algorithms (serial execution).....	153
5.4. Cycle period in gate delays for various 3D CORDIC algorithms (serial execution).....	154

5.5.	Overall time taken for various 3D CORDIC algorithms (serial execution). . .	154
5.6.	Area required for various 3D CORDIC algorithms (serial execution).....	155
5.7.	Cycle period in gate delays for various pipelined CORDIC algorithms.....	156
5.8.	Overall time in gate delays for various pipelined CORDIC algorithms. ....	157
5.9.	Area required in gates for various pipelined CORDIC algorithms.....	158

# Acknowledgements

I am indebted to many people for their help during my studies. My supervisor Prof. Graham Nudd has been a constant source of information, and advice, and has been instrumental in keeping this thesis on course. I would also like to thank Dr. Steve Summerfield for the many fruitful discussions that we have had, and for his assistance in the fabrication of the test chips described in this thesis.

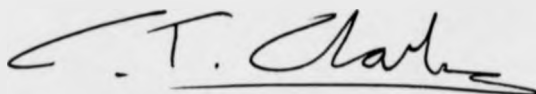
Many people have come and gone from the research group in which I have studied. I am very grateful to all the members of this group who have always been ready to help, when it was needed. The atmosphere of the group has a great help to me.

I am especially grateful to Vanessa Lever for proof reading this thesis, and for her encouragement, and understanding throughout our time together. Finally and most importantly I would like to thank my parents who have encouraged me to excel at whatever I do, and question anything I do not understand. Without them this thesis would not have happened.

# Declaration

This thesis has been prepared in accordance with the regulations for the degree of Doctor of Philosophy. It was composed by myself, and has not been submitted in any previous application for any degree. The work described in this thesis has been undertaken by myself except where otherwise stated.

The design of the current mode radix four product circuit is also described in [Summerfield92]. A description of the current mode multiple valued logic design methodology has been submitted for [Clarke94]. Work on the three dimensional CORDIC algorithm was presented as [Clarke92a] (which will be published), and was published as [Clarke92b].

A handwritten signature in black ink, appearing to read 'C.T. Clarke', with a horizontal line drawn underneath the name.

C.T. Clarke

November 1993

# Summary

Multiple-Valued Logic (MVL) takes two major forms. Multiple-valued circuits can implement the logic directly by using multiple-valued signals, or the logic can be implemented indirectly with binary circuits, by using more than one binary signal to represent a single multiple-valued signal. Techniques such as carry-save addition can be viewed as indirectly implemented MVL. Both direct and indirect techniques have been shown in the past to provide advantages over conventional arithmetic and logic techniques in algorithms required widely in computing for applications such as image and signal processing.

It is possible to implement basic MVL building blocks at the transistor level. However, these circuits are difficult to design due to their non binary nature. In the design stage they are more like analogue circuits than binary circuits. Current integrated circuit technologies are biased towards binary circuitry. However, in spite of this, there is potential for power and area savings from MVL circuits, especially in technologies such as BiCMOS. This thesis shows that the use of voltage mode MVL will, in general not provide bandwidth increases on circuit buses because the buses become slower as the number of signal levels increases. Current mode MVL circuits however do have potential to reduce power and area requirements of arithmetic circuitry. The design of transistor level circuits is investigated in terms of a modern production technology. A novel methodology for the design of current mode MVL circuits is developed. The methodology is based upon the novel concept of the use of non-linear current encoding of signals, providing the opportunity for the efficient design of many previously unimplemented circuits in current mode MVL. This methodology is used to design a useful set of basic MVL building blocks, and fabrication results are reported. The creation of libraries of MVL circuits is also discussed.

The CORDIC algorithm for two dimensional vector rotation is examined in detail as an example for indirect MVL implementation. The algorithm is extended to a set of three dimensional vector rotators using conventional arithmetic, redundant radix four arithmetic, and Taylor's series expansions. These algorithms can be used for two dimensional vector rotations in which no scale factor corrections are needed. The new algorithms are compared in terms of basic VLSI criteria against previously reported algorithms. A pipelined version of the redundant arithmetic algorithm is floorplanned and partially laid out to give indications of wiring overheads, and layout densities. An indirectly implemented MVL algorithm such as the CORDIC algorithm described in this thesis would clearly benefit from direct implementation in MVL.



# Abbreviations

ASIC	Application Specific Integrated Circuit
BILBO	Built In Logic Block Observation
CAM	Content Addressable Memory
CCD	Charge Coupled Device
CMOS	Complementary Metal Oxide Semiconductor
CORDIC	COordinate Rotation DIgital Computer
CR	Capacitor Resistor
DCT	Discrete Cosine Transform
DLE	Decode Logic Encode
ECL	Emitter Coupled Logic
ES2	European Silicon Structures
EXOR	EXclusive OR
FET	Field Effect Transistor
FPU	Floating Point Unit
IDCT	Inverse Discrete Cosine Transform
LED	Light Emitting Diode
LSB	Least Significant Bit
MAX	MAXimum
MIN	MINimum
MISR	Multiple Input Shift Register
MOS	Metal Oxide Semiconductor
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MSB	Most Significant Bit
MVL	Multiple Valued Logic
NAND	Not AND
NMOS	Negative Metal Oxide Semiconductor

NOR	Not OR
PLA	Programmable Logic Array
PMOS	Positive Metal Oxide Semiconductor
QFA	Quaternary Full Adder
RAM	Random Access Memory
ROM	Read Only Memory
RTD	Resonant Tunneling Diode
SBNR	Signed Binary Number Representation
SC	Switched Capacitor
TSUM	Truncated SUM
VLSI	Very Large Scale Integration

# 1

## Introduction

Multiple-Valued Logic (MVL) was used in early mechanical computers such as Babbages analytical engines that operated in decimal [Babbage1837]. However, since the advent of the electronic computer, MVL has become less used. This is due to the fact that the early electronic computers operated in binary. Subsequent generations of computers also used binary as it is simple to implement. There has been some interest in MVL in electronic circuits, and the Institute of Electrical and Electronic Engineers (IEEE) holds a symposium on the subject each year.

Computer applications such as image processing require very high arithmetic processing rates, and it is therefore necessary to explore potential areas of circuit design that could increase the processing rate of an Integrated Circuit (IC). MVL circuits have been shown to offer advantages of lower power and area for functions such as multiplication [Kawahito87], and the CORDIC algorithm for vector rotation has also proved itself useful [Yuen88]. Both of these functions are highly important in modern processors, and so the potential of the use of MVL style circuits with the CORDIC algorithm seems large. This thesis extends both MVL and the CORDIC algorithm. The combination of the two in the limited example of a binary implementation of a radix four CORDIC algorithm is shown. As and when current mode MVL circuitry matures sufficiently, there are yet more potential advantages to be gained by implementing the CORDIC algorithm using these circuits.

This thesis explores MVL both at the circuit and algorithmic level, and describes its applications in VLSI circuits. The introduction describes the aims and the layout of the thesis. The context in which the thesis is written is also explained.

## **1.1. The thesis**

### **1.1.1. Thesis aims**

This thesis investigates the applications of MVL and its implementation both directly using multiple-valued signals, and indirectly using groups of binary signals to represent the single multiple-valued signals. The relative advantages and disadvantages of MVL relative to the conventional techniques of using binary logic are also analysed. An important part of this thesis is the assessment of MVL relative to binary logic so as to give an indication of its suitability for various particular applications.

The direct implementation of MVL requires the construction of a set of basic logical operations to allow a full range of functions to be implemented. These basic logical operations are derived for 'current mode' MVL. The operations are then implemented in a common VLSI technology (CMOS) and they are assessed in terms of ease of design, and ease of manufacture. The effects of the present trends in computing technology on these designs is also investigated.

Indirect implementation of MVL has many applications. The evaluation of elementary functions such as sine and cosine are examples of this. The CORDIC algorithm which is capable of evaluating these functions by vector rotation is examined as an application for MVL implementation. The algorithm is extended to produce functions such as  $\sin A \times \cos B$  by vector rotation in three dimensions. MVL techniques are applied to the extended algorithm, and the resulting circuit designs assessed.

### 1.1.2. Thesis layout

The rest of this chapter is devoted to providing a context for the thesis. Current processing requirements are examined in order to find a motivation for the work. The concept of MVL is described, and the problems it inherits from being not as well used as binary logic are considered. Various number representations, and systems are discussed, as there are many practical options in an MVL environment. Finally the technology available for the production of MVL circuits, and the trends in the technology are described.

Chapter 2 evaluates the state of the art in MVL. The many circuits designed in MVL are described in categories according to the method used for indicating the value of a signal. Within each category, designs are shown to be built up of similar basic circuits at a level below that of the basic logic gate. In addition, the potential of MVL relative to binary logic is assessed for both present and future production technologies.

In chapter 3, methods available for the evaluation of trigonometric functions are described, including MVL methods. The CORDIC algorithm is examined in detail, and MVL modifications described. The state of the art in trigonometric function evaluation is shown to include MVL concepts.

Chapter 4 describes a new methodology for the design of MVL circuits using the novel concept of non-linear encoding of current mode signals. This new encoding strategy allows the designer to make much greater use of the 'free' summing node function than was previously possible. A worked example shows the methodology in use, and then basic logical operations are chosen and designed using the methodology. The methodology gives a good indication of how efficient the implementation will be at an early stage, and shows that obvious operations such as a sum are not necessarily the most efficient. The design of a cell library is considered, and a new set of guidelines for library construction given for the type of MVL circuit produced by the methodology given earlier. The results of test fabrications of MVL circuits are assessed to give an indication of the effect of the production technology on MVL circuits.

In chapter 5, The CORDIC algorithm is extended to the three dimensional case, and the error bounds of the new algorithm evaluated. The use of redundant numbers and other MVL techniques are shown to be applicable with these more complex function evaluations. This is followed by a detailed assessment of the different variations of the new three dimensional algorithm relative to each other as well as relative to the two dimensional CORDIC algorithm, and other methods described in chapter 3.

Finally, in chapter 6, conclusions are drawn from the earlier chapters. MVL is shown to be useful when the correct implementations are used for the correct applications.

## **1.2. Current processor technology and uses**

### **1.2.1. Current trends in computer use**

The motivation for the design of new more powerful computers is clear. More and more aspects of daily life involve computers. Increasingly, they are hidden from view controlling everything from the brown-ness of a piece of toast emerging from a toaster, to the routing of a telephone call to, say, Australia. As faster computers are designed, so more and more CPU intensive applications are found. For example, the United States Government has designated a set of computing problems as Grand Challenges including problems such as accurate long range weather forecasting, computer vision, and natural language processing. These problems will need computational power of the order of 1 Terra flop/s to give useful results [Wah93].

As computers become embedded more and more in to commercial products, more and more have to react in a non-binary manner to non-binary stimuli. The field of fuzzy logic has started address this problem, and whilst reading this thesis, the reader should be aware, that fuzzy logic is a non-binary logic; it is multiple-valued, although it is generally implemented using binary logic. The technology in which computers are constructed may change, in just the same way as for example the

material used for the construction of a wheel has changed, but there will be computers and they will be formed from logic in the same way that there will be wheels, and they will be round.

The design of early computers was based around a single unit, the CPU which performed one task at a time. In modern computers the CPU can spend a significant proportion of its time managing input from, and output to, other units within the machine that perform some tasks faster than the CPU itself could do them. Floating Point Units (FPUs), graphics processors, and sound generators are all examples. These units are designed to perform a small number of tasks quickly. They can do this because all the tasks the particular unit performs are similar, and so a large amount of effort can be put into making efficient designs for the operations used by all the tasks. Operations like trigonometric functions are now starting to be performed by dedicated hardware, rather than in software. For example, the INTEL 80387 includes specific circuits for some trigonometric functions [Yuen88]. The design of functions like these will become increasingly important in the future.

### 1.2.2. Multiple-valued logic

Multiple Valued Logic or MVL is digital logic using three or more states. Two valued logic is referred to as binary logic, and is considered to be different simply because it is used vastly more than all other types of logic in computer design. There are two major implementation types within MVL:

1. Direct electronic implementation.
2. Binary implementation of MVL algorithms.

In direct implementation of MVL, the basic transistor level circuits are designed to communicate with each other using multi-level signals on a single physical interconnection. This type of implementation substantially reduces the number of connections required for a given logical function. However, there are many problems

to be overcome since most integrated circuit processes are aimed at binary digital circuits, and hence other types of circuit are disadvantaged. Binary implementation of MVL algorithms is the use of standard well known and understood binary logic to implement algorithms formulated in systems with larger numbers of levels in each signal. In this case, a number of physical connections carry a single signal. It would be true to say that since the algorithm is implemented in binary, it must be possible to represent it as a binary algorithm. However, one of the major considerations in circuit design at present integration levels is the ease of design. So while it is possible to conceive of these multiple valued algorithms in their binary implementation form, it would often be counter productive to do so since time would be wasted in trying to understand concepts in binary that are simple in MVL.

Examining binary logic, which is well known, it can be seen that a circuit with for example two inputs, and no internal memory, can have only four states as shown on the left hand side of table 1.1. The circuit can output a true or false result for each state, so that for a two input, one output circuit, with no memory, there are sixteen possible functions that can be created. These functions are shown on the right hand side of table 1.1.

A	B	Output Functions															
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Table 1.1. Binary functions of two inputs.

This is  $r^n$  functions for a single digit output, where  $r$  is the number of values each digit can take (the radix in simple number systems), and  $n$  is the number of inputs. It is possible to look at the various available functions, and there are few enough for it to be reasonable to examine all the possibilities. The AND, OR and NOT functions are fairly intuitive to us since they are used in everyday language. These composite to form NOR and NAND which individually can form any function by combination with themselves. In Multiple-Valued Logic the choices are not so easy to see. For four



valued logic, there are 4294967296 or  $2^{32}$  functions to choose from for a two input one output function.

### 1.2.3. Digital circuits

Binary circuits are an everyday part of life. They are well understood, and so it is tempting when talking about them, to refer to binary circuits, as 'digital' circuits. While it is correct to say that a binary circuit is also a digital circuit, the converse is not necessarily true, as MVL circuits are also digital. In this thesis, a circuit that is two valued, will be referred to as a binary circuit, but when the word digital is used, the circuit referred to uses signals that have a defined set of discrete values. There may be two values, but there may be more.

### 1.2.4. Trigonometric functions

Basic trigonometric functions are usually non-linear functions of a single variable. More complex functions are sometimes calculable as easily. For example the CORDIC algorithm can perform a vector rotation as its basic operation. This can be used as a vector rotation, or as a sine and cosine generator. However, often in a computer programme, a sine or cosine would be calculated to perform a vector rotation, and so it may be better to give the programmer the choice of how to use the hardware. There are many other ways of calculating the sine and cosine, but for a particular application one solution may be better than the others. The important factor in the design of hardware for trigonometric function evaluation is the application, as this will determine the requirements for the evaluator. The different methods available for trigonometric function evaluation are explained in detail in a later chapter.

### 1.2.5. Number systems

In the past, many different number systems have been suggested (for example [Swartzlander75]). Most have been aimed at a particular application and hence show advantages for the operations that proliferate in that application. The most common

number system in use is the binary two's complement form. In this system, for a value  $A$ , the individual digits  $a_i \in \{0,1\}$  for  $0 \leq i < n$  are each scaled by  $2^i$ , except the digit  $a_{n-1}$  which is scaled by the factor  $-2^{n-1}$ . The value of  $A$  is therefore given by:

$$A = -2^{n-1}a_{n-1} + 2^{n-2}a_{n-2} + 2^{n-3}a_{n-3} + \dots + 2^0a_0$$

This gives an integer in the range  $-2^{n-1} \leq A \leq 2^{n-1} - 1$ . This range can be altered, and commonly, the scaling or significance of each digit is altered giving fractional accuracy at the cost of range. Basic arithmetic in this system is very simple, and important events such as overflow easily detectable. The main restriction of this system is that the carry is propagated successively from the least significant digit to the highest. Hence the complexity of a simple addition is  $O(n)$ , and the complexity of a multiplication by a simple shift and add technique is  $O(n^2)$ . Much effort has been put into reducing these complexities, for example [MacSorley61, Wallace64]. However, it will be seen that other arithmetic systems can offer substantial benefits over the two's complement scheme, indeed they often do so even when the assessment includes starting with a two's complement number, and finishing after the conversion back to two's complement has been effected.

Two possible unconventional number systems that might be considered useful are residue number systems, and redundant number systems. Residue number systems operate by having a number of data paths working in parallel, each of which has much smaller value ranges than in a conventional data path. Each data path operates modulo a different number. The data paths themselves are generally constructed from binary circuits, but will operate much faster since they have shorter carry chains. The modulus used for each data path is critical. If the moduli are relatively prime, then the range of representable numbers  $A$  will be  $0 < A < \prod(\text{moduli})$ . For example, consider two moduli 3 and 5, table 1.2 shows the representation of the values zero to fourteen. The useful fact about the system is that linear functions such as additions and multiplications are performed simply by performing that operation within each data path. No communication is needed between the data paths. There is however, one problem with the residue number system. It is

difficult to determine whether one value is greater than another without converting them into another number system, although methods are now available [Dimauro93].

$A$	$ A _3$	$ A _5$
0	0	0
1	1	1
2	2	2
3	0	3
4	1	4
5	2	0
6	0	1
7	1	2
8	2	3
9	0	4
10	1	0
11	2	1
12	0	2
13	1	3
14	2	4

Table 1.2. A residue number system.

A redundant number system is a number system in which any value can be represented in more than one way. A commonly used redundant number system is the Signed Binary Number Representation (SBNR) in which a variable  $A$  is made up of digits  $a_i$ , each scaled by  $2^i$ , but instead of each digit being a single bit,  $a_i \in \{-1, 0, 1\}$ . This allows a number of representations of each number in the range  $-2^n < A < 2^n$ , for example, if -1 is represented as  $\bar{1}$  the number 1 can be represented as  $1\bar{1}$ , or  $1\bar{1}\bar{1}$ , etc. The advantage of the redundant number system, is that adders can be constructed that cause the carry to be propagated not more than two digits up the word. This significantly increases circuit operation as addition becomes an  $O(1)$  problem in terms of time. As with residue number systems, this increases the time taken for a comparison from an  $O(1)$  problem for binary to an  $O(n)$  problem for SBNR. It will be shown later that this can be reduced to  $O(\log_2 n)$ . This increase in time delay for this kind of operation can be overcome, to give fast implementations of many circuits. Conversion back to binary from SBNR can be done using a conventional two's complement adder to subtract the negative digits from the positive digits. The advantages of an unconventional number system will be lost if they use the same clock

## 1.8: Circuit production processes

cycle as full word length two's complement operations, since their advantages are obtained by operating correctly with faster clocks. Care in design is needed to ensure that the advantages are not lost because of problems such as this. Redundant number systems use non-binary signals, and so they can be viewed as MVL circuits, even though they are often implemented by using two bits for each digit. It is possible to create redundant number systems that are not radix two. These number systems can be used to great effect when implemented in binary or multiple-valued circuits [Kawahito88].

### 1.2.6. Circuit production processes

The design of computing circuitry (whether electronic or otherwise) cannot be done in complete isolation from information about the production processes used to construct the circuit. The designer of a micro-processor does not necessarily need to understand the doping profiles in the semi-conductors being used to construct the logical gates, but an understanding of the kind of operation that can be performed efficiently is important for the overall design to be efficient itself. For the design of MVL and analogue circuits, more needs to be known about the transistors and other circuit devices used as the linear, as well as switching characteristics of the devices are used. This section gives brief details of the various major production technologies. Detailed figures are not given, as these vary from one fabrication line to another. Instead, an insight into the similarities and differences of the technologies is given.

The Complementary Metal Oxide Semi-conductor (CMOS) technology is derived from the older Negative and Positive Metal Oxide Semi-conductor (NMOS and PMOS) technologies [Gise88]. This technology is, at present the most widely used technology because of its low cost, and high circuit densities. The circuits are constructed from N and P type Field Effect Transistors (FETs). These transistors act as voltage controlled current sources, with a very high input impedance, and moderate output impedance. In a switching mode, the transistor switches on when about 1 volt is applied to the gate (the input) relative to the source (the common connection). The

switch on or threshold voltage is affected by many parameters, and varies considerably between transistors that are fabricated next to each other on a circuit. The main use of CMOS circuits is in digital circuitry, as it is difficult to design high precision analogue circuitry in this technology. This is due to the variability of the transistor parameters. CMOS circuits operate at moderate to high speeds.

Bipolar technology is based around the npn, and pnp bipolar transistor [Gise86]. Circuit densities are low, but transistor parameters vary much less than FET parameters. Bipolar transistors act as current controlled current sources, with a minimum switch on voltage of about 0.7 volts. This value is very stable. The linearity of the transistor transfer characteristics is far higher than that of FETs, and consequently, analogue circuits are often designed in the technology. The switching speed of bipolar circuits is also high relative to FETs, because the output impedance of a bipolar transistor is low. Input impedance is considerably lower than that of FETs, and so circuit fan-out can be a problem. High speed digital circuits are constructed using bipolar transistors in a logic family called Emitter Coupled Logic (ECL). These circuits are large, and dissipate large amounts of power [Haznedar91].

Bi-CMOS technology is a combination of bipolar, and CMOS technology. Basic bipolar transistors are made available to the designer. They are large, but can be used to drive bus lines, and other highly capacitive nodes. The correct use of relatively small numbers of the bipolar transistors can greatly increase the overall speed of operation of the circuit. The bipolar transistors can also be used as stable voltage references, or for analogue circuits. This is clearly shown by the early production of data converters in this technology [Haznedar91]. This technology is becoming more and more widely used, and has potential for MVL circuit design in the future.

Charge Coupled Device (CCD) technology is often used to produce camera input circuits, but can also be made to perform logical operations [Nash82]. The technology is based on charge storage under the gates of multiple gate FETs. This technology is described in more detail in a later chapter.

Optical logic is emerging as a potential new technology for computers. Light is used to indicate signal values, and gates are formed from composites of photo-detectors, and Light Emitting Diodes (LEDs) [Osawa92]. One major advantage of this technology, is that it is possible to construct devices whose inputs and outputs are on different sides of a semi-conductor wafer. This allows very parallel data paths to be constructed by stacking wafers. Integration techniques are already available for routing light signals around the surface of an integrated circuit if necessary [Seymour88].

### 1.3. Summary

In this chapter, the thesis and the context in which it is written have been described. The thesis deals with MVL primarily in a VLSI environment, but also considers potential future technologies such as optical logic. A short introduction to the concept of MVL has been given, describing both direct and indirect implementations. The major differences between MVL and binary logic have been explained. Number systems have been briefly described as many MVL number systems are available. The current major fabrication technologies have been described in terms of their characteristics that are relevant to MVL design.

# 2

## Multiple-Valued Logic.

### 2.1. Introduction

Modern commercial computers are constructed from integrated circuits that operate on binary signals. This is due to the way in which computers have evolved over the past forty years. Multiple Valued Logic (MVL) is simply logic that can deal with signals that have more states than the 'true' and 'false' or '1' and '0' available with binary logic signals. Binary logic is, in effect a very simplified case of MVL.

The idea of using non binary signals in computers is not a new one. Indeed some early computers were not even digital. Early digital computers such as the analytical engines designed by Babbage [Babbage1837], and most of the subsequent mechanical computers did their computations in decimal. This seems an obvious choice since humans also work in decimal, and so problems would be formulated, and answers required in decimal. Early electronic computers suffered from large amounts of electrical noise, and because of this, they were constructed using binary circuits as this offers the largest possible noise immunity. As electronic computers evolved, compatibility with old designs, ease of production, and the ability to use Boolean logic for all parts of the design made the use of binary the accepted norm. For this reason, digital design became further and further removed from analogue design.

Recently, as it has become possible to place whole systems on a single VLSI chip, designers have started to put analogue circuitry onto their devices along with

binary integrated circuits. This clearly shows that the original reasons for using binary logic do not necessarily still hold. In addition, the wiring inefficiencies of binary are starting to impact designs. Integrated circuit packages are having to provide more and more pins to connect the IC to the outside world. This can lead to circuits whose area is defined not by the actual circuit, but by the ring of pads for connection to the pins of the package which must be placed around the edge of the circuit. This is shown in a diagrammatic form in figure 2.1. The power consumption of integrated circuits is also becoming increasingly important as packing densities go up creating the risk of overheating. A major factor in this power consumption is the large percentage of the device taken up with buses, as these can have large parasitic capacitances to ground.

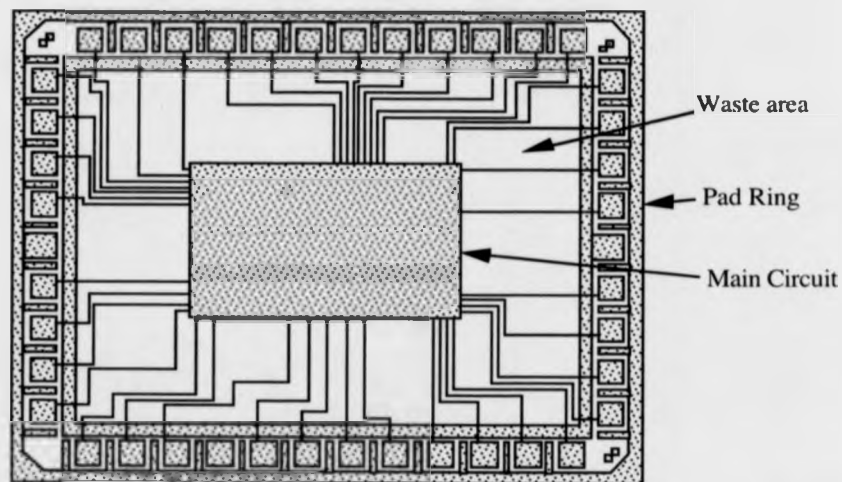


Figure 2.1. An Integrated Circuit whose area is bound by the pad ring.

The use of higher radices in digital circuits reduces the number of pins required for the circuit. In addition, it reduces the number of interconnections between the different parts of the individual integrated circuits, and hence the area of the circuit.

In this chapter, the use of higher radix digital logic (commonly referred to as Multiple-Valued Logic or MVL) is discussed. Initially, the potential of MVL is examined in its three major forms: current mode logic, voltage mode logic and charge mode logic. This is followed by an overview of the state of the art in the major areas



of MVL design. Finally, the impact on MVL of the trends in production technology is examined, and MVL is compared with binary logic.

The logic theory that provides a framework for the design of larger circuits, and of whole systems is not examined. There are good books [Muzio86, Epstien93], that can be used to gain an understanding of the subject. The theory of logic is a very large subject, and a treatment of it would simply serve to obscure the multiple valued circuit theory. Instead, it is sufficient to simply observe that there are sets of logical functions that are capable of the generation of all other logical functions. Once free from the limitations of binary circuits, there is a new and fundamental question to be asked: "What is the best number of levels to use?". There is no absolute answer to this question, but it has been shown [Hurst84] that higher radices have more potential for efficient circuitry, although this is dependant on the nature of the growth of the complexity of a circuit relative to the radix. The majority of modern practical circuits today are radix 4 (quaternary). The reason for this is that it is the usually highest attainable radix of the form  $2^m$ , and hence it is easy to encode from, and decode to binary. Higher radices of the same form often suffer from too little noise immunity.

## 2.2. Current mode MVL

Current mode logic uses the flow of current to represent a signal value. This contrasts with conventional voltage mode, in which the voltage level represents the signal value. Current mode logic has been used previously in binary circuits, for example in Current Injection Logic (known as  $I^2L$ ) [Texas], and current loop terminal lines. Some of the advantages of  $I^2L$  over MOS logic types include:

- Large temperature operating range.
- Low power consumption.
- Radiation hardness.
- Low internal stress voltage.
- High reliability.

## 2.2: Current Mode MVL

The reason for a number of these advantages is that  $I^2L$  is a bipolar process. The use of  $I^2L$  gives CMOS levels of circuit density with some of the advantages of bipolar technology. A similar circuit approach can be adopted in a CMOS technology, and some benefits are still seen, especially for MVL circuits. CMOS technology is now so widely used that for MVL circuits to be used they would have to be capable of being fabricated in CMOS. In the following discussion, only CMOS circuits will be described, but bipolar counterparts exist, and are equally well used.

The major advantage for CMOS current mode MVL, is not related to the above advantages, but stems from Kirchhoff's current law which states that the sum of currents at a node in a circuit is zero. This allows a number of circuit outputs to be connected together, and used as the input to another circuit as shown in figure 2.2. In effect this is a circuit with a high fan-in. It would, in theory, be possible to connect together as many inputs as required. If all that is required is a signal if any one of the inputs is active, then this is possible. However, due to inaccuracies in transistor parameters, it is not possible to construct a circuit that can accurately calculate how many inputs are active, beyond a certain fairly small number.

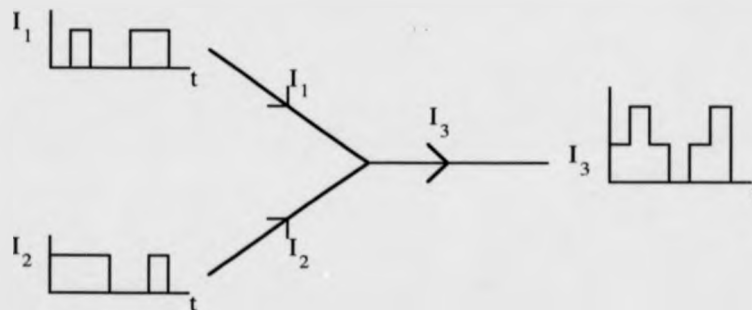


Figure 2.2. Kirchhoff's current law produces an MVL current mode signal.

Kirchhoff's current law also gives rise to the other major difference between voltage and current mode circuits which is fan-out. The effect of fan-in has been shown to be that the input currents are summed, but if an output of a circuit is fed to more than one circuit input, the current will be shared between the inputs, but not necessarily evenly. The result will be undefined. This is due to the fact that the two inputs will have

different impedances (which in turn is caused by transistor mismatches, and differing wire lengths to the inputs), which causes the split of the flow of current to be uneven. This is shown in figure 2.3, where it can be seen that  $I_1$  splits into  $I_2$  and  $I_3$ , however the ratio  $I_2:I_3$  is unknown.

So far only currents flowing from an output to an input have been considered, but it is possible to have circuit outputs that have current flowing into them. This means that circuit outputs can source or sink currents or both. Circuits in which outputs either source or sink currents but not both are referred to as uni-directional circuits as the current only flows one way. Circuits whose outputs both source and sink currents are referred to as bi-directional circuits. It should be noted, that since the sources of error in a current mode circuit are largely unaffected by the direction of the current, the number of levels obtainable with a bi-directional circuit is nearly twice that of a unidirectional circuit. However there is a penalty in terms of circuit complexity, and bi-directional currents need to be split into their positive and negative components before they can be subjected to some kind of threshold operation.

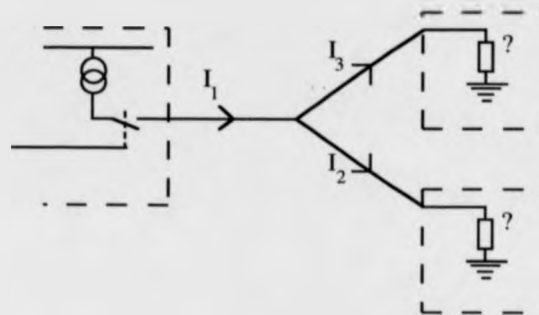


Figure 2.3. The effect of connecting a single current mode output to a number of inputs.

All current mode circuits are built up from the following basic circuit elements although they are not always represented or implemented in the same way. The basic five elements are:

1. Summing nodes.
2. Current mirrors
3. Switched current sources

## 4. Comparators

## 5. Binary logic

These elements are described in more detail in the following sections. After this, the partitioning of MVL circuits is examined. Important applications of MVL are then described, with particular attention to adder circuits, and the layout of interconnect intensive circuits such as the Wallace tree [Wallace64].

## 2.2.1. Summing nodes

The summing node is often referred to as a 'free function' since no logic is required to implement it. All that needs to be done to sum two current outputs is connect those outputs together, and the result is the sum. This sum can be larger than the radix of the circuit, and this must be compensated for. From an arithmetic perspective, this is simply an indication that a carry to the next digit is required.

## 2.2.2. Current mirrors.

The two transistor current mirror is a very simple device, as shown in figure 2.4. Its operation in basic terms is as follows. The input current  $I_i$  flows through transistor  $M_1$ . If the resistance of  $M_1$  is too high, then the remaining current that cannot pass through  $M_1$  charges up its gate (and that of  $M_2$ ) reducing the resistance of  $M_1$ . Thus the gate voltage of  $M_2$  is set such that if the difference between the gate and drain voltage is small, a current of the same magnitude as  $I_i$  will flow into the drain of  $M_2$ . The above description assumes that the width to length ratios of the two transistors is the same. This need not be the case however, and this fact can be used to great effect. If transistor  $M_2$  has a  $\frac{W}{L}$  ratio of  $aS_1$  where  $S_1$  is the  $\frac{W}{L}$  ratio of  $M_1$ , then the output current  $I_o$  will be  $a$  times the magnitude of  $I_i$ . Alterations in the  $\frac{W}{L}$  ratio are usually made by altering the gate width because altering the gate length of  $M_2$  relative to that of  $M_1$  will affect the symmetrical nature of the current mirror.

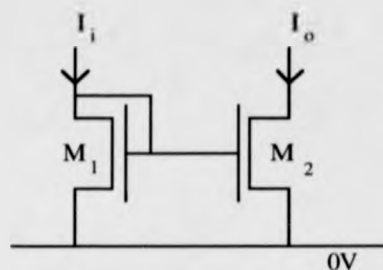
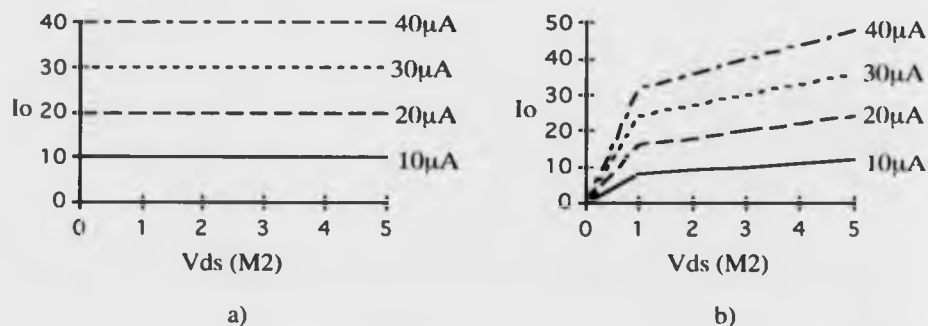


Figure 2.4. A simple current mirror.

The drain-gate voltage of  $M_2$  is not, of course, always going to be small. The effect of changes in the drain gate voltage for a typical current mirror are shown in graph 2.1 along with the ideal characteristics (Input current given on right hand side of graph). These effects can be overcome by the use of more complex current mirror circuitry, but more complex circuitry requires more area. There is a clear trade off between accuracy and area.



Graph 2.1. The output characteristics of a) an ideal, and b) a typical 2 transistor current mirror.

The current mirror allows an addition to be turned into a subtraction, and vice versa for uni-directional systems. It can also be used to produce multiple copies, of an input, each multiplied by a constant factor. The simple current mirror is however, not a perfect device as has already been explained. It has many potential problems, and so more complicated mirrors such as the cascode mirror are sometimes used [Freitas83]. For bi-directional systems, a slightly more complicated mirror is required, but it allows the inversion, and replication of an input current. In addition, it can be used to

split the bi-directional current into its positive and negative parts, i.e. a pair of uni-directional currents. A small modification to the current mirror provides a dynamic memory facility for current mode logic [Current89, Xu89, Current91]. Placing a pass transistor in between the gate of  $M_1$  and the gate of  $M_2$  allows charge to be stored on the gate. It is then possible to determine the amount of charge on the gate by using the transistor as a current sink (or source for a PMOS mirror). Clearly, this is a dynamic type of storage that would need to be regularly refreshed.

### 2.2.3. Switched current sources.

A current source is essentially the output stage of a current mirror. This source can be used on its own to supply constant current, or more usually with a transistor in series as shown in figure 2.5, to form a switched source.

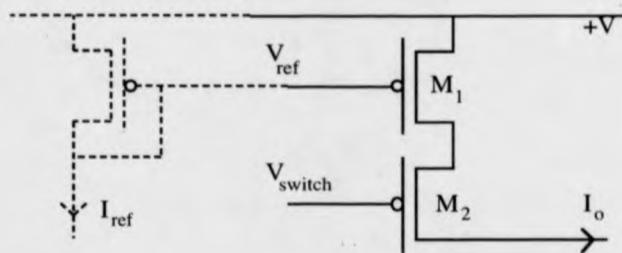


Figure 2.5. A switched current source.

As with the current mirror, it is possible to use a single reference voltage to produce a number of sources each with a different current output value. This can be done by altering the width of the source transistor relative to the transistor used to produce the reference voltage. The width of the switch transistor will be determined by the maximum allowable voltage drop for the maximum current output. The input  $V_{switch}$  is a voltage mode binary signal, and as such it can be driven from conventional CMOS binary circuits. It is possible to use a number of switch transistors in series and in parallel, to produce a logical function of the input voltage mode signals.

### 2.2.4. Comparators.

A comparator gives an indication of whether a signal is above or below a pre-determined level. This is simply done by current subtraction as shown in figure 2.6. The positive input current  $I_i$  is mirrored (making it negative  $I_n$ ), and summed with a reference current  $I_r$  produced by a positive un-switched current source. The difference in the currents  $I_n$  and  $I_r$  either charges or discharges the nodes' parasitic capacitance  $C_p$  according to whether  $I_n$  or  $I_r$  is greater. Hence an output voltage  $V_o$  is produced that is high if  $I_i$  is lower than the threshold, and low if it is higher than the threshold. Bi-directional comparators can be constructed using a current mirror to split the positive and negative currents, and then uni-directional comparators are used on these signals.

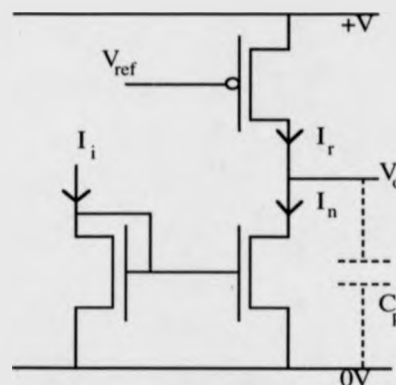


Figure 2.6. A comparator.

Comparators are required so that a current mode signal level can be restored. The comparator is often combined with switched current sources, but it should be noted that the communication between these two sections is a voltage mode binary signal. Comparators require references to compare against. These can be generated locally as with Kawahito's circuits [Kawahito87], or broadcast as a gate voltage for a current source local to the current mode circuit to use. In practical implementations a stable on-chip reference would be required for example [Sansen88], or accurately defined transistor parameters as used by Kawahito et al.

### 2.2.5. Binary logic.

Signals between comparators and switched current sources are binary voltage mode signals. Hence this is an ideal place to use standard binary logic. This is used to great effect by Current et al. [Current87] in the construction of a quaternary full adder as shown in figure 2.7. The full adder inputs are two quaternary signals, and a carry signal which is a binary current mode signal. All three of these inputs are joined together to form a single octal current mode signal. Seven comparators are then used to distinguish between the different possible input levels. The values shown on the comparators are the switching levels, and are given in units of the standard unit current. The output of these comparators is fed into binary logic, and from there to switched current sources that provide a quaternary current mode sum output, and a binary current mode output.

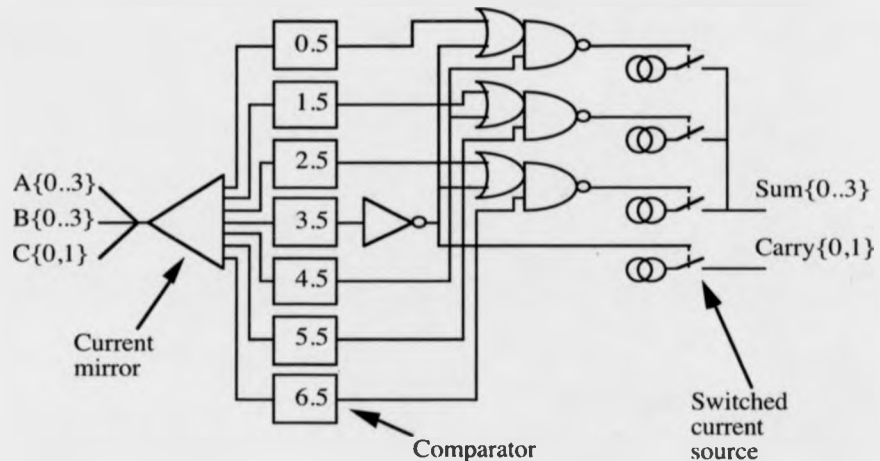


Figure 2.7. A quaternary current mode full adder schematic.

### 2.2.6. Partitioning current mode circuits

The basic building blocks for current mode MVL have been described. However, there are many other ways of partitioning these circuits into logical blocks. For example, a comparator is often followed by a switched current source, and so designers sometimes combine the two functions together. Figure 2.8 shows two



different methods of decomposing a comparator followed by a switched source. The decomposition in figure 2.8a is that used by Kawahito et al. [Kawahito87, Kameyama88]. Kawahitos' circuits are slightly different to those shown in figure 2.8a, as their circuit makes use of a high accuracy depletion mode PMOS transistors in a diode connection instead of enhancement mode PMOS transistors, and a reference voltage. However, the operation of the circuit is still the same.

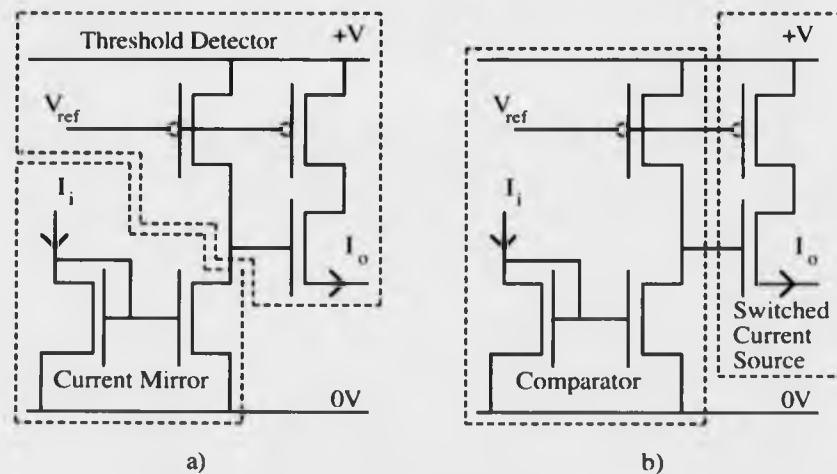


Figure 2.8. Decomposition of a current mode circuit.

Decomposition of circuits is useful at a higher level. Consider a circuit the size of a full adder like that in figure 2.7. This circuit can be decomposed into three major sections: Decoder, Logic and Encoder as is shown in figure 2.9. This will be referred to from now on as a DLE decomposition.

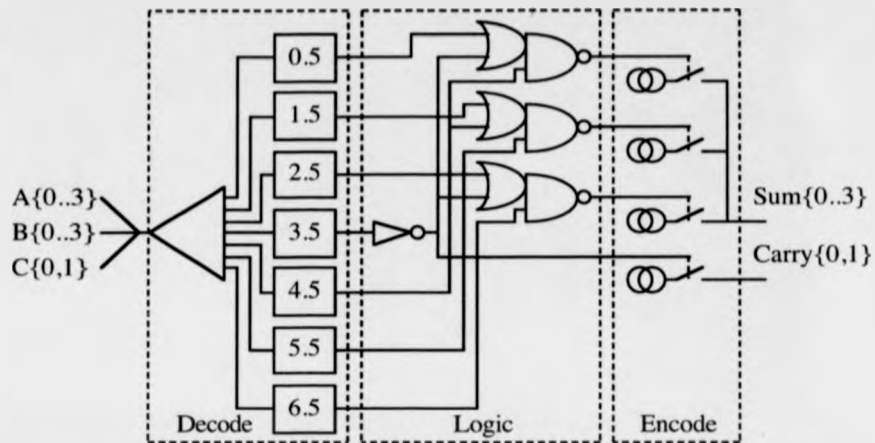


Figure 2.9. A Quaternary Full adder showing its DLE decomposition.

The inputs from the previous circuits are summed together, and decoded. These decoded signals are then passed through conventional binary logic if required, and then encoded back into current mode MVL signals.

### 2.2.7. Current mode adders

With the importance of the summing node in current mode MVL, it is natural that a full adder would be an important circuit building block. The adder described by Current et al. [Current87] which is shown in block form in figure 2.10a has already been examined. It has uni-directional inputs and outputs, and a simple decomposition into the DLE form. Kawahito et al. [Kawahito87] use a different approach as shown in figure 2.10b. The adder is used inside a multiplier which uses redundant numbers as an inherent part of its design. The numbers are represented by bi-directional currents and the adder is constructed in two distinct parts. The first part splits the sum of the two bi-directional quaternary inputs into a carry, and the intermediate sum. In the second stage, the carry from the preceding digit is added to the intermediate sum, and the result is quantised. This circuit has been implemented in CMOS with a depletion mode PMOS transistor, with very low threshold voltage variation. The first stage of this design does not quantise the intermediate sum.

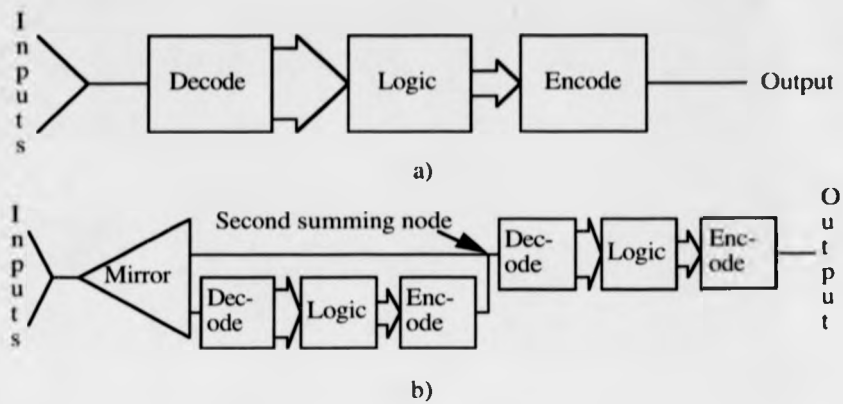


Figure 2.10. The two current mode adder types. Carry lines are omitted for clarity.

Current's adder, and the individual stages of Kawahito's adder can be seen to be made up of the same three elements: a decoder which is a set of comparators, optional binary logic, and an encoder consisting of switched current sources. These two examples show the two methods available to us for dealing with a summed input. One decoder can be used, followed by logic to determine the outputs, or a second summed node can be used (the quantiser) which sets the input into the range of the second stage.

### 2.2.8. Using current mode logic in networks

The use of current mode logic in a beneficial way relies upon the ability to utilise the wiring advantages of the current mode logic. In this section, the use of current mode logic circuits in Wallace trees [Wallace 64] is discussed. The advantages gained with these implementations can also be apparent with other algorithm implementations, but are particularly noticeable with this example, which was first pointed out to the author by Dr Steve Summerfield of Warwick University.

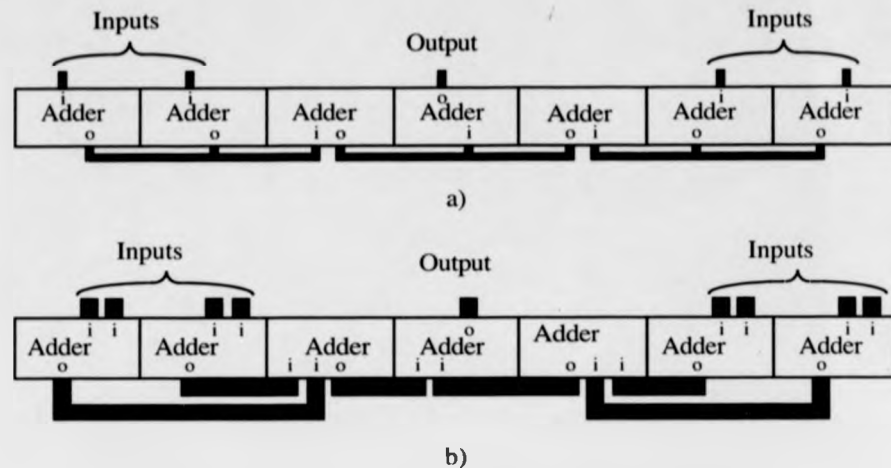


Figure 2.11. a) An MVL, and b) a binary Wallace tree structure.

Wallace tree structures are notorious for being difficult to lay out efficiently. They are not naturally placed in rectangular blocks, and routing large trees is a major problem. With current mode circuits the routing problem is greatly reduced, as it is with MVL. The reduction in current mode circuits is due to the fact that the inputs to each full adder share the same physical input connection, reducing the number of buses. There is also an MVL advantage due to the reduced number of wires in each bus. This leads to great savings when laying out structures such as these in current mode MVL. This is illustrated in figure 2.11a. Figure 2.11b is a corresponding binary voltage mode block representation. The thick lines are interconnection buses, and the 'i's show the inputs, and the 'o's show the outputs from and to the buses. The MVL buses are half as wide, as it is assumed that quaternary signals are used, and hence half the number of connections that would be required for binary. In addition, the sharing of inputs in the current mode circuit greatly reduces the interconnection difficulties. The bus area for the current mode quaternary circuit could be as little as 25% of that of the binary circuit.

### 2.2.9. Major applications of current mode MVL

There have been a number of major applications reported for current mode MVL. Major applications have been centred around the adder as it provides an obvious use for the current summing node. There are many more applications yet to be investigated. In a later chapter, other uses for the current summing node will be shown, giving a much wider scope for possible applications. This section gives a brief description of some of the more notable applications reported.

Kameyama has suggested the use of bipolar bi-directional current mode logic in digital filtering, and decimal arithmetic circuits [Kameyama80, Kameyama81]. These circuits give large potential time savings over conventional binary circuits in terms of gate delays (in the region of 80%). However, the gates for the two types of circuit are very different, and so this is not a true indication of the speed advantage. Kawahito et al. [Kawahito87] constructed a 32 by 32 bit two's complement multiplier by using bi-directional quaternary signals. Redundancy was used within the number representation to remove the ripple carry used within conventional addition schemes. The circuit used a  $2\mu\text{m}$  CMOS process enhanced with depletion mode PMOS transistors, and operated at the same speed as other multipliers available at the time. It consumed approximately 50% of the power and was about half the size of its binary equivalent. The circuit used a  $30\mu\text{A}$  unit current. Similar techniques have also been proposed for a division circuit with a speed proportional to the word length  $n$ , and a multiplier with speed proportional to  $\log n$  [Kawahito91]. These circuits were designed in a  $5\mu\text{m}$  CMOS process, and had a 40 to  $70\mu\text{A}$  unit current. K.W. Current [Current88] proposed the use of adders described earlier, and a ROM designed by Stark [Stark81] to produce a chip capable of performing Discrete Cosine Transforms (or DCTs), and Inverse DCTs (or IDCTs). This circuit offers potential savings of around 20% of the equivalent binary circuit area whilst still operating at the 10MHz pipeline clock rate required for video signals. The design was completed using a single metal  $2\mu\text{m}$  CMOS process, and a  $50\mu\text{A}$  unit current. A prototype current mode radix 4 PLA structure has been reported [Pelayo91] using a  $3\mu\text{m}$  single metal CMOS process,

and a unit current of  $10\mu\text{A}$ . Instead of the binary PLA structure of an AND plane followed by an OR plane, this circuit used a NOR plane followed by a Truncated SUM (or TSUM) plane, which gives superior results to the MIN plane, MAX plane structure used in other MVL PLAs.

### 2.3. Voltage mode MVL

Voltage mode logic is the most commonly used form of logic used for binary circuits. The relative voltage of a node indicates the value of the signal on that node. Voltage mode logic is very easily observed. With discrete circuits, a voltmeter or oscilloscope is connected to the node in question. MVL voltage mode circuits operate in the same manner, but with more voltage levels on a single node. The big advantage of voltage mode logic over other forms is its high fan-out (almost infinite for CMOS). This makes it an ideal candidate for applications where buses are used to broadcast to a number of circuits at once. Voltage mode circuits are usually ternary or quaternary. There are many more examples of ternary design with voltage mode logic than there are with current mode. To a certain extent, this can be attributed to problems with resolution.

In this section, the potential of voltage mode MVL is examined by assessing the power and time effects of using higher radices. Following this, the basic circuits used in voltage mode MVL are described. Finally, the major applications of these logic circuits are described.

#### 2.3.1. Power and time in voltage mode MVL

It is possible to analyse the power dissipation of the bus structures of a voltage mode circuit, in order to compare binary and MVL circuits. It is assumed that the circuits all operate at the same frequency, and so the energy dissipated in a cycle can be evaluated as a measure of power. With a large enough circuit, the charging and discharging of nodes will average out. This allows a fairly simple model to be used that assumes that at any particular time, an equal number of nodes is at each of the

### 2.3: Voltage Mode MVL

possible voltage levels. Further assuming that each node has the same capacitance makes the model simple, without reducing its effectiveness for large circuits. It is important to realise that this model does not include data for the circuits themselves, it simply models the power used in bus driving within the circuits. The model is a circuit containing  $n$  nodes, each with a capacitance of  $C$  farads, and the circuit is supplied with a voltage  $V$ . In the binary case, at the beginning of each clock cycle, each node can be at either logic level (i.e. either 0 or  $V$  volts), and at the end of each clock cycle, there is a 50% chance that it will have changed. The energy required to change a node is  $\frac{1}{2}CV^2$ , and so the total energy for all the nodes will be  $\frac{1}{4}nCV^2$ . For the MVL case, consider a ternary (radix 3) system. There will now be only  $n \log_3 2 = 0.63n$  nodes, as each carries more data. The voltage between adjacent levels  $V_{r3}$  is no longer the maximum voltage change on a node. Table 2.1 shows each of the voltage changes for the different starting and finishing logic levels.

		Finish level		
		0	1	2
Start level	0	0	1	2
	1	1	0	1
	2	2	1	0

Table 2.1. The change in level during a cycle for a ternary node.

There is the same possibility of each of the nine possible changes taking place, so the energy used by each of the changes can be individually calculated, and then summed:

$$total\_energy = 0.63n \left( \frac{3}{9}(0) + \frac{4}{9} \left( \frac{1}{2} CV_{r3}^2 \right) + \frac{2}{9} \left( \frac{1}{2} C(2V_{r3})^2 \right) \right)$$

$$total\_energy = 0.42nCV_{r3}^2$$

The relative values of  $V$ , and  $V_{r3}$  must now be considered. There are two realistic possibilities:

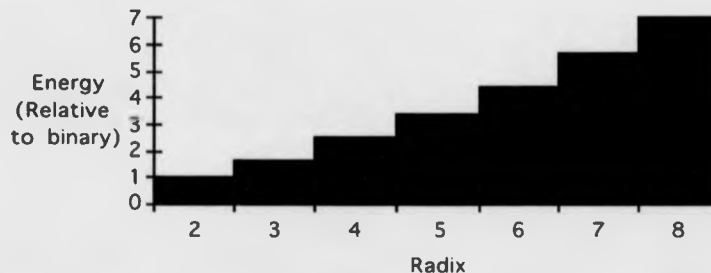
### 2.3: Voltage Mode MVL

1. The voltage difference between adjacent logic levels is the same as a binary circuit, i.e.  $V = V_{r,3}$ . This possibility assumes that the binary system has its power supply voltage as low as possible, for logic detection. This is applicable to situations where ternary is used to reduce the device pin counts. In this case, the ternary system consumes 68% more power than its binary equivalent. This causes a direct trade off between power consumption, and the chip pin count. Also, an additional power supply would be needed for the ternary circuits.
2. The power supply voltage is the same, i.e.  $V = 2V_{r,3}$ . This possibility assumes a single power supply for both binary and ternary circuits, and would be used for example when a small part of a binary circuit were to use ternary in order to gain some advantage. In this case, the energy used by the ternary system would be:

$$total\_energy = 0.1 \ln CV^2$$

For this situation, the ternary circuit will use approximately 58% less power.

The above analysis has been extended to higher radices, and the results are shown in graphs 2.2 and 2.3 for the first and second possibilities respectively. The graphs show the average energy used per cycle for radices 2 (binary) to 8 (octal).



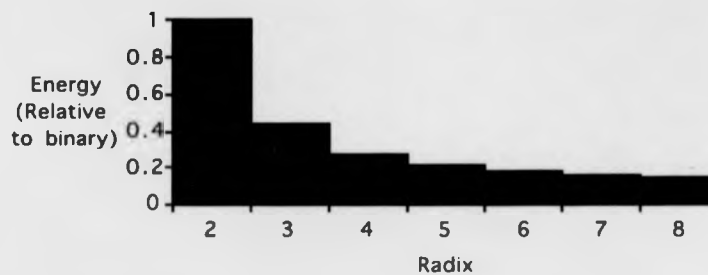
Graph 2.2. The energy used per cycle for bus structures of differing radices, but with the same voltage step between levels.

Note that the energy shown in graph 2.2 is lower than might be expected for the higher radices. It would not be unreasonable to expect the energy used to be proportional to the square of the radix, since the radix is proportional to the maximum voltage swing, and the energy is proportional to the square of the voltage swing. The



### 2.3: Voltage Mode MVL

reason that this effect is not seen is that a node rarely goes from one extreme value to another, and the rarity of this event increases in proportion to the square of the radix. This negates a very significant proportion of the increase. If the option of using the same voltage step with a higher radix is used, then the power dissipation will increase roughly in proportion to the radix, for small ( $\leq 10$ ) radices.



Graph 2.3. The energy used per cycle for bus structures of differing radices, but with the same power supply.

With the constant power supply situation depicted in graph 2.3, the effect of reduced voltage swings is markedly seen. In this case, the power supply voltage is the same, but because the majority of level changes are not the maximum, less energy is used. It can be seen from this graph, that if non-binary circuits are to be used, power reductions may be available, even if bus drivers are less efficient at that radix. It can also be seen that the advantages quickly become very small, and so radices of 3 or 4 offer the most promise. Clearly, the noise immunity of a higher radix circuit will be smaller than that of a binary circuit in this situation. This noise immunity is likely to become important when dealing with new binary circuits designed to operate at 3 volts or even lower.

The time taken by a bus line to reach its required value is also an important factor, when deciding whether MVL is practical in a particular application, or not. The time taken may be dependant on the voltage supply, but in the simplest case it is not. Assuming that the impedance of a transistor is constant, it is possible to calculate the time taken relative to  $\tau$ , the time constant of the bus and driver, for the worst case level transition, which will be a transition between the highest and lowest levels. For a '1'

### 2.3: Voltage Mode MVL

to '0' transition, in the binary case, the time delay is the time taken for the bus to reach 50% of its minimum value. For ternary and higher radices, this value is lower as shown in figure 2.12. The figure shows binary, ternary and quaternary levels, and the switching points (i.e. the voltage at which for example a '0' turns into a '1') are at the dotted lines. The arrow on each diagram shows the maximum voltage that can be considered a '0'.

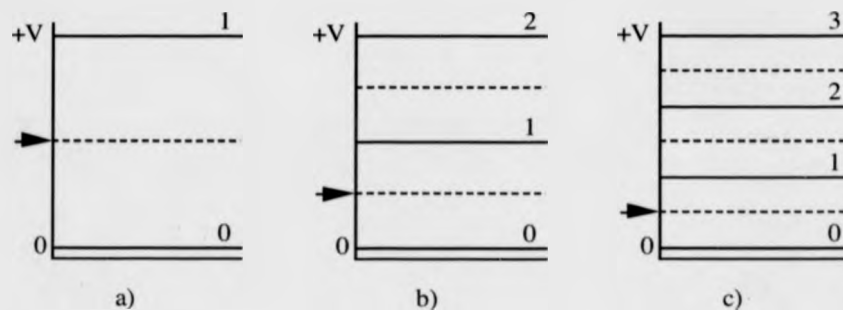
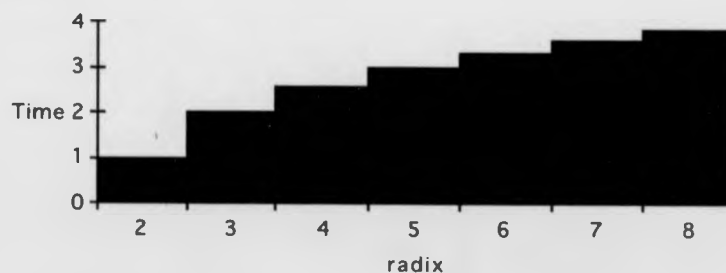


Figure 2.12. The switching points for a) binary b) ternary and c) quaternary voltage mode signals.

One effect of the lower switching voltage for a '0' with ternary and quaternary signals is that the time taken to discharge a capacitive bus line to that voltage will be longer. Graph 2.4 shows the time delay on a bus line as a function of the radix. These results are similar to those given by Baltus et. al. [Baltus90]. From this, it can be seen that the speed of a bus is nearly halved (relative to binary) when ternary is used. This is because of the smaller voltage differences between adjacent voltage levels. For this reason, binary is likely to remain the choice for conventional buses in the critical path.



Graph 2.4. bus time delay as a function of radix assuming equal CR constants.

### 2.3.2. Basic circuits in voltage mode MVL

Voltage mode circuits are used in many ways and of course, some circuits bear little or no relation to the standard basic circuits. It is still useful to examine the basic building blocks of the majority of voltage mode MVL circuits. This section does not ignore bipolar circuits, but a larger emphasis is placed on CMOS circuits. This reflects the proportion of published work on voltage mode MVL in the two technologies. There are five basic blocks from which almost all voltage mode MVL circuits are constructed from, whether they are ternary or quaternary. The blocks are:

1. The diode connected transistor.
2. The inverter.
3. The transmission switch.
4. Encoders.
5. Binary logic.

These basic building blocks are now described in more detail.

#### 2.3.2.1. The diode connected transistor.

The diode connected transistor is a resistive element formed by connecting the gate of an FET to its drain as shown in figure 2.13. This element is used extensively to produce a voltage drop between two nodes. It can also be used to tie a node to a default level, in much the same way as depletion mode devices are used in NMOS and PMOS binary circuits. In the figure, the direction of the diode effect is indicated by the dashed diode.

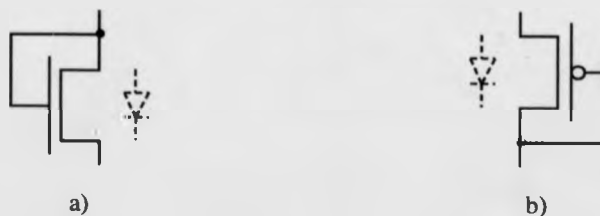


Figure 2.13. Diode connected a) NMOS, and b) PMOS transistors.

The diode connected transistor is used in other building blocks as will be seen later, and can be used on its own to produce intermediate voltage levels between the power supply voltage and ground [Mangin86].

### 2.3.2.2. The inverter.

The binary CMOS inverter shown in figure 2.14a is the most simple active binary circuit. The transfer characteristics of the circuit are typically similar to those shown in figure 2.14b. The figure shows a number of important values on the transfer characteristic. The switching voltage,  $V_{sw}$ , is the input voltage at which the output changes its value from one state to the other. The low output voltage,  $V_{ol}$ , is the voltage indicating a 'low' or '0' state. The high output voltage  $V_{oh}$  is the voltage indicating a 'high' or '1' state.

There are other important facets of figure 2.14b that should be noted. The output transition is not a perfect one, i.e. the output voltage drops away from  $V_{oh}$  before the input voltage gets to  $V_{sw}$ , and this can be an important factor in accurately restoring signals to their correct levels.

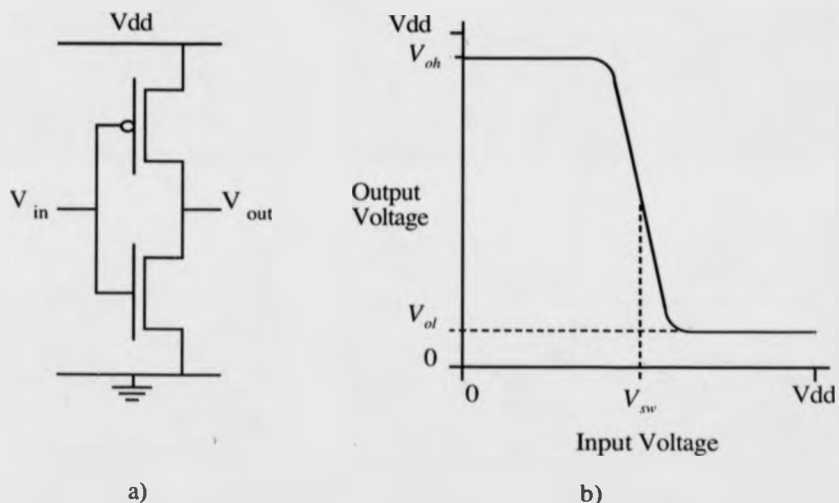


Figure 2.14. An inverter a), and b) its transfer characteristics.

The inverter is used in voltage mode MVL, but the usage is far more complex than that of the binary case. The inverter is used to distinguish between adjacent logic levels on a node, by constructing the inverter such that the switching voltage is at the switching point as shown in figure 2.12. The switching voltage can be adjusted by changing the width to length ratios of the N and P transistors relative to each other [Huertas81]. However, the ratio difference required increases exponentially as the switching voltage approaches 0, or the supply voltage. More reasonable transistor ratios can be used if a diode connected transistor is used with the inverter [Prieto88] as shown in figure 2.15a. Typical characteristics for this type of circuit are shown in figure 2.15b. The main problem with this kind of circuit is the shift in the  $V_{ol}$  (or  $V_{oh}$  if the diode connected transistor is placed between the inverter and the power supply rather than between the inverter and ground). This means that the logic levels are not exact (sometimes referred to as a logic level being 'soft'). In extreme cases this could cause an MVL circuit to detect an incorrect level. However this can be resolved by cascading inverters, if necessary.

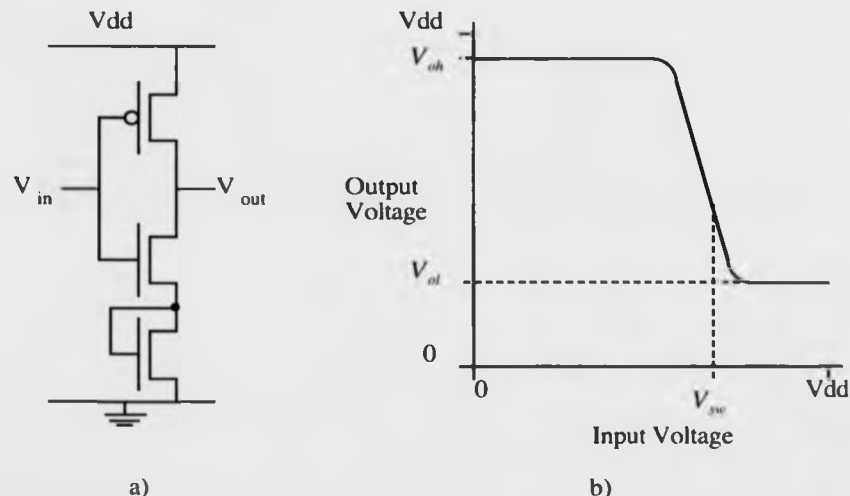


Figure 2.15. A modified inverter a), and b) its transfer characteristics.

Other modifications to the standard inverter are also possible. The supply lines normally connected to V<sub>dd</sub> and ground, can be connected to intermediate supplies

### 2.3: Voltage Mode MVL

corresponding to other signal values, or a signal itself can be used to power an inverter [Watanabe87]. This gives inverters that do not have binary outputs; they output either the upper or lower supply voltages according to the value of the input. A further modification that reverses the normal power supplies, and uses one supply as a secondary input, results in a logical sum or product circuit (a logical sum circuit is shown in figure 2.16a) [Watanabe88]. An even bigger alteration is shown in figure 2.16b. This circuit is a non-inverting buffer, that outputs the input value incremented by one (unless it is a maximum input value). This circuit has a pre-charged clock input. The circuit given in figure 2.16b, is clearly no longer an inverter, but has the same form of circuit.

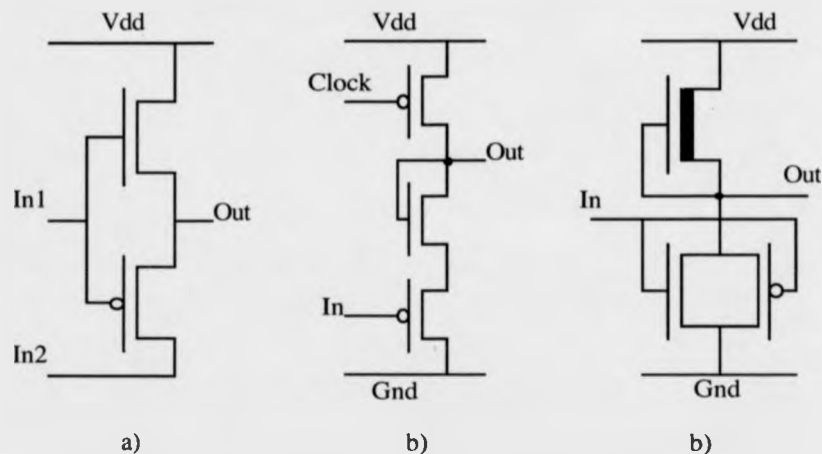


Figure 2.16. a) A logical sum, b) a level shifter and c) a permutator derived from an inverter.

Another interesting modification to the basic inverter, for a ternary NMOS style circuit is to add a P type FET in parallel with the input N type FET, and connect their inputs together as shown in figure 2.16c [Aytac87]. This yields a modulo 3 incremter or 'permutator', the truth table for which is shown in table 2.2.

Input	Output
0	1
1	2
2	0

Table 2.2. Truth table for the permutator circuit.

The inverter is generally used to drive a node to a particular voltage, as has already been discussed. However, this is not always the case. For example Schultz et al. [Schultz89] use a set of three equal strength inverters with binary inputs to drive a single node to one of four possible values corresponding to 0, 1, 2 or 3 inverter inputs being high. There is also an additional inverter connected to the node by both its input and output. This inverter reduces the voltage swing on the node, in order to make decoding easier. The circuit used by Schultz is shown in figure 2.17.

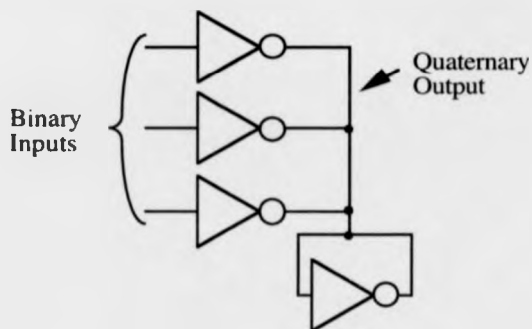


Figure 2.17. Inverter outputs ganged together to form a quaternary signal.

#### 2.3.2.3. Transmission switch.

The transmission switch is a very commonly used device in voltage mode MVL circuits. Transmission switches can be N-type, P-type or 'perfect'. The so called perfect transmission switch is an N-type and a P-type switch in parallel with each other, and their gates driven as the inverse of each other. This removes the voltage drop that sometimes occurs in N-type and P-type switches, because the voltage drop occurs under different conditions in the two switches. A 'perfect' transmission switch is shown in figure 2.18a, and its symbol is given in figure 2.18b. When E is high, and E\* is low, the resistance of the switch is low, and when E is low, and E\* is high the resistance of the switch is high.

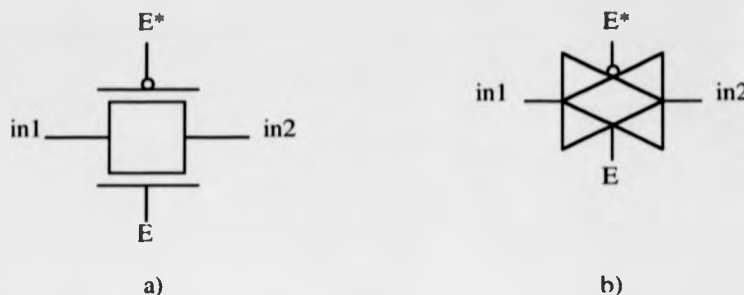


Figure 2.18. a) A 'perfect' transmission switch, and b) its logical symbol.

This switch can very easily be used to set the value of a node to a logical value when required by connecting a source of that logical value to one side of the switch, and the node to the other side [Mangin86]. The switch can then be activated, setting the node as appropriate. In the same way, the transmission switch can be used to set one node to the value of another by connecting them together [Chew87]. However, care must be taken, because the switch is not an active device, and so there is a degradation of any signal transmitted through it, and values stored on a node as charge, can be drained away in an effect known as 'charge sharing'. Wu and Prosser [Wu88, Wu89] have suggested the use of N, and P-type transmission switches that have differing threshold voltages. Multiple implants give three threshold voltage variations for each type of switch, and this allows logical functions to be formed, as the thresholds are set to differentiate between each pair of adjacent levels in a quaternary system.

#### 2.3.2.4. Encoders.

The design of encoders for ternary and quaternary voltage mode MVL is well explored. There are two major types of encoder; multiple power supply, and single power supply. The multiple power supply types simply connect the node to the appropriate power line, whilst the single power supply types use varying resistive connections to the power and ground lines to charge the node to a particular intermediate voltage. The inverter structure used by Schultz et. al. [Schultz89] has already been described. The three inputs each contribute equally to the quaternary



output. However, it is possible to combine a pair of the input inverters, and discard the fourth inverter used by Schultz, to produce an encoder of the form given in figure 2.19a [Shanbhag90]. This circuit relies on the resistive properties of the FETs, but the threshold voltage can also be put to use as shown in figure 2.19b [Singh87b]. In this circuit, the bus is precharged to 2.5 volts, and then either charged or discharged through one of the four transistors in the encoder to the appropriate level. Discharging through an N-type, or charging through a P-type FET results in the bus being fully discharged or charged respectively, whilst charging through an N-type, or discharging through a P-type FET results in an incomplete charging or discharging. Similar results have also been reported for NMOS circuits [Singh87b].

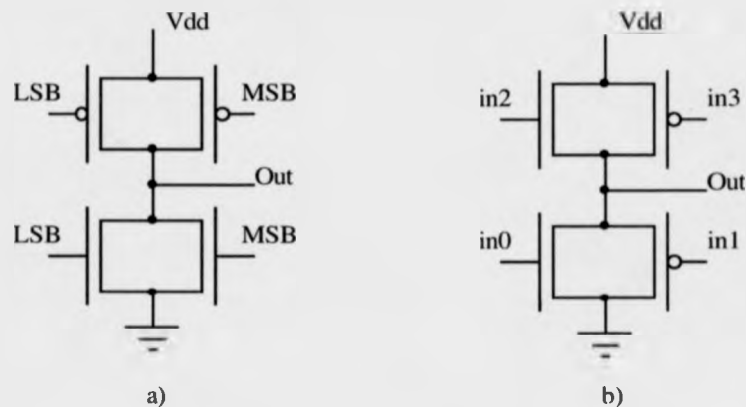


Figure 2.19. Encoders using a) resistive, and b) threshold voltage drop techniques.

The circuit used by Bhattacharya [Bhattacharya90] has many similarities to the CMOS circuits described by Singh. Singh's circuit requires the MSB and LSB to be decoded into four activation signals, one for each of the drive transistors. Bhattacharya's circuit is the equivalent of a transmission switch: either the MSB, or a weakened version is connected to the bus line according to the level of the LSB. The weakened version of the MSB is created by passing it through an inverter with swapped power supply connections. This is, in effect, the same as the N-type connected to Vdd, and the P-type connected to ground in Singh's CMOS circuit. The circuit is shown in figure

2.20. As with Singh's CMOS circuit, the bus line must be pre-charged to 2.5 volts (assuming a 5V supply) prior to operation.

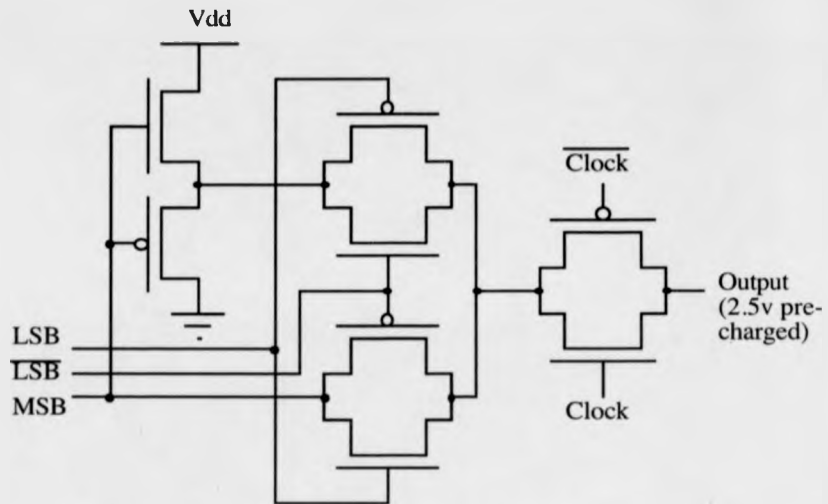


Figure 2.20. A transmission switch based quaternary encoder.

Another notable use of threshold voltage drops is shown in figure 2.21 [Etiemble90]. The a, b and c binary signals are formed from the MSB and LSB of the input, and control which one, if any, of the pull down paths is used. The pull down paths activated individually by a, b and c produce approximately 0,  $V_{tp}$ , and  $2V_{tp}$  volts respectively, where  $V_{tp}$  is the threshold voltage of a P-type FET. The pull up transistor is significantly more resistive than the N-type pull down FET. The fourth level in this quaternary encoder is obtained by not activating any of the pull down paths.

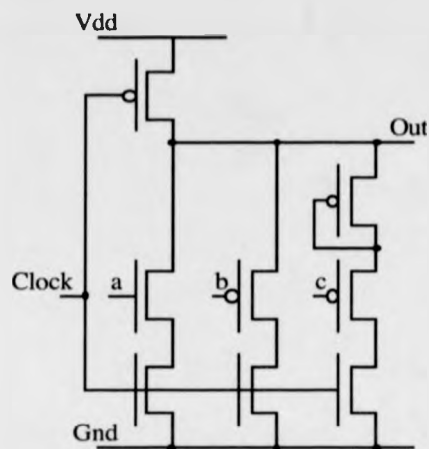


Figure 2.21. An encoder based on threshold voltage drops

Multiple power supply encoders are inherently simpler than single power supply encoders since all that is required is to switch the appropriate voltage onto the node. This can be done by the use of transmission switches [Mangin86], or binary logic followed by a very simple driver [McCluskey82, Wu90]. The use of transmission switches for Mangin's quaternary encoder is shown in schematic form in figure 2.22a. The power supplies  $V_{0-3}$  (which are the logic levels for the four logical levels) could either be external to the IC, or generated on chip as Mangin suggested. The simple encoder described by McCluskey, and Wu is shown in figure 2.22b. The a and b signals are derived from the two binary inputs required to represent a ternary signal.

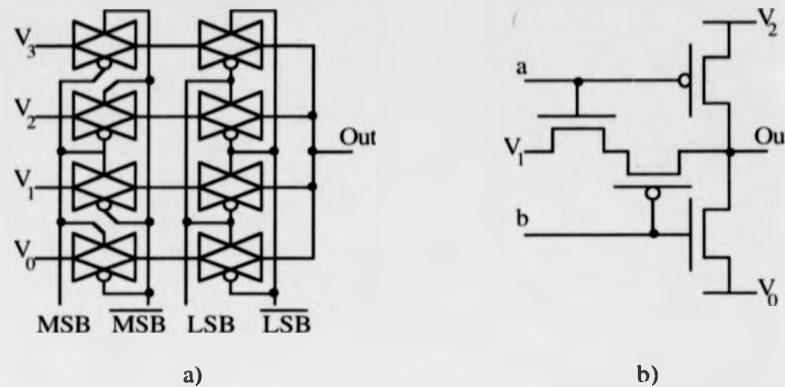


Figure 2.22. Multiple power supply encoders. a) Transmission switch based. b) A simple driver.

### 2.3.2.5. Binary Logic.

As with current mode logic, standard binary voltage mode logic is used within voltage mode MVL circuits. In some cases, though it is really the MVL that is being used inside the binary circuit, for example the binary full adder designed by Schultz [Schultz89] contains a single quaternary node. Part of the decoding of this signal is binary logic. In McCluskey's modulo 3 adder [McCluskey80], a Decode Logic Encode form similar to the current mode form described earlier can be seen.

Apart from the five major components already discussed, there have been other building blocks reported, but they have been constructed from the basic circuits that have been described. One noteworthy larger size building block in voltage mode logic is the T-gate [Chew87]. This circuit is basically the ternary version of a multiplexer: one of three inputs is fed to the output depending on the value of the ternary selection input. A transmission switch version of the circuit is shown in figure 2.23. The numbers by the inverters are the logical switching levels of the inverters. Only positive switch inputs are shown to the transmission switches, but clearly inverted switch inputs would also be applied (to the P-type FET). A binary multiplexer can be used to implement logical functions directly from their truth table [Horowitz80]. The T-gate can be used for direct implementation of ternary functions in just the same way.

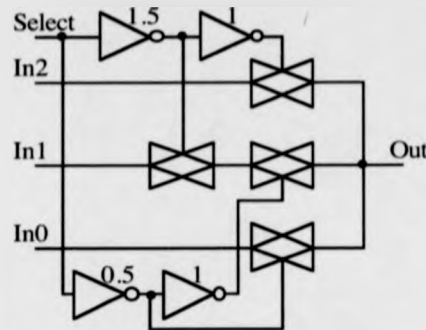


Figure 2.23. The schematic of a T-gate.

### 2.3.3. Applications of voltage mode MVL

Voltage mode MVL has a number of applications. This section examines some of these applications. It was shown earlier, that high speed buses are not practical in simple voltage mode MVL circuits, but this does not preclude its use. For example, Etiemble et. al. [Etiemble90] use a four valued bus structure for inter-processor communication in a massively parallel architecture. The limiting factor in this architecture is not bandwidth, but pin count, and so either MVL or time domain multiplexing is needed. The use of a BiCMOS process gave satisfactory speed performance, and a reduced encoder area. Other bus structures have been reported [Baltus90, Kaliman88] using differential signalling. Differential voltage mode MVL does not give advantages over conventional single ended binary, but it is better in bandwidth terms than binary differential signalling. Kaliman et. al. designed a ternary ring LAN based on a twisted pair wiring system. The three signals used are a positive, negative, and no difference between the pair of wires, which are detected using a pair of opto-electronic switches connected to the twisted pair in an opposite polarity to each other. Baltus et. al. established a method for the implementation of MVL differential signalling, by showing that a twisted set of  $n$  wires can be used instead of just a twisted pair. In this case, there would optimally be  $n$  possible levels that each wire could take. This gives simple differential interfaces (Kaliman's is not a simple differential system since no difference must also be detected) that have superior

bandwidth performance to that of a binary differential interface using the same number of wires.

Logical functions can either be directly implemented on ICs using custom logic gates designs, gate array designs, or logic arrays. In MVL, logic arrays can be created [Shanbhag90]. These logic arrays operate in quaternary, and can be used in a quaternary environment, or be used as part of a larger binary circuit, when encoders and decoders are added to the circuit. A  $2\mu\text{m}$  CMOS process was used to design this circuit. Gate arrays have been reported for the specific application area of pattern matching [Hanyu87]. Pattern matching circuits are used in Artificial Intelligence (AI) processors, as a building block. These circuits use a standard NMOS process with the addition of the choice of 4 threshold values for the transistors. This work was extended to the design of an AI processor [Hanyu88]. The multiple threshold values of the transistors were replaced by a new MOS device similar to an EPROM transistor. This device, called a FLOTOX device is a FET with an extra gate above the first gate. The lower gate is allowed to float (i.e. it is unconnected), and the amount of charge stored on the floating gate can be controlled by the upper gate. In addition, Silicon On Sapphire techniques are used to allow the back bias of the FETs to be individually controlled, and hence the threshold voltage can be electrically altered. The use of MVL in conjunction with these devices allowed a very compact design to be constructed.

Some voltage mode MVL applications are small extensions to binary circuits that improve the performance of the binary circuit. For example, Schultz et. al. presented a binary full adder, with a single MVL node [Schultz89]. The encoder part of the circuit is shown in figure 2.17, and this is decoded using 3 inverters with modified switching voltages. The sum output is then created using a small amount of logic. The whole adder circuit is shown in figure 2.24, with the numbers by the inverters indicating the logical switching level of the inverter. The advantage is seen in the carry propagation, since there are just two inverters between the carry input and carry output of the circuit. Hence, a pair of words are added more quickly than with a standard binary circuit.

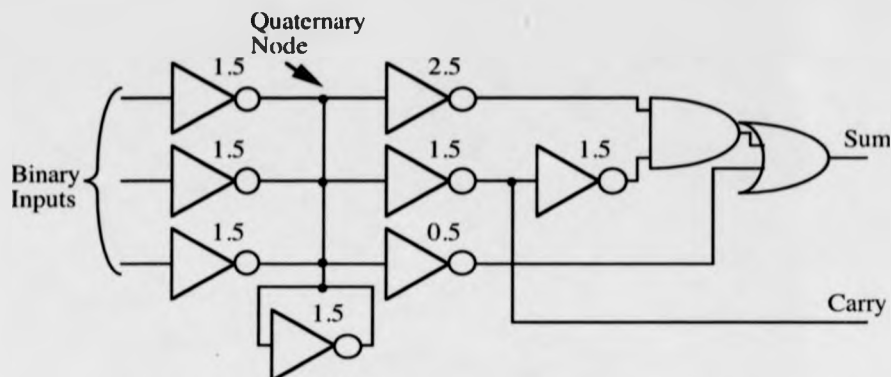


Figure 2.24. A binary adder with a quaternary internal node.

Katter et. al. [Katter90] showed how to design binary circuits such that if all inputs were set to 2.5 volts (i.e. half the supply voltage), the outputs should all be at 2.5 volts. This allows a device to be quickly tested, and the majority of failures would show up using this kind of circuit. The circuit uses depletion mode NMOS and PMOS devices which requires a number of extra processing steps during fabrication, and hence is expensive. In addition to this, extra transistors are used which in turn means lower gate densities, and higher power consumption. The disadvantages of these circuits make them impractical at present, but this is an area which shows some promise.

## 2.4. Charge mode MVL

Charge mode logic can be split into two basic types: Charge Coupled Device (CCD) logic and Switched Capacitor (SC) circuits. CCD circuits store charge in potential wells beneath the gate of an FET. Whereas SC circuits use capacitors, often created using a pair of polysilicon layers, and FETs as switches. In the next section CCD circuits are described. Following this, MVL SC circuits are examined.

### 2.4.1. CCD MVL circuits

CCDs are derived from analogue circuits known as bucket brigade delay lines. These were first integrated in the late 60's in switched capacitor form, but Boyle and

Smith [Boyle70] suggested a technique that allowed charge to be stored under transistor gates. The basis of the technique is that a single FET can have a number of gates, and a potential well can be produced beneath a gate, allowing charge storage for a significant period of time. This charge can then be moved from one well to another as shown in figure 2.25. In figure 2.25a the charge is shown in the well created by the voltage on gate A. In figure 2.25b, the voltage of gate B is made equal to that of A, so that the charge is shared between the two adjacent wells that effectively form a single large well. In figure 2.25c the voltage on A has been reduced to the minimum level removing its potential well. The minimum voltage is marked on the diagram as zero, but it would be slightly above 0 volts. Using this technique, the charge drops into the well beneath B. The charge is shown in the diagram at the bottom of the depletion region to ease understanding. However, the charge will in fact accumulate at the top of the depletion region near the gate. A three phase system has been shown here, but two phase systems can be created by varying the oxide thickness beneath the gates of the CCD [Millman87]. The effect of this is to maintain isolation of one data bit from the data on either side of it in the CCD.

MVL CCD circuits can be produced by allowing for a variable amount of charge in each potential well [Butler88]. Four logic levels is practical, and a good choice, since conversion to and from binary is simplified. Eight levels (the next practical choice) is difficult to obtain due to difficulty maintaining accurately sized charge packets. Noise is not much of a problem since it causes errors of only about 3 percent of a charge unit using quaternary logic. Transfer of charge from one well to another can be done to a high accuracy, so long as the clock is slow enough. There is a direct trade-off between clock speed and charge transfer accuracy since the well acts like a capacitor being charged through a resistor. However, this has limits, because of the 'dark current effect'.



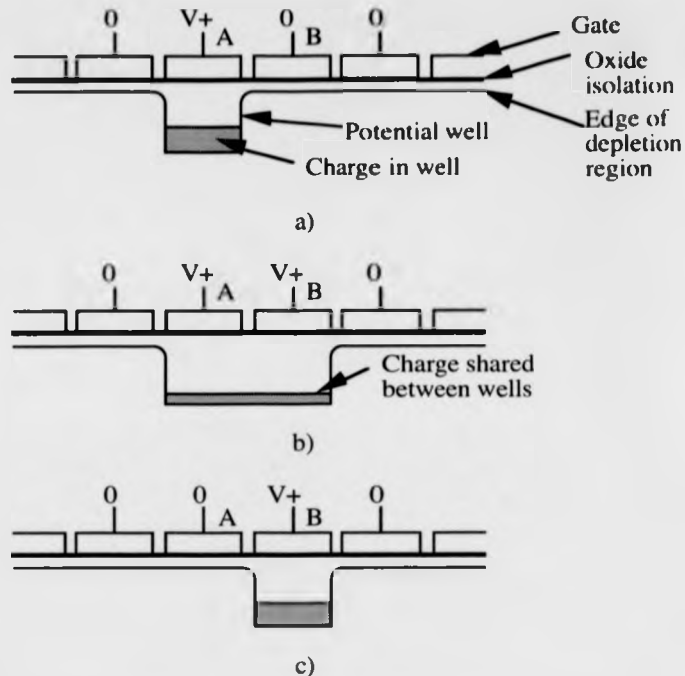


Figure 2.25. The movement of charge in CCD circuits.

Logic in CCD MVL is based upon four basic circuits. These circuits allow the construction of any logical function. They are:

1. *Constant*. A constant source is produced by supplying charge to a well of known size. The well fills to its maximum capacity, and this known amount of charge can then be passed to another circuit.
2. *Adder*. Two wells can be made to supply their contents to another single well. The contents of this well is then the sum of the initial two values.
3. *Threshold*. If more charge than a well has capacity for is supplied to a well, it will overflow, and this overflow can be used to detect that a signal has a value greater than the threshold. Composites of this cell can easily be constructed which detect a series of thresholds, and is well suited to MVL applications.
4. *Inhibit*. Instead of determining the gate potential for a well by a clock signal, as would be normal, the gate can be tied to a well in which the result of a logical

operation is being stored. This allows the path of one signal to be affected by the results of another signal path.

The basic logical circuits described above have been used to produce a number of different larger circuits. Abd-El-Barr et al. [Barr88] described how to produce efficient unary functions from these circuits, and provided more efficient results than had previously been obtained for more than three quarters of all unary functions. This work has been extended [Barr89, Barr90] to the production of dynamically reprogrammable PLA structures. In addition, a method for the decomposition of a function in order to allow direct implementation on a single row of the PLA structure was given. A radix independent method for the synthesis of functions has been shown [Zaky90]. This method directly yields circuit implementations that are heavily based on the efficient overflow circuit. CCDs can also be used within fairly small binary circuits to reduce circuit complexity. For example, Manzoul et. al. [Manzoul87] suggested its use in binary adders. The use of a radix 4 CCD carry look-ahead array reduces the number of arrays needed from three to one for an 8 bit adder. The savings available vary with word size, and there are regions where there is no gain from this type of circuit [Manzoul88].

#### 2.4.2. Switched capacitor MVL circuits

Switched capacitor circuits have been designed for use in MVL applications. A switched capacitor circuit is in fact a hybrid of charge and voltage mode circuits. This is because the charge is held on capacitors which have well defined capacitances, and so the voltage across the capacitor is also well defined. Whilst the signal value is determined by the amount of charge on the node, the output of a circuit can be a specific voltage, which is converted to a charge by the input stage of the next circuit. This is shown in figure 2.26. where the output voltage  $V$  is fed via switch A to a capacitor. The charge  $Q$  on the input node of the device is:

$$Q = \frac{V}{C}$$

This happens during the  $\phi_1$  clock period, and during the  $\phi_2$  clock period, the value of that charge is connected to the rest of the circuit (although its sign is changed).

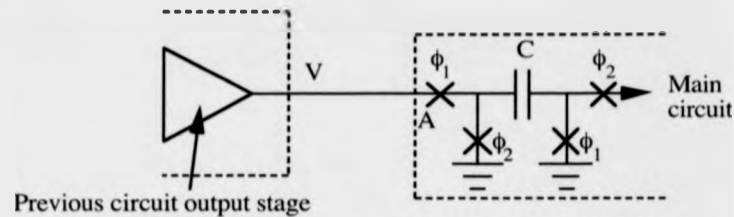


Figure 2.26. The input of a switched capacitor circuit.

There are four main building blocks in switched capacitor circuits [Ho89]. These circuits are all extremely similar, but each has a different clocking procedure. The circuits contain an operational amplifier, so that charge levels are easily maintained. This fact would make this type of circuit ideal for ASIC designs. The building blocks are:

1. *Gain stage*. This circuit outputs a fixed multiple of its input. This is done using ratioed transistors.
2. *Adder*. The charge stored on two capacitors is transferred to a single capacitor. This gives the sum of the inputs.
3. *Integrator*. A single input adder, that is not cleared at the beginning of each cycle. This clearly has limits, as the circuit has a limited range. Overflow (or underflow) causes the integrator simply to remain at its maximum (or minimum) output value.
4. *Comparator*. The circuit outputs an indication of which input is greater in value.

Ho and Smith [Ho89] suggested the use of these building blocks in the production of a fixed point multiplier that is topologically base independent. This is attractive, since if high radices become practical, they will enhance this design with little need for modification. It has also been shown that these types of circuit would be useful in the production of binary to MVL, and MVL to binary converters that could cope with a number of different radices [Ueno89]. This is a particularly attractive proposition if non  $2^i$  radices are available since then, a single converter could be used for the different digits in a system residue number arithmetic.

## 2.5. MVL memory

To a large extent so far, memory circuits have been ignored. The design of large scale binary memory circuits has become very different from conventional binary logic design, and this is also true of MVL memory circuitry which has followed the advances made in binary memory circuits. The design of multiple valued memories has been examined by others [Rich86, Etienne92]. This section gives a brief overview of MVL memory technology. Memory is examined in two parts: register memory, and large scale memory.

### 2.5.1. Register memory

Register memory consists of latches, flip-flops and similar circuits that are generally used in sets of a few digits. The two main types of latch are shown in figure 2.27. Both are current mode circuits. Figure 2.27a is the block diagram of a basic latch, in which a quantising circuit produces two outputs, one of which can be switched to be its input. With the switch in position A, the output is a quantised version of the circuit input, but in position B, the last input is retained indefinitely. This switch can be created in a number of ways. K.W. Current et al. [Current89, Current91] switches the voltage on a diode connected transistor since this will remain stable during the transition of the circuit from its sample state to its hold state. This system is used for both unidirectional and bi-directional circuits. Xu's quantiser [Xu89] feeds back a positive, and negative version of the signal that cancel each other out. When the input is disconnected, so is the negative quantised signal, and the positive signal is latched. Figure 2.27b shows a different system. The input is fed to the gate of a diode connected transistor during one clock phase, and then during another clock phase, the transistor gate, and the capacitance connected to it, is disconnected from the drain. The drain is now connected to the output so that the transistor acts like the output stage of a current mirror. Lee et. al. [Lee92] showed this circuit used in conjunction with a quantising circuit to provide a latch. More recently, it

has been suggested [Karasawa93], that MOSFETs with a staircase like transfer function could be used to form an MVL latch.

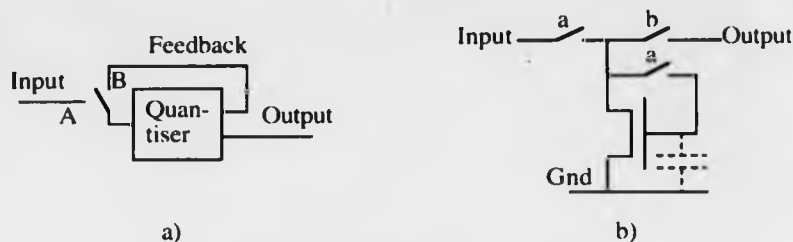


Figure 2.27. Two current mode latch circuits.

Flip-flops are not well defined in MVL. For example, considering the binary JK flip-flop, it is not obvious what the ternary extension should be, indeed even the number of inputs is unclear. Wu and Prosser have suggested a ternary three input 'JKL Flip-Flop' [Wu88], but this gives the circuit an unnecessarily high 27 possible functions in comparison to the four possible with the binary JK type. Even the two input ternary JK type of Webb et. al. [Webb91] does not use all of its nine possible functions uniquely. The construction of these types of flip flop is usually done from the basic voltage and current mode building blocks described earlier [Etiemble80].

### 2.5.2. Large scale MVL memories

The use of MVL in ROM circuits is an area in which the commercial potential of MVL has already been shown. INTEL used a four-valued ROM in the 8087 maths co-processor, and in other devices [Stark81, Silio83]. The benefit of this circuit over a binary equivalent was a reduction in area. This was due to the reduction in the number of cells required in the ROM. Although the four valued cells were larger than their binary counterparts, they were not twice as large, and more significantly, there needed to be half as many gaps between cells to prevent wires shorting. INTELs approach was to use a number of different widths of transistors in the ROM cells, but it would also be possible to use multiple supply rails to tie the ROM transistors to [Cho88], or multiple implants to alter the transistor characteristics [Rich86].

Large scale memories that can be written to have had some successes in MVL. The most notable of these is the 4Mbit dynamic RAM constructed from a 1Mbit RAM with modified input and output buffers [Horiguchi88]. One of 16 levels is stored on each memory cell capacitor. This circuit is set up for file storage applications, and is ideal as a store in between main memory, and disk storage, that greatly reduces the effect of "thrashing" [Dube90]. Static RAMs have also been suggested [Ishizuka90], using a number of transistors with different threshold voltages. It has been shown that just two threshold voltages can give a radix 4 memory cell. Another writeable memory type is the Content Addressable Memory (CAM) which can be used in artificial intelligence applications, and caches. An efficient MVL CAM can be designed using floating gate MOS transistors [Hanyu90], and more recently efficient circuits have been reported using just conventional MOS transistors [Aragaki92]. In this case, both time and area improvements resulted from the MVL design. As with binary circuits, novel technologies sometimes lend themselves particularly well to the design of memory circuits in MVL. The floating gate MOS transistor has already been shown, and another important example is the Resonant Tunnelling Diode (RTD). The RTD is a two pin device which has a sawtooth I-V characteristic as shown in figure 2.28a. When combined with a resistor in series, an I-V characteristic results that has multiple hysteresis loops in it such as that shown in figure 2.28b, and can thus be used as a multiple valued storage element [Wei91, Shieh93].

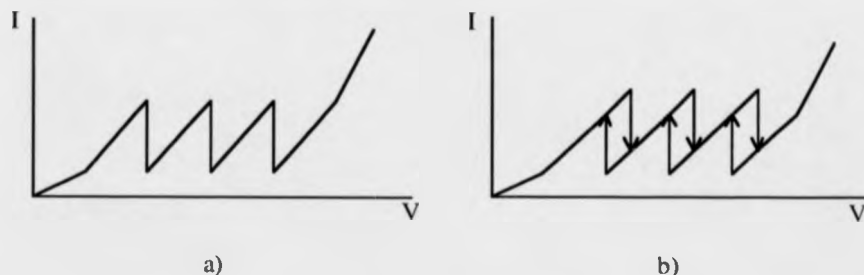


Figure 2.28 I-V characteristics of a) and RTD b) an RTD in series with a resistor.

## 2.6. Testing MVL circuits

In an earlier section it has been shown, that MVL can be used to test binary circuitry. However the testing of MVL circuits has not been addressed so far. Testing is a very large subject, and a full investigation of the subject would be excessively lengthy in this context. Instead, a brief insight into the testing of MVL circuits is given. The testing of MVL circuits can be split into two major areas: the provision of circuits to enable testing, and the generation of test vectors to apply to the circuits. Both of these areas are now described.

### 2.6.1. Circuits for MVL testability

MVL test circuits have closely followed their binary counterparts, and so concepts like scan testing are well known in the MVL community. It is possible to construct Built In Logic Block Observation (BILBO) circuits for MVL [Rozon88, Rozon90]. And the area overhead for scan testing can be greatly reduced by using ternary clock signals [Hu86, Wu93]. These overhead reductions can easily be applied to binary circuits as well, by introducing a ternary clock into the circuit. Another well known binary technique, is to use a Multiple Input Shift Register (MISR) to compress the test output data, in order that a small amount of data can be used to check the operability of a circuit. One drawback of MISRs is that there is a small possibility of a fault masking itself (known as aliasing), so that a faulty circuit gives a correct MISR output. MISRs can be applied to MVL circuits [Edirisooriya92], with an added advantage, that the aliasing probability is lower for the MVL MISR. Cellular arrays are a quick and easy way to design logic circuits, and can be designed for ease of testability [Kamiura92]. Indeed, this has been extended to provide a cellular array that is also diagnosable and repairable [Kamiura93]. Clearly, this level of fault tolerance could be a big advantage in certain applications.

### 2.6.2. MVL test vector generation

The generation of test vectors is a big problem. Methods for stuck at x detection have been proposed [Whitney88] using a method known as 'partial differences'. In addition, test sets have been produced for a fairly standard set of voltage mode ternary building blocks [Rozon91]. These can be used to construct test vectors for any circuit constructed from the building blocks. Test pattern generation has also been devised for PLA structures in MVL [Nagata93], and these patterns can also detect the position of some faults, giving the potential for fault correction. At a high level, ternary systems can be made fault tolerant by applying methods shown by Xu et. al. [Xu88].

### 2.7. MVL and VLSI production processes

The design of VLSI circuits cannot be totally devoid of consideration for the process in which it is to be manufactured. In this section, the interaction between MVL and the process it is manufactured in is examined.

It is possible in binary logic design to settle on a process type, for example CMOS, and then to design to a typical set of design rules. When the time comes for fabrication, the circuit can then be scaled such that all the rules used in the layout of the circuit obey the rules actually specified for the process. For a number of years, this was a perfectly satisfactory way of working, as the relative sizes of most of the components of a VLSI circuit remained roughly the same. However, as circuit dimensions have become smaller and smaller, the dimensions of components such as contacts between metal layers have not kept pace with other circuit components such as wire widths. This means that older circuits are less efficient than they might be in a new process. In addition to these problems, the smallness of modern integrated FETs has brought new effects to a level at which they must be considered. This is not much of a problem for binary circuits, but for MVL it can have a profound effect. This section investigates the effect of the trends in VLSI production processes on the use of MVL.



There have been many papers published that describe analogue circuits, and converters that have accuracies equivalent to eight to ten bits, that operate at very reasonable speeds. For example Nakamura et. al. [Nakamura91] produced a digital to analogue converter with 10 bit accuracy, operating at 70MSamples/s. Because of this, it would not be unreasonable to assume that MVL circuits should be capable of operating with as many as 1000 levels. The reason that MVL circuits don't operate at these levels is mainly due to the production process. There are many parameters in the production process, such as doping densities, and oxide thickness, which affect the parameters of the transistors produced. These parameters vary across the device like the example shown in figure 2.29. To minimise the effect of this variation, transistors whose parameters must be matched, need to have a common centroid [Shyu84, Bastiaansen91]. In small circuits, this can be done by using circuits such as that shown in figure 2.30. This diagram shows two transistors in series, but they have been split into a pair of parallel transistors each to give a common centroid for the composite transistors [Nakamura91].

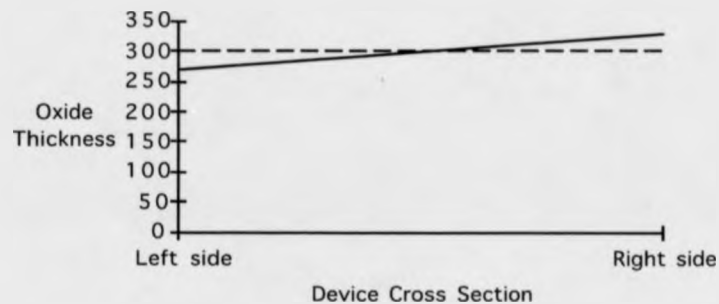


Figure 2.29. A slowly varying device parameter (Oxide thickness in Angstroms).

Other parameters such as threshold voltage, and edge effects are not slowly varying, and so other techniques such as unit transistor layout, and the use of active mirrors rather than simple passive mirrors need to be employed [Nairn88]. It may also be possible to alter the power supply voltage to reduce the threshold voltage effects [Yamakawa86].

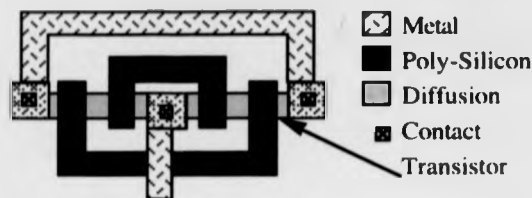


Figure 2.30. A VLSI layout of a pair of transistors in series, with common centroid.

The most used technique for reducing errors in analogue circuits is simply that of not using minimum size transistors. Transistor sizes in the region of  $10\mu\text{m}$  by  $10\mu\text{m}$  are not at all uncommon, and as a result, analogue circuits make much less of the advantages available with modern sub-micron semi-conductor processes. As the processes shrink, other effects can cause problems. For example, when the gate length becomes small, the proximity of the source and drain doped areas to each other means that the field under the gate is affected, altering the transistors parameters. This is the 'short channel effect' [Viswanathan85]. MVL circuits attempt to compete with binary circuits in terms of time and area, and so circuits that use  $10\mu\text{m}$  long gates, and are therefore large and contain high parasitic capacitances are not particularly practical. Instead, large numbers of MVL designers seek to alter the process the circuits will be fabricated in, by introducing additional devices such as high precision depletion mode PMOS transistors [Kameyama88], or multiple threshold voltages [Hanyu87]. Despite these novel device types, commercial circuits would need to be able to operate over wide temperature ranges which can affect the threshold voltage. However it is interesting to note that this temperature dependence reduces as the gate length gets shorter [Tzou85], and so it is likely to be less important for MVL circuits than for analogue circuits, where it is a major problem [Sansen88]. There is already the knowledge available to MVL designers to allow many more levels in their circuits. Advances in analogue technology may in the future make larger numbers of levels a practical option, or may increase the yield of existing designs to a stage where they are commercially viable.

## 2.8. MVL and optical technology

MVL does not necessarily have to be implemented in an electronic circuit. An example of another method of performing logical operations is Optical logic. Optical logic is a relatively new area of research, based upon communicating signals as beams of light rather than as electrical signals. There are many advantages to be gained from this approach:

1. Negligible crosstalk.
2. High bit rates due to the lack of an equivalent to electrical parasitic capacitances.
3. Electrical isolation between the ends of the signal connection, reducing the ground loop problem.
4. Low noise in an enclosed system.
5. A single connection can carry a number of independent signals at different frequencies.
6. A connection off the integrated circuit does not need to be at the edge of the circuit, and hence the pad ring problem is removed.

Many of these advantages were examined by Hurst, when he examined the possibility of using optical fibres for MVL communication [Hurst80]. Hurst recognised the opportunity to use optical signals in a similar manner to that described by Kirchoff's current law. A set of fibres carrying signals fused and then fed to a single output provides an output light intensity equal to the sum of the inputs. The logic associated with optical technology either uses a single input connection with inputs at different frequencies [An92], or a threshold function based upon the intensity of the input sources, and the required function from the gate [Osawa92], in much the same way a current mode logic operates as described earlier. From this, it can be seen that optical technology is potentially well suited to MVL use. In addition to this, current mode circuit concepts should also be appropriate for optical MVL circuits.

## 2.9 MVL versus Binary logic

There has already been some discussion in this chapter on the relative merits of MVL and binary logic. Voltage mode circuits have been examined, and it has been shown that bandwidth increases that would be expected from these MVL circuits will not actually appear in practice. Current mode logic has not been directly compared to binary logic simply because the comparison is so heavily affected by the choice of the current unit, and by the fabrication process. The higher the current unit, the higher the speed, and the higher the power dissipation. The power dissipation is static however, and so simply lowering the clock frequency of a circuit will not affect power dissipation. The fabrication process has a great effect on the performance of these circuits, as the variability of parameters such as the threshold voltage affects the choice of current unit, and hence power and speed. There are, however practical results from which conclusions can be drawn. For example, the multiplier designed by Kawahito et al. [Kawahito87] has shown that power and area advantages are available when current mode MVL is used. Consequently, current mode logic appears to be a good choice for further development.

## 2.10. Summary

In this chapter, the methods used in designing, fabricating, and testing MVL circuitry have been examined. As the circuits will always have to interact with binary circuitry to some extent, converters to and from conventional voltage mode binary logic is important. This causes radices that are integer powers of two to be favoured. The only radix that is in common use which does not fall into this category is radix 3. This radix has been used extensively as a 'first step' away from binary. MVL concepts have been applied to many different logic types, and fabrication technologies, with varying degrees of success. The use of a four valued ROM in the Intel 8087 maths co-processor [Stark81] proves that MVL can be commercially viable. However, trends in CMOS fabrication processes mean that it is becoming increasingly difficult to design efficient MVL circuits. This is especially true of voltage mode MVL, since the

## 2.10: Summary

proposed reduced voltage power supplies would give even smaller ranges for the circuits to operate within. Newer devices such as RTDs could well give rise to much more efficient MVL circuits, due to their non-binary nature. New technologies such as optical logic may also benefit from the use of MVL. In the near future, it seems unlikely that MVL circuits will gain wide acceptance, but this is probably more to do with the fact that the whole of digital circuit design and production is tuned to the needs of binary than anything else.

# 3

## Elementary Function Evaluation.

### 3.1. Introduction

This chapter will look at the evaluation of elementary functions such as sines cosines tangents, and natural logarithms. These functions are non-linear, and generally are functions of just one variable. Combinations of these functions are also examined, and it will be seen that these combinations can be as quick to calculate as the individual functions themselves. The motivation for implementing such functions is clear. There are many operations taken for granted in computing today that use functions of single variables very heavily. Frequency analysis (using the Fourier transform), Givens' rotations, basic geometry, and rotating an object on screen using a CAD package are obvious examples. As a consequence scientific calculators invariably have these functions, and any software language of a higher level than an assembler would be incomplete without them. The example of the Fourier transform shows why these functions are needed in hardware. To take the Fourier transform of a 512 pixel by 512 pixel image for example, it would be necessary to calculate more than 4 million vector rotations (conventionally done using multiplication by stored sine and cosine values). A single vector rotator would have to complete one calculation every 10ns to achieve television rate throughput.

The different methods of evaluating these functions fall into four categories: Non-iterative, Quadratic convergence, Linear convergence, and Hybrid. Non-iterative methods are table lookups in general and produce very high speed results, but the cost quickly becomes prohibitive as word length increases. Quadratic convergence is obtained from Taylor series expansions [Farmwald81], and similar methods. These methods rely heavily on multipliers and dividers, which are themselves complex functions in VLSI. Linear convergence is found in algorithms such as the CORDIC algorithm which provides roughly one bit of accuracy per iteration of the algorithm. Hybrid techniques take the best of a number of these individual techniques, and combine them to form more efficient solutions.

This chapter starts with a brief look at elementary functions. This gives us information on what an elementary function evaluation algorithm will be required to produce. Following this, non-iterative, quadratic, and linear convergence algorithms will be examined. The CORDIC algorithm which has linear convergence is examined in detail. Hybrid algorithms are mentioned in these sections where appropriate.

### 3.2. Nature of the problem

Functions such as sine, log and square root are inherently non-linear, and hence their range of convergence is important, as the output can be non-linear, discontinuous, complex, and can even go to infinity. An example of this is the tangent of an angle as shown in figure 3.1. We can see that this function is non-linear, discontinuous, and goes to infinity. This function also has the property that it is repetitive all the way to  $\pm\infty$ . Some other functions, for example the square-root of a number (also shown in figure 3.1), are non-repetitive, and also in some ranges ( $<0$  for a square-root or a logarithm) the function is not a real number (i.e. it is complex).

Repetitive function algorithms need only to be able to calculate a large enough range of the function to be able to reproduce the rest of the function. In the case of the tangent, the range 0 to  $\frac{\pi}{2}$  is sufficient. However, for the square-root, the range required would be the entire range of positive numbers representable with the data type

used in the algorithm. Some algorithms such as the CORDIC algorithm cannot cope with the full range in their basic form, and so work has to be done to increase the range in which the algorithm operates [Hu91].

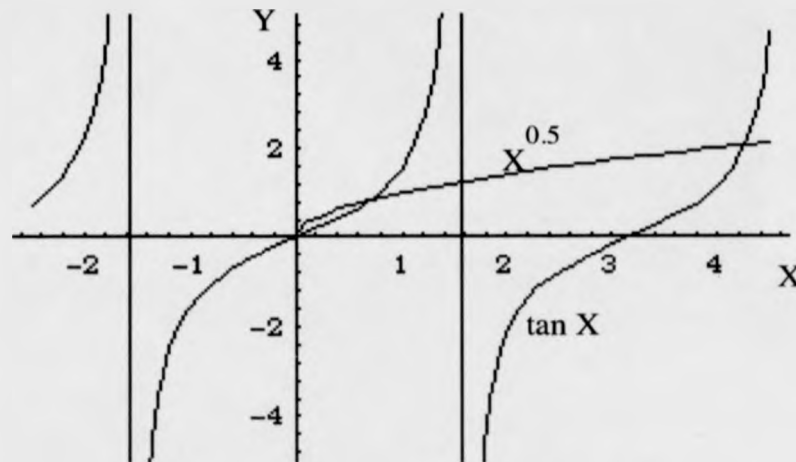


Figure 3.1. The tangent, and square-root of a number.

### 3.3. Non-iterative algorithms

One simple way to evaluate functions quickly is to pre-calculate the function for all possible input values, and store the results in a Read Only Memory (ROM). The input indicates the function to be evaluated, and the value that the function is to be evaluated for. The ROM acts as a look up table, and the output is the stored result for the input value and function. The problem with this, is that every possible input value and function have to have their output stored separately. For word lengths less than 8 bits, this doesn't matter, but as the number of bits rises, the size of the ROM quickly becomes impractical. For each new bit on the input, there are twice as many input combinations, and one more output bit required. This more than doubles the area used for each additional bit. In general

$$\text{Number\_of\_functions} \times 2^{\text{Input\_Word\_Length}} \times \text{Output\_Word\_Length}$$



bits of storage are needed for fixed point numbers. This method is fastest, but is only usable for small word lengths. With floating point numbers, the problem is vastly increased as the input word length effectively becomes

$$\text{Input\_Mantissa\_Length} + 2^{\text{Input\_Exponent\_Length}}$$

Clearly, a repetitive function can reduce the memory requirement by a significant factor, but this quickly becomes less significant as word length increases.

### 3.4. Algorithms with quadratic convergence

Another system that can be used for function evaluation calculates a series expansion of the function using the Taylor series [Farmwald81]. The Taylor series for various functions are shown below. There is a heavy requirement for multiplication and division. A 64 bit result would require about 20 multiplications or divisions. These would have to be done in series, and consequently, this method is quite slow.

$$\cos \theta = \sum_{n=0}^{\infty} (-1)^n \frac{\theta^{2n}}{(2n)!}$$

$$\sin \theta = \sum_{n=0}^{\infty} (-1)^n \frac{\theta^{2n+1}}{(2n+1)!}$$

$$\cosh \theta = \sum_{n=0}^{\infty} \frac{\theta^{2n}}{(2n)!}$$

$$\sinh \theta = \sum_{n=0}^{\infty} \frac{\theta^{2n+1}}{(2n+1)!}$$

$$\log_e \left( \frac{1+x}{1-x} \right) = 2 \sum_{n=1}^{\infty} \frac{x^{2n-1}}{2n-1} \quad (|z| < 1)$$

$$e^z = \sum_{n=0}^{\infty} \frac{z^n}{n!}$$

Implementations of the Taylor series, combined with a table look up for the first few bits have been shown [Farmwald81, Ohhashi85], as a very effective hybrid.

### 3.4: Algorithms with quadratic convergence

The result is a 32 bit floating point function generator that can calculate the square-root, the reciprocal, and the reciprocal of the square-root. The circuit requires approximately 30K bytes of ROM, two multipliers, and two adders, as shown in figure 3.2. There is an 8% error between the LSB value calculated by the circuit, and the same function calculated on a VAX.

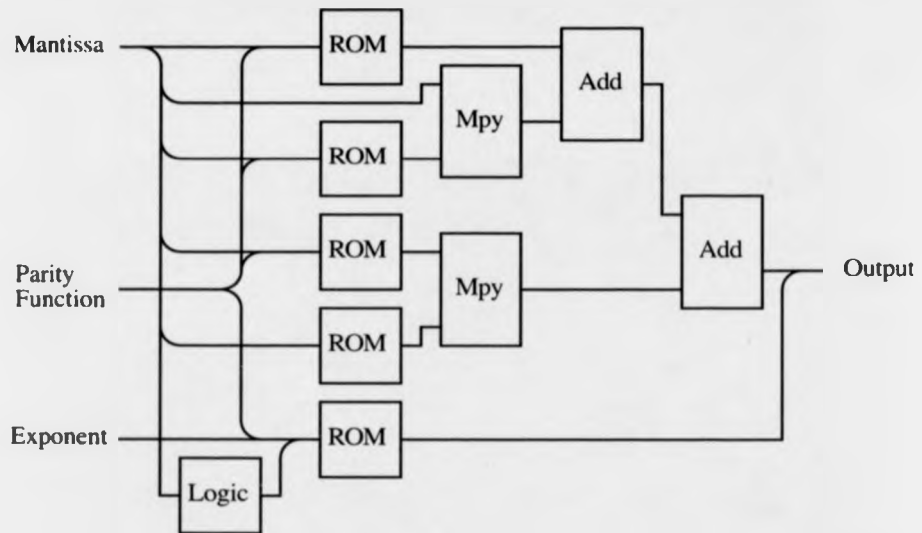


Figure 3.2. A hybrid ROM-Taylor Series Circuit [Ohhashi85].

### 3.5. Algorithms with linear convergence

Another way to calculate functions of a variable, is to use digit by digit approximation algorithms such as those proposed by Specker [Specker65], or Volder [Volder59], or by using chebyshev polynomials [Hwang87]. These algorithms need more iterations to get a result to the same accuracy as the Taylor series, but the iterations can be considerably faster, due to the reduced complexity of the calculation. The following section will examine the most popular digit by digit algorithm; the CORDIC algorithm as proposed by Volder. This algorithm has been the subject of much discussion due to it's simplicity of design, and flexibility [Fulcher89,

Dewilde92]. Following this radix 4 and 16 algorithms with similarities to the CORDIC algorithm will be examined [Rodrigues81, Ercegovac73].

### 3.5.1. The CORDIC algorithm

#### 3.5.1.1. Origins

The COordinate Rotation Digital Computer or CORDIC was first suggested in 1959 by Jack E. Volder [Volder59]. His motivation was a need for accurate calculations on board an aircraft for navigation. The system was capable of quickly computing vector rotations, and performing Cartesian to circular coordinate system conversions. The same hardware was also able to multiply, divide, and convert between binary and mixed-radix number systems.

CORDIC is based on reducing a rotation into a set of smaller sub-rotations. The sum of these sub-rotations is the overall rotation required. If a vector  $R$  is considered, that is to be rotated through an angle  $A$ . If  $|A|$  is greater than  $2\pi$ , then rotating by the  $A$  modulo  $2\pi$  will have the same effect. The sub-rotations used in an iteration is greater than  $\frac{1}{2}$  the sub-rotation used in the previous stage, so the maximum rotation is at least  $\pm 2 \times \text{first\_sub\_rotation\_angle}$

#### 3.5.1.2. Basic CORDIC algorithm

Rotating a vector is a fairly complex operation involving multiplications, which is best avoided if possible. One way to do this is to perform a pseudo-rotation which simply moves the end of the vector along a perpendicular to the original vector. The effect is to rotate the vector through an angle say  $\alpha$ , and extend it a little. This extension is ignored for now, but it will be discussed again later. The vector  $(X_0, Y_0)$  becomes  $(X_1, Y_1)$  where

$$X_1 = X_0 - Y_0 \tan \alpha$$

and

$$Y_1 = Y_0 + X_0 \tan \alpha$$

this is shown in figure 3.3. We do a number of such rotations by fixed, and carefully chosen angles, which combine to form the correct angle of rotation. We can make

$\tan \alpha$  equal to 1 for the first iteration of the algorithm,  $\frac{1}{2}$  for the second,  $\frac{1}{4}$  for the third and so on. For the  $(i-1)$ th iteration,  $\tan \alpha_i$  will be  $2^{-i}$ . The pseudo rotation rotates the vector by exactly the angle  $\alpha_i$ , but note from the diagram that it also extends it. We will ignore this extension for the moment, so that the iteration is simply:

$$X_{i+1} = X_i \pm \partial_i Y_i$$

and

$$Y_{i+1} = Y_i \mp \partial_i X_i$$

where

$$\partial_i = 2^{-i}$$

The  $\pm$  simply indicates the fact that the rotation can be in either direction.

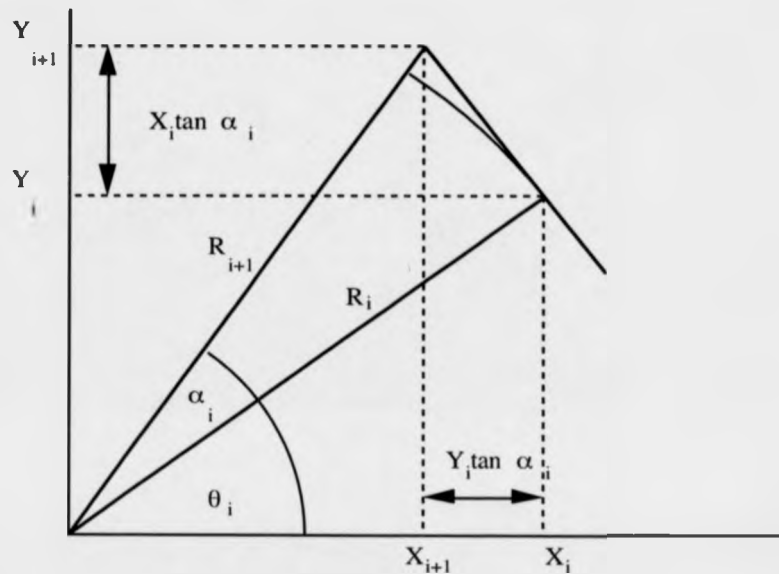


Figure 3.3. A CORDIC pseudo-rotation.

Clearly, it is important to keep track of how much the vector has been rotated. For this purpose, another register, conventionally the Z register, is used. A pre-stored value for  $\alpha_i$  is either added to or subtracted from the Z register at each iteration:

$$Z_{i+1} = Z_i \pm \alpha_i$$

To start the process, it is necessary first, to consider what is required of the CORDIC machine. The functions fall into two categories called ROTATION, and VECTORING. ROTATION starts with a vector, and rotates it through an angle. To do this, the X and Y coordinates of the vector are loaded into the X and Y registers, and the angle through which the vector is to be rotated is loaded into Z. For each iteration, Z is forced towards 0 by doing a positive rotation if Z is negative, and a negative rotation if Z is positive. In VECTORING mode, Z starts equal to 0, and X and Y are loaded with the coordinates of the vector as before. This time, at each iteration Y must tend to zero, so a positive rotation is performed if Y is negative, and a negative rotation if Y is positive. The result, is X as the vector magnitude, and Z as the vector argument (angle).

As has been seen, the rotations performed on the CORDIC iterations are not perfect rotations, but rotation extensions. For a rotation of  $\tan^{-1} 2^{-i}$ , both the X and Y values are extended by a factor of  $\sqrt{1 + 2^{-2i}}$ . These extensions combine to form an overall extension or scale factor which is conventionally called K. The problem of the scaling factor is dealt with in more detail later in this chapter.

### 3.5.1.3. Error and Convergence

As a set of CORDIC iterations is performed, two things must be ensured, first that the values in the registers do not become greater than the register can handle (i.e. it overflows), or that an accumulation of rounding errors causes the final result to be in error. For this reason, extra bits are required at the top and bottom of the word to guard against errors of this kind. In addition, it is necessary to ensure that enough iterations of the algorithm are performed in order for the output to be accurate to the desired number of bits.

Walther suggested that, for an  $n$  bit result,  $n + \log_2 n$  bits are needed in the registers to maintain accuracy [Walther71]. His argument was that each iteration could produce an error no larger than the least significant bit of the word. The sum of the errors could therefore not affect any more than the lowest  $\log_2 n$  bits of the word.

However, in 1992, Y.H. Hu developed error bounds for the CORDIC algorithm that show that more bits are required than were previously thought [Hu92]. These bounds included Walther's extension [Walther71] which will be mentioned later in this chapter. The errors are made up from two areas, the approximation error, and the rounding error. For the purposes of this discussion, it will be assumed that a  $b$  bit result is required after  $n$  iterations. The  $b$  bits are made up of 1 sign bit, and  $b-1$  fractional bits.

The approximation error is the difference between the required rotation, and the rotation actually performed by the algorithm. It is known that:

$$\tan^{-1} 2^{-i} \geq \alpha_{i+1} \geq \frac{1}{2} \tan^{-1} 2^{-i}$$

when  $i$  is in the range of interest ( $\leq 0$ ). So long as the required rotation angle  $z_0$  is within range, i.e.

$$|z_0| \leq \sum_{i=0}^{n-1} \alpha_i,$$

the remaining rotation  $z_i$  after any iteration  $i$  will be given by:

$$|z_i| = |z_{i-1} - \alpha_{i-1}| \leq \sum_{j=i}^{n-1} \alpha_j$$

Hence the approximation error  $e_{approx} (= z_n)$  is given by:

$$e_{approx} \leq \alpha_{n-1}$$

In other words, the error approximation is smaller than the angle the vector was last rotated through. This effect is shown in figure 3.4. In this example the angle rotated through is zero (for the sake of simplicity in the example). The first rotation is in a positive direction, and all the following rotations are in the negative direction. The approximation error is the difference between the first iteration shown, and the sum of all subsequent iterations. This difference is smaller than (or equal to) the last rotation performed.

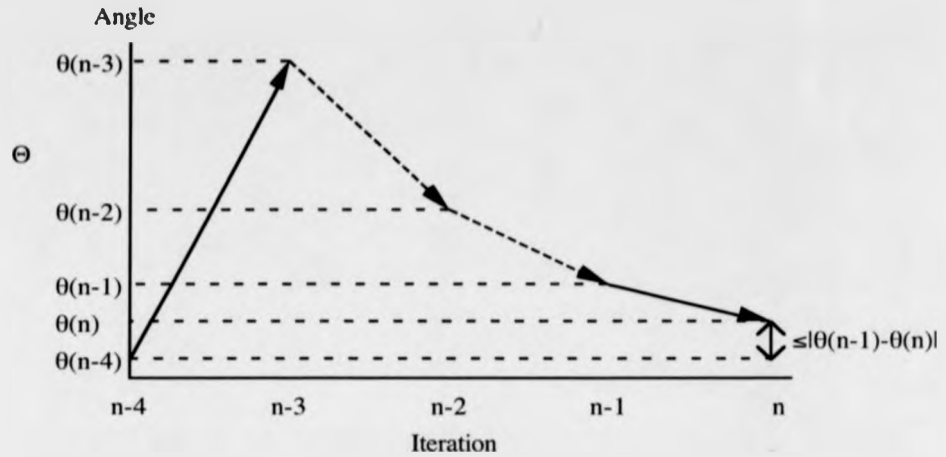


Figure 3.4. The error in a rotation using a set of sub-rotations as in the CORDIC algorithm.

The rounding error is the error caused by the finite length of the words used in the iteration. At each iteration, additive quantisation noise is introduced into  $x_i$  and  $y_i$ . In a  $b$  bit signed fractional number, this error  $e_{i(x)}$ , and  $e_{i(y)}$  is given by:

$$|e_{i(x)}| \leq 2^{-b}, \text{ and } |e_{i(y)}| \leq 2^{-b}$$

We can consider a single variable  $e_i$  to be the worst case of  $e_{i(x)}$ , and  $e_{i(y)}$ . In this case, the final error is the sum of the individual errors propagated to the output through subsequent iterations. The overall error in either  $x_n$  or  $y_n$ ,  $e_{total}$  will be:

$$|e_{total}| \leq |e_n| + \sum_{i=1}^{n-1} \left( |e_i| \cdot \prod_{j=i}^{n-1} 1 + 2^{-j} \right)$$

However, because  $x$  decreases as  $y$  increases, and vice-versa, the combination of the two errors is not as large as this. If the distance between the end of the vector, and the point at which the end of the vector should be is referred to as  $E$ , then  $E$  will be the magnitude of the combination of the error in the  $x$  direction, and the error in the  $y$  direction:

$$E = \sqrt{2e_{total}^2}$$

$$\text{where: } |e_{total}| \leq |e_n| + \sum_{i=1}^{n-1} \left( |e_i| \cdot \prod_{j=i}^{n-1} \sqrt{1 + 2^{-2j}} \right)$$

If the vector is pre-scaled, then there is nothing more to do, and so the error is unchanged. However, if the vector is post-scaled, then the error is scaled with it. When both errors are evaluated together, it is possible to build tables for the internal accuracy, and number of iterations relative to the number of bits of output accuracy. The results show that there is a trade off between internal accuracy, and number of iterations. An acceptable balance seems to be, for an  $b_i$  bit internal accuracy:

$$b_i \approx b + \log_2 b + 2, \text{ and}$$

$$n \approx b + 2$$

Considering overflow, if two fractional numbers are input, then if both  $x$ , and  $y$  inputs are maximum (i.e. just less than 1), then the vector length is maximum. This can be rotated to make a maximum  $x$  by rotating the vector onto the  $x$  axis. This will give a maximum value of less than  $\sqrt{2}$ . This must be increased by the scale factor, so that the maximum result is approximately 2.3. Hence two guard bits are needed, and a sign bit at the top of the word.

#### 3.5.1.4. Walther's extension

In 1971, Walther proposed an extension to the CORDIC algorithm to allow the computation of hyperbolic sines, and cosines [Walther71]. He also showed how to form other functions using multiple passes of the basic CORDIC system. The basic idea behind the extension proposed by Walther was that changing the sign of the addition/subtraction in the equation for  $x$ , changes the coordinate system. Also, if there is no addition/subtraction to  $x$ , a linear system is produced. Walther introduced a new factor called  $m$  which can take the values (1,0,-1). The CORDIC equations now become:

$$X_{i+1} = X_i \pm m \delta_i Y_i$$

$$Y_{i+1} = Y_i \mp \delta_i X_i$$

$$Z_{i+1} = Z_i \pm \alpha_i$$



For the circular system,  $m=1$ , and  $\alpha_i = \tan^{-1} \delta_i$ . It can be seen that this makes the equations the same as for the simple CORDIC described earlier. For multiplication and division,  $m=0$ , and  $\alpha_i = \delta_i$  so that the iterations become a shift and add of  $X$  to  $Y$ : a serial multiply. The hyperbolic functions are formed by setting  $m$  to  $-1$ , and  $\alpha_i = \tanh^{-1} \delta_i$ . This set of equations can be arrived at by analysing the hyperbolic case in the same way as the circular case was derived earlier in this section. The scale factor for these extended equations is:

$$\sqrt{1 + m\delta_i^2}$$

for an iteration with a shift of  $\delta_i$ . The iteration sequence is slightly different for each coordinate system. For the circular system  $i$  increments from zero, for the linear system  $i$  increments from 1, and for the hyperbolic system,  $i$  increments from 1, but some iterations are repeated. The repeated iterations are  $k, 3k+1, 3(3k+1)+1$ , and so on with  $k$  starting at 4. The reason for these repeats, is to maintain convergence for the algorithm. The algorithm does not converge for  $m=-1$  since:

$$\alpha_i - \sum_{j=i+1}^{n-1} \alpha_j < \alpha_{n-1} \text{ no longer holds.}$$

However, the following equation does still hold:

$$\alpha_i - \left( \sum_{j=i+1}^{n-1} \alpha_j \right) - \alpha_{3i+1} < \alpha_{n-1}$$

So by repeating a small number of iterations, the hyperbolic rotation can be made to converge. This means that  $\delta_i$  is no longer simply  $2^{-i}$ , instead:

$$\delta_{i+1} = \frac{\delta_i}{2} \text{ if this iteration doesn't need to be repeated}$$

$$\delta_{i+1} = \delta_i \text{ otherwise.}$$

These extra rotations add very little overhead to the computation, as only four extra iterations would be needed for a 128 bit result (4,13,40, and 121).

Apart from the functions that can be obtained directly from the CORDIC system, it is possible to combine results to form other functions. The most useful of these functions are given below:

$$\tan \theta = \frac{\sin \theta}{\cos \theta}$$

$$\tanh \theta = \frac{\sinh \theta}{\cosh \theta}$$

$$e^\theta = \sinh \theta + \cosh \theta$$

$$\log_e \theta = 2 \tanh^{-1} \left( \frac{y}{x} \right) \text{ where } x = \theta + 1 \text{ and } y = \theta - 1$$

$$\sqrt{\theta} = \sqrt{x^2 - y^2} \text{ where } x = \theta + \frac{1}{4} \text{ and } y = \theta - \frac{1}{4}$$

### 3.5.1.5. Hybrid Taylor series and CORDIC

In 1982 Ahmed suggested a combination of CORDIC, and the Taylor series for vector rotation [Ahmed82]. The first term of the Taylor series for sine  $Z$  is simply  $Z$ , and it is possible to perform the last of our CORDIC iterations using this approximation, rather than  $\partial_1$ . Hence after the last CORDIC iteration, a final iteration, as follows is performed:

$$X_{m+1} = X_m - Z_m Y_m$$

$$Y_{m+1} = Y_m + Z_m X_m$$

It has been shown by Ahmed that only  $m \left( = \frac{n+1}{2} \right)$  (where  $n$  is the number of bits of accuracy required in the output) iterations of the CORDIC algorithm are needed followed by the final iteration as given above. Timmermann et al. extended this work to include vectoring mode [Timmermann89a]. In this case  $\frac{n}{3} + 0.472$  iterations of the CORDIC algorithm are used followed by the iteration:

$$Z_{m+1} = Z_m + \frac{Y_m}{X_m}$$

However, to calculate  $X_{m+1}$ ,  $m$  CORDIC iterations must be performed, at which point:

$$X_{m+1} = X_m$$

The advantage of these techniques is that they speed up the calculation of the function, but multiplication and division are now required in the final iteration.

### 3.5.2. Range extension

It is easy to see that the range of convergence of the CORDIC algorithm is an important factor. In the linear case ( $m=0$ ) the range of convergence, is:

$$\left| \frac{X_{in}}{Y_{in}} \right| \leq 1 \text{ for vectoring, and}$$

$$|Z_{in}| \leq 1 \text{ for rotation with the shift sequence } 1, 2, 3, \dots$$

We can increase the range of convergence by using a sequence  $-M, -M+1, \dots, 0, 1, 2, \dots$

In this case, the range of convergence is:

$$\left| \frac{X_{in}}{Y_{in}} \right| \leq 2^{M+1} \text{ for vectoring, and}$$

$$|Z_{in}| \leq 2^{M+1} \text{ for rotation.}$$

It is important to remember that this will increase the number of bits required at the top of the word to stop it from overflowing.  $M+1$  extra bits will be needed.

With cyclic functions such as sine or tan, all that is really needed is enough of the function to be able to calculate the rest of the function simply. 0 to  $\frac{\pi}{2}$  is sufficient for the given examples, but, as Hu pointed out [Hu91] if the angle representation given by Daggett [Daggett59] is used, then a range of  $\pm\pi$  is useful since no conversion needs to be done at all on the angle data. The angle representation uses a scaled two's complement number as shown in figure 3.5, so that the range of a value  $a$  in this representation is given by:

$$-\pi < a \leq \pi$$

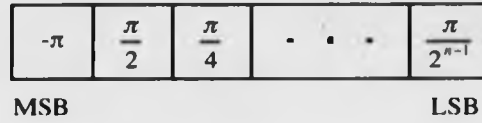


Figure 3.5. Daggett's number representation.

An obvious way to increase the range to this level in the circular case would be to start the iteration sequence  $-2, -1, 0, 1, 2, \dots$  instead of  $0, 1, 2, 3, \dots$ , and this would increase the range to roughly  $\pm \frac{4}{3}\pi$  which is fine. However, this necessitates right shifts, which will increase errors greatly, and the range  $\pm\pi$  can also be obtained using the sequence  $0, 0, 0, 1, 2, 3, \dots$  as well, which does not have the undesirable right shifts.

The hyperbolic case is non-trivial for two reasons. Firstly, the hyperbolic functions are not cyclic so that a very large range is desirable for a general purpose function evaluator, and secondly the rotation angle is  $\tanh^{-1} 2^{-i}$  for the hyperbolic case, and for  $i \leq 0$ ,  $\tanh^{-1} 2^{-i}$  is a complex number. This can be solved by performing a slightly more complicated rotation of  $\tanh^{-1}(1 - 2^{-2^{-(i+1)}})$  when  $i \leq 0$ . In this case no extra iterations are required when  $i \leq 0$  since:

$$\frac{\tanh^{-1}(1 - 2^{-2^{-(i+1)}})}{\tanh^{-1}(1 - 2^{-2^{-(i+1)+1}})} \leq 2$$

The range of convergence for these rotations is:

$$\theta_{\max} = \sum_{i=-M}^0 \tanh^{-1}(1 - 2^{-2^{-(i+1)}}) + \left[ \tanh^{-1}(2^{-N}) + \sum_{i=1}^N \tanh^{-1}(2^{-i}) \right]$$

$M$  determines the range, and is the maximum negative value of  $i$ .  $N$  determines the accuracy of the calculation, and in total there are  $M+N+1$  iterations. The range increases sharply, but care must be taken to prevent overflow since the range increase also implies an increase in the maximum values in the registers. However, as the scaling factor is less than 1 for the hyperbolic case, there is potential for large errors since the vector lengths are reduced by the scaling factor, and the errors will increase with the vectors when they are post-scaled. The scale factor is now:

$$K = \left[ \prod_{i=-M}^0 \sqrt{1 - (1 - 2^{-2^{-i+1}})^2} \right] \left[ \prod_{i=1}^N \sqrt{1 - (2^{-i})^2} \right]$$

This scale factor can very quickly become very small ( $k < 10^{-4}$  for  $M=4$ ). Clearly, the smallest possible range should be used, as the higher the value of  $M$ , the larger the number of extra bits will be required at the low end of the word to protect against errors.

### 3.5.3. Scale Factor Compensation

Previously, it has been assumed that scale factor correction would be done only by the obvious pre- or post-scaling using a multiplier or divider. This is a reasonable option which has been used [Steer77, Dixon90] but a number of methods have been developed to compensate for the scale factor within the algorithm itself. Despain developed a technique called the 'Compensated CORDIC Method' [Despain74]. The basis of this method was a magnitude correction at each step. The basic iteration as described by Volder is now modified to:

$$X_{i+1} = X_i + b_i X_i \partial_i + a_i Y_i \partial_i$$

$$Y_{i+1} = Y_i + b_i Y_i \partial_i - a_i X_i \partial_i$$

Where  $b_i \in \{0,1\}$  indicates the sign of the rotation correction.  $b_i$  is not allowed to be -1 since this seriously affects convergence.  $a_i$  indicates the direction of rotation.  $b_i$  can be pre-calculated. The only problem with this is that the angle rotated is no longer  $\tan^{-1} \partial_i$ , and convergence is not assured. A small number of iterations must be repeated. Despain wished to calculate 32 bit numbers, and reported that only shifts of 8 and 21 bits needed to be repeated for this purpose. This was found by a heuristic search.

Haviland and Tuszynski [Haviland80] used a standard CORDIC iteration to produce the intermediate results  $X_{i+1}^*$ , and  $Y_{i+1}^*$ . These become the final result of the iteration normally, but if a scaling factor iteration is required the result is found by:

$$X_{i+1} = X_{i+1}^* (1 \pm \partial_i)$$

$$Y_{i+1} = Y_{i+1}^* (1 \pm \partial_i)$$

To ensure convergence in the hyperbolic case, double iterations of the CORDIC iteration given by Volder are sometimes required, as has already been discussed. To compensate for both the scale factor and hyperbolic non-convergence, 11 extra iterations are needed for a 24 bit output.

As Ahmed pointed out, it is possible to perform a scaling and CORDIC operation in parallel. This leads to the iteration:

$$X_{i+1} = X_i + ma_i Y_i \partial_i + b_i X_i \partial_i + b_i ma_i Y_i \partial_i^2$$

$$Y_{i+1} = Y_i - a_i X_i \partial_i + b_i Y_i \partial_i - b_i a_i X_i \partial_i^2$$

There is clearly a small time penalty from having more values to add together, and more circuit area would be required. However, convergence is assured with no extra iterations than the algorithm as extended by Walther. Despite the fact that there are more terms, these equations should take no longer than the equations given by Despain since a Wallace tree of adders requires two rows of adders for three or four terms.

#### 3.5.4. Floating point CORDIC processors

Floating point numbers are widely used, especially in high level languages to aid programming. Floating point numbers simply take away from the programmer the need to constantly check that a value is within range, and shift it accordingly. The use of CORDIC in a floating point environment has been discussed many times for general application, and specific applications [Chown90a, Chown90b, Walther71, Harding91, Steer77]. The floating point CORDIC processors can be split into three constituent parts: floating point pre-processing, floating point post-processing, and the integer CORDIC module. This is shown in figure 3.6.

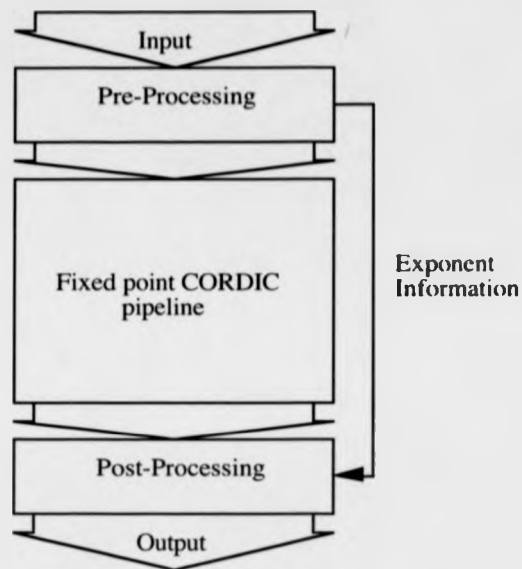


Figure 3.6. A floating point CORDIC processor.

#### 3.5.4.1. Floating point pre and post processing

The basic idea behind a floating point CORDIC processor is that the input floating point  $x$  and  $y$  values are normalised relative to each other, so the most significant bit of the mantissa in both  $x$  and  $y$  registers has the same significance. The pre-processor does this, and gives input exponent information to the output post-processor, which combines this information with the mantissas it has re-normalised. This pre- and post-processing seems fine at first sight, but it introduces errors that the user must be aware of. Consider the vector  $(0.5 \times 2^{10}, 0.5 \times 2^{-15})$ , or written in full in binary  $(1000000000, 0.0000000000000001)$ . This vector is easily represented by two floating point numbers with 24 bit mantissas, and 8 bit exponents, as is the angle between them  $(0.5 \times 2^{-24} \text{ rad})$ . However as the angle representation is still fixed point in these floating point CORDIC processors, the  $0.5 \times 2^{-15}$  would be lost if a 24 bit fixed point representation was used for the pipeline. It is possible to make the rotation angle a floating point number as has been shown by Hekstra et. al. [Hekstra93].

### 3.5.5. Redundant CORDIC processors

Redundant number systems can be of great advantage to CORDIC processors, but they introduce some interesting problems. A single shift and add iteration can be done in time  $O(1)$  rather than in time  $O(n)$  for an  $n$  bit word which should speed up the algorithm greatly. However, the sign of  $Z$  (the angle register) for rotation, and the sign of  $Y$  (the  $Y$  component register) for vectoring must be evaluated at each iteration to decide on the direction of the next rotation. The evaluation of sign is an  $O(\log_2 n)$  problem for redundant numbers, since the sign of the word is the sign of the first non zero bit, which could be anywhere in the word. Hence all bits of the word must be tested. It would clearly be possible to create a tree structure of cells to evaluate the sign, and this would have a delay of time  $O(\log_2 n)$ . The structure is shown in figures 3.7 a), and b).

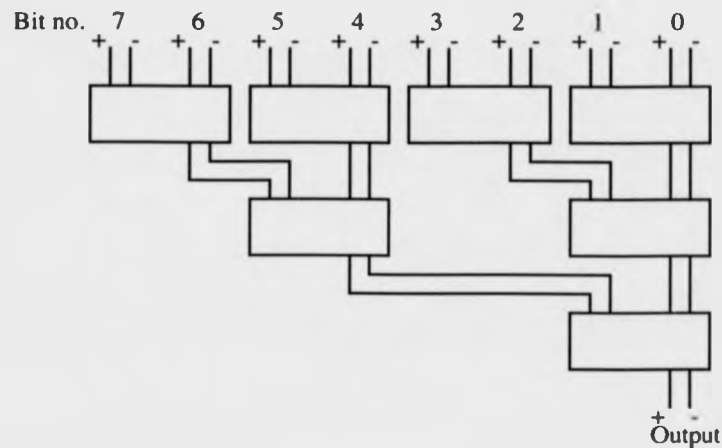


Figure 3.7. a) A sign evaluation circuit for SBNR



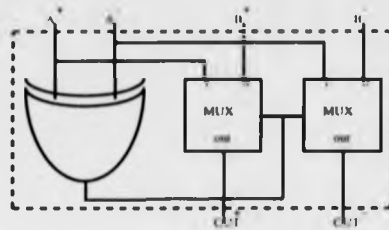


Figure 3.7. b) Cell details for a sign evaluation circuit.

A	A*	A <sup>-</sup>
-1	0	1
0	0	0
0	1	1
1	1	0

### 3.5.5.1. The Variable Scaling Factor Method

There are a number of ways to get round the problem of sign detection, to stop us from having to evaluate the sign of the word at each stage. The first was suggested by Ercegovac and Lang [Ercegovac90], and gets around the problem by not doing a rotation if the sign of the register is not apparent from the first two bits that can contain non-zero data. Note that these two digits are the most significant two digits for the first iteration, and shift down by one digit at each iteration. The only problem with this system is that the scale factor  $K$  is no longer a constant.  $K$  is given by:

$$K = \prod_{i=0}^{n-1} \sqrt{1 + a_i 2^{-2i}}$$

where  $a_i \in \{1, -1\}$  in the conventional CORDIC algorithm, and represents the decision made at each stage of which direction to rotate the vector. In Ercegovac's redundant implementation,  $a_i \in \{1, 0, -1\}$ , as no rotation takes place in certain iterations. The method used to calculate  $K$  is first to calculate a value  $P$  where:

$$P = \prod_{i=0}^{n-1} (1 + |a_i| 2^{-2i})$$

This can be done by a simple iteration of the form:

$$P_{i+1} = P_i + |a_i| 2^{-2i} P_i$$

Following this,  $K$  is calculated by taking the square root of  $P$ . Since Ercegovac's implementation is on-line, an on-line square root algorithm is used. The scale factor must be removed by division after completion of the standard CORDIC iterations.

Techniques for removal of the scale factor during the CORDIC iterations are of no use here since  $K$  is not known until some time after the last iteration.

### 3.5.5.2. The Double Rotation Method

In 1991, Takagi, Asada and Yajima [Takagi91] proposed two methods for the use of redundant numbers in the CORDIC algorithm that give a constant scale factor. The methods are closely related to each other. They are the Correcting Rotation method, and the Double Rotation method. The Correcting Rotation method is a generalisation of the Double Rotation method.

In the Double Rotation method, each iteration is made a little more complicated by making the rotation from a pair of smaller sub-rotations. If the two sub-rotations are in the same direction, the overall result is a rotation in that direction, but if the sub-rotations are in different directions to each other, then no rotation of the vector occurs. However, in this case, the vector extension does occur, and so the scale factor is constant. The effect in the angle register is shown in figure 3.8.

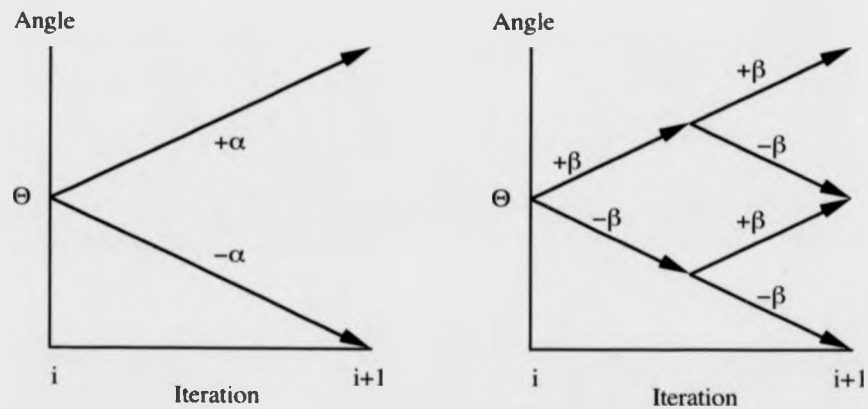


Figure 3.8. The double rotation method.

The equations for the iteration are shown below:

$$X_{i+1} = X_i - q_i 2^{-i-1} Y_i - p_i 2^{-2i-4} X_i$$

$$Y_{i+1} = Y_i + q_i 2^{-i-1} X_i - p_i 2^{-2i-4} Y_i$$

$$Z_{i+1} = Z_i - q_i 2 \tan^{-1} 2^{-i-2}$$

$q_i$  and  $p_i$  are composites of the two rotations, and are calculated directly from the angle register  $Z$ . Three digits of  $Z_i$  are used at each iteration as shown below:

$$(q_i, p_i) = \begin{cases} (-1, 1) & \text{if } [z_i^i z_i^{i+1} z_i^{i+2}] < 0 \\ (0, -1) & \text{if } [z_i^i z_i^{i+1} z_i^{i+2}] = 0 \\ (1, 1) & \text{if } [z_i^i z_i^{i+1} z_i^{i+2}] > 0 \end{cases}$$

where  $z_i^b$  is the  $b$ th fractional bit of  $Z_i$ .

This method increases the complexity of the iteration, but keeps the number of iterations to  $n$  for an  $n$  bit result. The selection of  $q_i$  and  $p_i$  is simple, requiring only a small constant number of digits. The complexity of this algorithm is  $O(n)$ .

### 3.5.5.3. The Correcting Rotation Method

The Correcting Rotation method uses the fact that the extra rotation done at each iteration is not strictly necessary. Provided enough bits of the angle register are examined, a double iteration every few cycles is sufficient. An example of this is shown in figure 3.9 with a double rotation every three cycles.

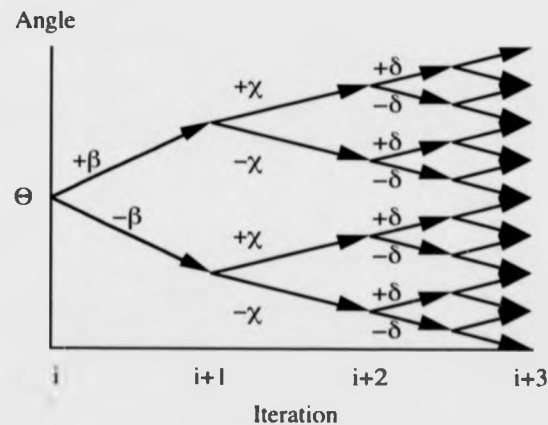


Figure 3.9. The correcting rotation method.

The diagram is somewhat simplified by the assumption that:

$$\tan^{-1} 2^{-i-1} = \frac{1}{2} \tan^{-1} 2^{-i}$$

In other words, it is assumed that the angle rotated through at each iteration is half that of the previous rotation. This assumption is reasonable, and becomes more accurate as  $i$  increases. A correcting rotation is performed every  $m$  steps. The variable  $m$  is called the correcting period. The equations for a single iteration for this method are the same as for the conventional CORDIC algorithm, and every  $m$  iterations, the last iteration is repeated. At each iteration, however the number of bits that needs to be evaluated is different. This can be simplified to a constant of  $m+2$  digits. The selection of a positive or negative rotation is then made depending upon the sign of those  $m+2$  digits.

This method gives us the ability to trade off time for a single iteration (when the selection process is in the critical path), and the number of iterations. In the case of  $m=2$ , the iteration pair can be combined to form the Double Rotation method.

#### 3.5.5.4. Branching CORDIC Rotation

The Branching CORDIC Rotation was suggested in 1991 by Duprat and Muller [Duprat91]. The basis of this technique, is that two CORDIC engines are operated in parallel, and while it is clear which direction to rotate in, the two engines perform the same operations. When the sign of the angle cannot be evaluated by evaluating three digits of the redundant number representation of the angle, branching occurs, and the two engines rotate in different directions. The branching effect is shown in figure 3.10. For the following iterations, the two engines operate independently, until one of the branches is shown to be incorrect, at which point the values of the registers from the correctly rotated engine are copied into the other engine. One branch must be incorrect if the other branch needs to branch, or rotates in the direction that it rotated when it first branched. When the branch occurs, the 'positive' engine rotates in a positive direction, and the negative engine rotates in the negative direction. For subsequent iterations, if the positive engine does not rotate in a negative direction, the

negative engine result must be incorrect since the sum of all subsequent rotations cannot converge to the same angle for the two engines. Similarly, if the negative engine does not rotate in a positive direction, the positive engine result must be incorrect.

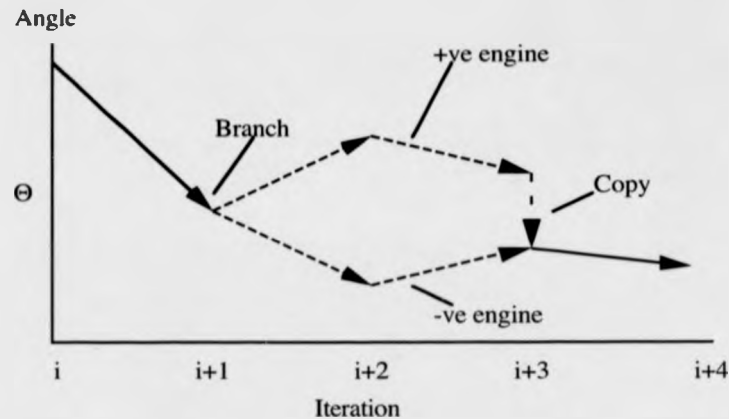


Figure 3.10. The Branching CORDIC Rotation.

Figure 3.11 shows the algorithm operation, and is taken from Duprat and Muller's paper. The  $s$  variable is the three digit approximation of the angle register.

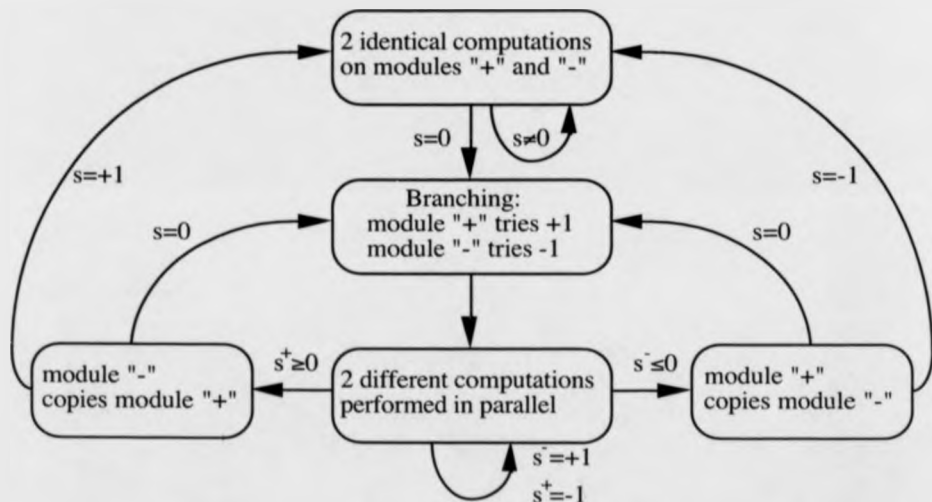


Figure 3.11. The branching CORDIC algorithm (from [Duprat90]).

This algorithm has a small number of digits to be evaluated for the angle register, and a simple iteration (the equations are the same as the conventional CORDIC algorithm). However, time must be allocated to allow transfer of the data from one of the CORDIC engines to the other. This will either reduce the clock speed, or increase the number of cycles used. In the later case, it is no longer possible to determine how many cycles the algorithm will take, although an upper bound can be found.

### 3.5.6. Multi-Dimensional systems

So far only 2 dimensional coordinate systems have been considered. Delosme and Hsiao [Delosme90, Delosme89] have expanded the CORDIC algorithm to include three and four dimensional cases. The approach they have taken, which, it will be shown later, is not the only practical approach, is to use the three Cartesian coordinates  $x$ ,  $y$ , and  $z$ , and rotate the vector formed by them about the directions  $(1,1,1)$ , and  $(1,-1,1)$ . This leads to the following iteration:

$$x_{i+1} = (1 - 3\delta_i^2)x_i + 2\delta_i(\gamma_i y_i - \eta_i z_i) + \gamma_i 2\delta_i^2 s$$

$$y_{i+1} = (1 - 3\delta_i^2)y_i + 2\delta_i(\gamma_i z_i - \gamma_i x_i) + \eta_i 2\delta_i^2 s$$

$$z_{i+1} = (1 - 3\delta_i^2)z_i + 2\delta_i(\eta_i x_i - \gamma_i y_i) + \gamma_i 2\delta_i^2 s$$

Where:

$$s = \gamma_i x_i + \eta_i y_i + \gamma_i z_i$$

$$\gamma_{i+1} = \text{sign}(x_{i+1} \cdot y_{i+1})$$

$$\eta_{i+1} = \text{sign}(x_{i+1} \cdot z_{i+1})$$

The shift sequence is modified in a similar way to that for producing hyperbolic functions. The optimal sequence repeats iterations roughly as  $k, 2k-1$ , etc.

### 3.5.7. CORDIC Applications

The CORDIC algorithm has many applications, and a full description of all the applications, and examples of use would take many pages to do the subject justice. A

description of this kind is outside the scope of this thesis. Instead of this a brief description of the more notable papers is given. The reader should be aware that the papers referenced here are simply examples, and by no means an exhaustive list. The Fourier transform is one obvious application for the CORDIC algorithm, and the algorithm has been used for this purpose for many years [Despain74]. Other applications include phase detectors for digital FM de-modulation [Ginderdeuren85], chirp z-transform processors in which the scaling factor is not an overhead [Hu90], Hough transformations [Timmermann89b], digital signal processing [Timmermann91], and singular value decomposition [Cavallaro87, Lin90, Lee91]. N-Dimensional CORDIC has applications in Householder transformations [Hsiao91]. The CORDIC algorithm has been used in a number of commercial devices, including the Plessey PDSP163XX series [Dixon90], Hewlett-Packard calculators [Walther71], and the Intel 80387 (used inside high power PCs) [Yuen88]. In addition, non-commercial CORDIC systems have been designed for image processing [Vaudin87], and general processing [Haviland80].

### 3.6. Higher Radix Algorithms

It is possible to evaluate certain functions at higher radices. If a redundant radix 4 system is to be used, as Rodrigues et al. suggested [Rodrigues81], then the selection process will be similar to that in the CORDIC algorithm, but instead of a selection  $s_i$  from the set  $\{-1,0,1\}$ ,  $s_i$  is selected from the combination of two of these sets i.e.  $\{-3,-2,-1,0,1,2,3\}$ . It should be noted however that  $\pm 3$  is difficult to generate as a variable multiplier relative to all the others which are simple shifts and/or negations. The set  $\{-2,-1,0,1,2\}$  is still redundant, and so this reduced set can be used for our selection. Figure 3.12 shows the  $\{-3\dots 3\}$ , and  $\{-2\dots 2\}$  sets over two iterations to show the redundancy in each.

With a CORDIC style iteration, it is preferable to do the same amount of rotation at an iteration  $i$ , whatever the data, to keep the scale factor constant. However, the scaling factor  $K$  for the CORDIC algorithm is accurate to  $n$  bits after  $\frac{n+1}{2}$  cycles,

and so iterations after this have no effect on the scaling factor. This means that a radix 4 approach can be used, using the selection set  $\{-2, -1, 0, 1, 2\}$  without having to calculate the scale factor, so long as a standard radix 2 approach is used for the first  $\frac{n+1}{2}$  iterations.

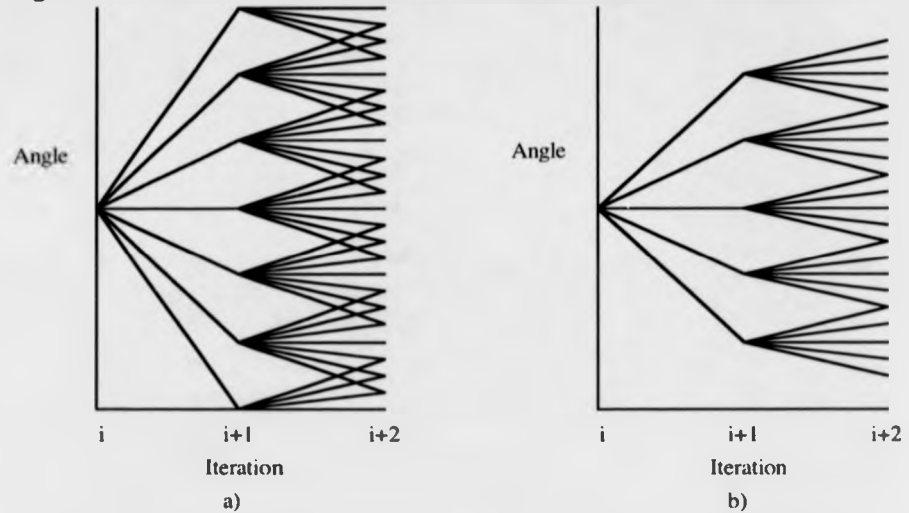


Figure 3.12. The selection sets a)  $\{-3, -2, -1, 0, 1, 2, 3\}$  and b)  $\{-2, -1, 0, 1, 2\}$  over two iterations.

The equations for the radix 4 iterations are:

$$x_{i+1} = x_i + s_i y_i 4^{-i}$$

$$y_{i+1} = y_i - s_i x_i 4^{-i}$$

$$z_{i+1} = z_i + \tan^{-1} s_i 4^{-i}$$

where the selection variable  $s_i$  is given by:



$$s_i = \begin{cases} -2 & \text{if } z_i 4^i \geq \frac{13}{8} \\ -1 & \text{if } \frac{5}{8} \leq z_i 4^i < \frac{13}{8} \\ 0 & \text{otherwise} \\ 1 & \text{if } -\frac{13}{8} < z_i 4^i \leq -\frac{5}{8} \\ 2 & \text{if } z_i 4^i \leq -\frac{13}{8} \end{cases}$$

This gives a 25% reduction in the number of cycles required for a given accuracy of results. Rodrigues also described algorithms for division log, exponential, tan cot and  $\tan^{-1}$ . Each of these algorithms converge at a rate of two bits per iteration.

Ercegovac described an algorithm for the calculation of logarithms divisions and exponentials [Ercegovac73]. The system is based on radix 16 redundant numbers. For example, to calculate the logarithm of  $x$  (in the range  $\frac{1}{2} \leq x < 1$ ), recall that:

$$\ln\left(\frac{a}{b_i}\right) = \ln a - \ln b_i$$

So if a simple set of factors  $b_i$ , can be found that makes  $a=1$ , and:

$$\frac{a}{\prod_{i=0}^M b_i} = x$$

$M+1$  factors are used in the approximation. An iteration can be formed to calculate  $\ln x$  of the form:

$$l_{i+1} = l_i - \ln b_i$$

Where  $l_i$  is an approximation of  $\ln x$ , and  $\ln b_i$  is a stored constant.  $l_0$  is set to zero, and hence the final result  $l_m$  is given by:

$$l_m = -\sum_{i=0}^M \ln b_i = \ln \frac{1}{\prod_{i=0}^M b_i}$$

It is now necessary to consider how to find a set of numbers  $b_i$  that will allow any number to be represented to the required accuracy. Ercegovac showed that this can

be done by making  $b_i = 1 + s_i 16^{-i}$  where  $s_i$  is a selection digit such that  $s_i \in \{-8, \dots, 8\}$ . He called this process multiplicative normalisation, as the  $s_i$  values are found by normalising  $x$  to 1:

$$x \cdot \prod_{i=0}^M (1 + s_i 16^{-i}) = 1$$

The selection digits are found by the following algorithm given by Ercegovic. In the algorithm,  $r_i$  is the scaled remainder (in binary two's complement form) and is given by:

$$r_i = 16^{i-1}(x_i - 1)$$

The  $j$ th bit of  $r_i$  is  $r_{i,j}$ . The multiplicative normalisation algorithm:

1.  $i \leftarrow 0$

$$s_0 \leftarrow \begin{cases} 1 & \text{if } \frac{1}{2} \leq x_0 < \frac{5}{8} \\ 0 & \text{if } \frac{5}{8} \leq x_0 < 1 \end{cases}$$

$$r_1 \leftarrow x_0(1 + s_0) - 1$$

2. for  $0 < i \leq M$

$$i \leftarrow i + 1$$

$$|s_i| \leftarrow \lceil (r_i + u_i) 16 \rceil$$

$$\text{sign}(s_i) \leftarrow -\text{sign}(r_i)$$

if  $i < i_1$  then

$$r_{i+1} \leftarrow 16r_i + s_i + s_i r_i 16^{-i+1}$$

otherwise

$$r_{i+1} \leftarrow 16r_i + s_i$$

where:

a)  $u_i$  is a step dependant rounding constant given by:

$$u_i = \sum_{j=3}^6 u_{i,j} 2^{-j}$$

$u_{i,j}$  are bits found by the Boolean functions:

$$u_{i,3} = K_1 r_{i,0} \overline{r_{i,2}}$$

$$u_{i,4} = K_1 r_{i,0} \overline{r_{i,4}} (\overline{r_{i,2}} + \overline{r_{i,3}})$$

$$u_{i,5} = K_1 (r_{i,0} + \overline{r_{i,3}} r_{i,4}) + K_2 [r_{i,0} + \overline{r_{i,3}} (\overline{r_{i,2}} + \overline{r_{i,3}}) + r_{i,6}] + K$$

$$u_{i,6} = K_1 \overline{r_{i,3}} r_{i,4} + K_2 r_{i,0} (r_{i,3} + r_{i,2} r_{i,3})$$

and  $K_1$ ,  $K_2$ , and  $K$  indicate (i.e. are high during) the iterations  $i=1$ ,  $i=2$ , and  $i>2$  respectively.

b)  $t_i$  is the scaled truncated unsigned remainder:

$$t_i = \begin{cases} \sum_{j=1}^6 r_{i,j} 2^{-j} & \text{if } r_{i,0} = 0 \\ \sum_{j=1}^6 \overline{r_{i,j}} 2^{-j} & \text{if } r_{i,0} = 1 \end{cases}$$

c)  $i_1 = \left\lceil \frac{(m+3)}{2} \right\rceil$

The combination of the two algorithms shown above was reported by Ercegovic, and this produces a result at a rate of four bits per iteration. Division and exponential algorithms of a similar type were also shown. Returning to the logarithm evaluation, Ercegovic showed that the memory storage requirement can be greatly reduced by the approximation:

$$\ln(1 + s_i 16^{-i}) \approx s_i 16^{-i} \text{ when } i \leq \left\lceil \frac{2 \log_2 10 - 1 + 4M}{8} \right\rceil$$

This is a Taylor series expansion making the algorithm a hybrid type.

### 3.7. Summary

In this chapter a number of different methods for calculating elementary functions have been shown. Different algorithms have different costs and benefits. The fastest of all is ROM table look ups, but they are not practical for more than about 20 bits, even with the best of today's memory technology. When fast multipliers are

available, series expansion can be a good solution, but with dedicated hardware, algorithms such as the CORDIC algorithm provide the most promising results. The ability to use redundant number systems can increase the algorithms throughput greatly. The numerous applications of the CORDIC algorithm shows that the tightness of the range constraints is not always a problem. The algorithm can be pipelined, and this gives benefits as the shifts used by the algorithm can then be hard wired. The CORDIC algorithm has been extended to three and four dimensions, but it will be shown in a later chapter that there are other ways to do this, and a generalised multi-dimensional case will be developed.

Redundant number systems, and higher radix methods are clearly of use in the context of elementary function evaluation, in the algorithms themselves, the number systems that the algorithms use, and in the hardware they are implemented on (for example multipliers for the Taylor series expansions). Table look ups require vast amounts of memory, and so any technology that can increase memory storage capacities will be of use in this area.

# 4

## A design methodology for MVL circuits

### 4.1. Introduction

In a previous chapter the state of the art in MVL circuit design was described. The theory of building general combinational logic circuits from voltage mode circuits is well known, but the use of current mode circuits is not. Indeed current mode circuits tend to have been designed for a specific purpose [Current88], and have not been built up into sets of logical functions. One reason for this is that voltage mode circuits tend to be radix three (or ternary), whereas the current mode circuits are mainly radix four (or quaternary). This makes the choice of function easier in the voltage mode case. For two input functions, there are more than five orders of magnitude fewer ternary functions than there are quaternary functions. The aim of this chapter is to describe a method for the design of current mode circuits that will allow the best possible use to be made from current mode MVL design. Once it is possible to design current mode MVL circuits easily, a basic set of cells capable of generating any logical function is required. These are described for a quaternary system. The use of current mode MVL circuits in a cell library is also discussed. Finally, the design of three MVL circuits in VLSI is discussed, along with details of test results from the fabrication of these circuits.

Optical logic is very different to electronic logic, but as has already been pointed out, some optical circuits operate by having a thresholding function

[Osawa92]. This makes the design style very similar to current mode MVL as optical signals can be summed easily. Specific mention will not be made to optical circuits in this chapter, but it is worth noting that this potential new logic type would be highly compatible with a large amount of the design and methodology described in this chapter.

## 4.2. An MVL design methodology

Circuits designed using current mode MVL will only be efficient in terms of time area and power if they take full advantage of the summing node. Because of this, any methodology for current mode MVL design must focus on this. Circuits can have many inputs, that enter the cell by the same physical connection, and these can be formed in a manner of the designer's choice so that the decoding logic is simple. As was noted in an earlier chapter, current mode circuits can generally be decomposed into the free summing function and three distinct sections: a decoder, binary logic (which is not always present), and an encoder circuit. This form is shown in figure 4.1.

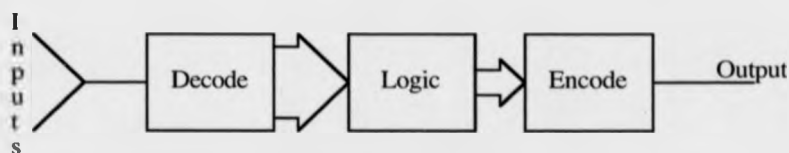


Figure 4.1. The DLE decomposition of a current mode circuit.

The inputs to the circuit are summed on the input node and fed to the decoder. The decoder produces a set of binary signals each of which indicates whether the input current total was above or below a particular current level. These signals can then be adapted by the binary circuits to form signals that can correctly operate the encoders which provide the current mode output signal. This structure has been shown in Current's Quaternary Full Adder (QFA) [Current87]. The currents are encoded in a linear form, i.e. the value "1" is encoded as  $10\mu\text{A}$ , the value "2" is encoded as  $20\mu\text{A}$ , and so on. These individual inputs are summed together so that the decoder input is

directly proportional to the sum of the input signal values. The decoder thresholds then distinguish each possible input level from each other, and the binary logic drives the outputs so that they are linearly encoded modulo 4 sum and carry outputs. This in itself is a useful circuit, but many different and useful logic circuits can be constructed in the same form. The important factor is that almost any part of the circuit can be modified, and this may result in a different function from the circuit. If a circuit has two input values  $x$  and  $y$ , and they have been encoded as currents using the functions  $A()$ , and  $B()$  respectively, then when they are summed together, the sum is equal to  $A(x)+B(y)$ . The functions  $A()$  and  $B()$  may be linear, and may be (but are not necessarily) the same as each other. The summed current is then decoded and acted on by the logic, which can be represented by the function  $C()$ . Hence the final encoding is  $C(A(x)+B(y))$ . In the full adder example, there are two outputs, and each has its own function  $C()$ . The carry function is in effect a quantised division by four, and the sum output is a modulo four function.

The choice of the functions  $A()$ ,  $B()$ , and  $C()$  is not easy to automate. An iterative method for finding the functions is described below. It is a very simple method, and it may fail to find a set of functions that will work. Even if it does find a set of functions, they may not be the best functions in terms of implementation. This method should however, with the help of a little intuition from its user provide a fairly quick method for finding  $A()$ ,  $B()$ , and  $C()$  in most simple cases. The algorithm which is demonstrated later is as follows:

1. Choose arbitrary functions for  $A()$ , and  $B()$ . In the first instance, a linear encoding is probably best, unless the user has prior knowledge about the desired function of the circuit.
2. Form a table of  $A(x)+B(y)$  and the required circuit output for all  $x$ , and  $y$ .
3. Use this table to form a new table of which outputs should be produced for a given  $A(x)+B(y)$ .
4. If more than one output value is required for one value of  $A(x)+B(y)$ , then either  $A()$  or  $B()$  (or both) must be changed, and all previous steps repeated.

5. As long as the highest threshold needed is within the available resolution, select threshold levels, and form  $C()$  using threshold function outputs. Otherwise, either  $A()$  or  $B()$ , or both, must be changed, and all previous steps repeated.

#### 4.2.1. A worked example: The maximum function

To make the operation of the algorithm more easy to understand, an example is given below. The function chosen is the maximum function of a pair of variables. This is not a simple function to implement and was deliberately chosen for this reason. It should give a better understanding of the algorithm than would be gained from a trivial case. The first stage is to choose  $A()$  and  $B()$ . As the function is unaffected by the order of its inputs ( $\text{MAX}(x, y) = \text{MAX}(y, x)$ ) the same encoding is chosen for both inputs. To start with, a simple linear encoding is used as shown in table 4.1.

input	A(input)	B(input)
0	0	0
1	1	1
2	2	2
3	3	3

Table 4.1. First encoding functions in current units.

The function outputs are shown in current units. A current unit is the difference in current between two adjacent logic levels, and is typically 10 to 50  $\mu\text{A}$ . The linear function is used first as it is simple to calculate, and produces a small range of summed currents with no gaps that represent wasted logic levels.

The next stage (stage 2) is forming a table of  $A(x)+B(y)$ , and the output required for all  $x$  and  $y$ . This table is shown as table 4.2. The first two columns give each possible  $x$  for each possible  $y$  in a similar manner to that of a binary truth table. The next two columns give  $A(x)$ , and  $B(y)$  in current units as the encoding is linear these are the same as the first two columns. The final two columns show  $A(x)+B(y)$ , and the maximum of  $x$ , and  $y$ . It is the relationship between the data in these two columns that is important, as will be seen at the next stage.



x	y	A(x)	B(y)	A(x)+B(y)	MAX(x,y)
0	0	0	0	0	0
0	1	0	1	1	1
0	2	0	2	2	2
0	3	0	3	3	3
1	0	1	0	1	1
1	1	1	1	2	1
1	2	1	2	3	2
1	3	1	3	4	3
2	0	2	0	2	2
2	1	2	1	3	2
2	2	2	2	4	2
2	3	2	3	5	3
3	0	3	0	3	3
3	1	3	1	4	3
3	2	3	2	5	3
3	3	3	3	6	3

Table 4.2. Sum of encoder outputs and Maximum function for linear encoding.

Stage three is to form a table of the sum of the encoded currents relative to the required output. This is done in the form shown in table 4.3. The table is formed as follows. Each possible output value forms a column of the table, and each possible value of the sum of the encoded currents  $A(x)+B(y)$  forms each row. Possible values are simply all the values that appear in the appropriate column of table 4.2. Then for each row in table 4.2, a mark is made in the cell of table 4.3 that corresponds to the sum of currents and output value shown. The cell in table 4.3 is either marked or unmarked. The number of times it is used is not relevant.

		Max			
		0	1	2	3
Sum	0	✓			
	1		✓		
	2		✓	✓	
	3			✓	✓
	4			✓	✓
	5				✓
	6				✓

Table 4.3. Sum of currents vs. output function

Having created this table, it is used in the fourth stage to decide whether it is worth continuing with the current encoder functions. For it to be worth continuing, it must be possible to find a function  $C()$  that satisfies:

$$C(A(x) + B(y)) = \text{required\_function}(x, y)$$

If  $C()$  is to satisfy the above equation, its input (the sum of the encoded currents) must be different for each required output value. It can be seen in table 4.3, that if a current input of 2 units is received by the decoder, the output required could be a 1 or a 2, and hence  $C()$  is not realisable in this case. An input to the decoder of 3 or 4 units of current is also ambiguous. In general, if more than one marked cell appears in any one column, then  $C()$  is unrealisable with the input encoders used. Having found that a linear encoding does not work, it is possible to use the information in tables 4.2 and 4.3 to help choose a different encoder. Examining the lines in table 4.2 corresponding to a current sum of 2 units, it can be seen that this value is produced by  $x$  and  $y$  inputs of 1 and 1 (maximum 1) or 0 and 2 (maximum 2). To make this different, the encoding of input values of either 0, 1, or 2 must change. This analysis can be completed for the other two current sum rows on which the encoding fails. In all cases, the encoding for the input values of either 1 or 2 must change. One way to change the encoding is to increase both the encodings of input values of 2 and 3 by one current unit. However, if this encoding is used, then it is clear that there will still be no difference between the sums of the encodings of a 1 and a 2 input, and a 0 and a 3 input since both sums will increase by one current unit. So the encoding for a 3 three input must be increased by 2 to 5 current units. This encoding is shown in table 4.4.

input	A(input)	B(input)
0	0	0
1	1	1
2	3	3
3	5	5

Table 4.4. An alternative encoding.

Having chosen a new encoding, stages 2 and 3 in the algorithm are repeated. The results are shown in tables 4.5 and 4.6.

x	y	A(x)	B(y)	A(x)+B(y)	MAX(x,y)
0	0	0	0	0	0
0	1	0	1	1	1
0	2	0	3	3	2
0	3	0	5	5	3
1	0	1	0	1	1
1	1	1	1	2	1
1	2	1	3	4	2
1	3	1	5	6	3
2	0	3	0	3	2
2	1	3	1	4	2
2	2	3	3	6	2
2	3	3	5	8	3
3	0	5	0	5	3
3	1	5	1	6	3
3	2	5	3	8	3
3	3	5	5	10	3

Table 4.5. The alternative encoding sum, and required output.

		Max			
		0	1	2	3
Sum	0	✓			
	1		✓		
	2		✓		
	3			✓	
	4			✓	
	5				✓
	6			✓	✓
	8				✓
	10				✓

Table 4.6. The alternative encoding sum vs. output required.

It can be seen from table 4.6 that there is still no realisable C() with the alternative encoding given in table 4.4. The encoding must be further modified before it can be used. To prevent the encoded sums of 3 and 1 giving the same result as 2 and 2, the encoding for 3 must be 7 current units. This final encoding is shown in table 4.7, and the current sum vs. output required is shown in table 4.8.

input	A(input)	B(input)
0	0	0
1	1	1
2	3	3
3	7	7

Table 4.7. An encoding to give a maximum function.

		Max			
		0	1	2	3
Sum	0	✓			
	1		✓		
	2		✓		
	3			✓	
	4			✓	
	6			✓	
	7				✓
	8				✓
	10				✓
	14				✓

Table 4.8. The sum vs. required output for the maximum function

The encoding given in table 4.7 is such that the sum of two lower encoded values cannot be the same as a higher encoded value. In other words:

$$A(n+1) > 2A(n)$$

Table 4.8 shows that there is no problem with implementing a function  $C()$  to give a maximum function. The threshold circuits produce an output as follows:

$$Threshold\_output = \begin{cases} 1 & \text{if } I_{input} < Threshold \\ 0 & \text{Otherwise} \end{cases}$$

The final stage is to check that the solution is within the available resolution, and implement it if possible. The available resolution is the highest threshold value possible, and this will be determined by the relative error of the current sources. A number of single current sources are summed together in the encoder as will be seen later. When the combined error of these sources is greater than half a current unit, the decoder circuit cannot distinguish between two adjacent logic levels. In this case, the highest threshold level needed is 6.5 current units, corresponding to the change in output value from 2 to 3. This is likely to be within the resolution of the system, whereas the maximum current value of 14 is not. Since it is not necessary to differentiate between 14 and 13 current units, this does not matter. Assuming that the output of the circuit is to be encoded in a linear manner, no logic at all is required. All that is needed is three threshold circuits (thresholds at 0.5, 2.5, and 6.5 units), and

three switched unit current sources. The decoder is shown in figure 4.2, with its output linearly encoded.

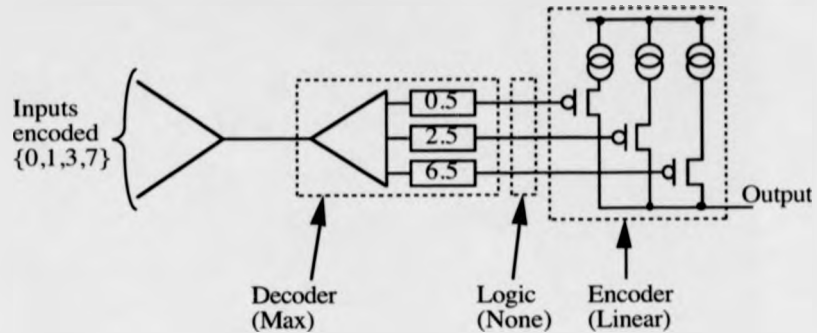


Figure 4.2. A maximum function decoder.

The encoder can also be designed easily. Figure 4.3. shows the encoder circuit with three binary inputs that are identical to those output by the maximum function decoder. The values by the current sources indicate the source output in current units.

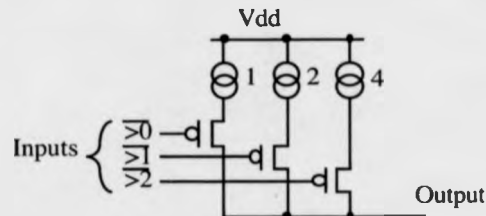


Figure 4.3. The maximum function encoder.

A maximum function circuit has been shown, but this is not necessarily the best circuit. Many different encoder and decoder functions will exist that perform the same function.

#### 4.2.2. General function implementation

It has already been mentioned that it is not necessary for the encodings  $A()$  and  $B()$  to be the same. There are some obvious encoding strategies that may, at first sight seem like a good idea. For example, if  $A()$  and  $B()$  are both linear encoders, but  $A()$

has 4 times the magnitude of B(), as shown in table 4.9, then every possible combination of inputs is distinguishable from every other combination.

input	A(input)	B(input)
0	0	0
1	4	1
2	8	2
3	12	3

Table 4.9. A linear encoding that makes the inputs totally distinguishable

It would therefore be simple to create any function by generating a switching signal indicating each individual input combination. This signal could then activate a current source when that input combination occurred. This encoding suffers however from two major problems, size and resolution. The circuit, because it is not aimed at a particular function, yields very inefficient implementations for all but a few functions. For example, if the maximum function were to be generated using this function, the decoder would have 8 threshold functions instead of 3, and about 14 binary logic gates would be required instead of none in the earlier design. The output encoder for this circuit would also be considerably larger (about 100%), than in the earlier design. The second problem is that of resolution. Taking the maximum function example again, the highest threshold function required would be 10.5 current units as opposed to 6.5. Clearly encoder functions such as this are not a practical solution.

#### 4.2.3. Ease of implementation of a function

There are clear advantages to be gained by understanding how easily a particular encoding can be decoded, in order that not just any implementation can be found for a function, but if possible, an efficient implementation of that function. The encoder circuitry is in general considerably simpler than the decoder and logic circuitry. The decoder size is basically determined by the number of thresholds required. Table 4.10 shows the sum of current vs. output for the modulo 4 sum output of Current's adder (the effect of the carry input is ignored). When this is compared to table 4.8, it can be seen that there is a distinct difference between these two tables. The encoding for the maximum function causes all of the sums of currents that correspond

to a particular output value to be within one range of currents, and that no current within that range corresponds to any other output value. This is very different to the modulo 4 sum, in which no two adjacent current levels correspond to the same output value.

		Mod 4 Sum			
		0	1	2	3
Current	0	✓			
	1		✓		
	2			✓	
	3				✓
	4	✓			
	5		✓		
	6			✓	

Table 4.10. The current vs. output value for a modulo 4 sum.

The difference in the arrangement of the output values relative to the sum of the encoded inputs has a profound effect on the size of the circuit. The modulo 4 sum needs 6 threshold circuits, and about 5 binary logic gates, as opposed to the 3 threshold circuits, and no binary logic of the maximum circuit. Some functions such as the maximum function lend themselves to efficient implementation in current mode MVL even if the encodings used are not obvious at first. An examination of a balanced ternary product function in the context of this methodology helps to explain why some functions are easier to implement than others. Table 4.11 shows the required function as a truth table.

x	y	output
-1	-1	1
-1	0	0
-1	1	-1
0	-1	0
0	0	0
0	1	0
1	-1	-1
1	0	0
1	1	1

Table 4.11. The truth table for a balanced ternary product function.

An efficient set of encoders causes the sums of currents corresponding to a particular output value to be grouped together. In order that the output values are grouped together, one of two situations must exist:

1. The sum of currents corresponding to a 1 output is larger than the sum of currents corresponding to a -1. In this case:

$$A(1) + B(1) > A(-1) + B(1)$$

Thus, if the inputs are both 1 the output should be 1, whereas if the inputs are 1 and -1, the output should be -1. This equation can be simplified by the removal of  $B(1)$  from both sides, to give:

$$A(1) > A(-1)$$

However, it is also clear that:

$$A(-1) + B(-1) > A(1) + B(-1)$$

Which simplifies to:

$$A(-1) > A(1)$$

Clearly, the two constraints cannot hold at the same time.

2. The sum of currents corresponding to a -1 output is larger than the sum of currents corresponding to a 1. This is not realisable either for almost identical reasons for those given in the first case.

There is therefore, no way of encoding a balanced ternary product function such that the sums of currents are grouped to give a minimal number of threshold functions in the decoder. In general, the absolute minimum number of threshold functions possible will be just enough to separate each output value from the others, i.e. one less than the total number of possible output values. The maximum number of threshold functions that should ever be needed will be 1 less than the total number of different possible current sum values. This allows for a threshold function between each adjacent current sum level. The number of threshold circuits required is therefore bounded by:

$$No_{current\_sum\_values} - 1 \geq No_{threshold\_circuits} \geq No_{output\_values} - 1$$



#### 4.2.4. Negative Currents

The encoders described so far have all sourced current to the summing node. However there is no reason why the encoder should not sink current from it, or a combination of the two. If one encoder sources current to the node, and the other sinks current from it, then the free function becomes a subtraction rather than an addition. In this case, the sum of currents becomes a difference in current magnitudes. The easiest way to consider these encoders is simply to view an encoder sinking a current from the node as a negative current. An example of this kind of encoder is given in figure 4.4, with its encoding function.

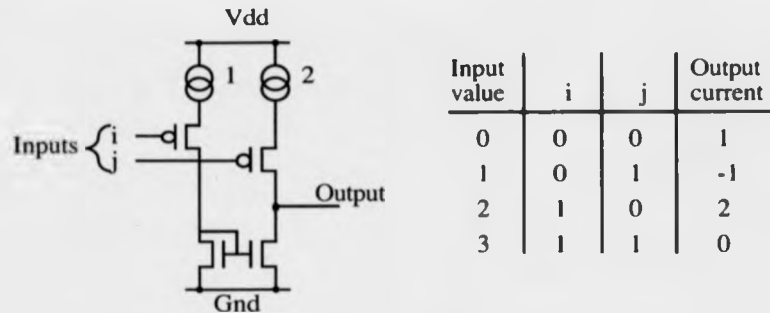


Figure 4.4. An encoder with a negative output.

In certain situations, an encoder can be designed so that a negative current does not have any specified current itself, but instead can sink so much current that the node is grounded when that encoding is specified. This can be demonstrated by returning once again to the example of the maximum circuit. A new encoding is shown in table 4.12, with the output for 3 being simply  $<0$ . This means that if one of the encoders has a 3 input to it, the output of the other encoder will flow into the first encoder rather than the decoder. The decoder input stage is a simple n-type current mirror, so it does not source current to the input node, and hence the current flow into the encoder is small. The decoder will detect no current since it all flows into the encoder. The other encodings could be increased by 1 current unit so that a 0 input can be distinguished from a 3. However, a better way to do this would be to leave the 0, 1, and 2

encodings as they were in table 4.7, and add a 1 unit current source to the node un-switched.

input	A(input)	B(input)
0	0	0
1	1	1
2	3	3
3	<<0	<<0

Table 4.12. A maximum function encoding that uses a grounding output.

This form of encoding is very simple to produce, as shown in figure 4.5. The n-type transistor must be made large enough that it can sink the maximum output current from the other encoder, and the 1 unit current source, with a negligible voltage drop across it. The n-type transistor does not have to sink current from its own current sources since it is switched so that only the current source or sink is on at any one time, and not both.

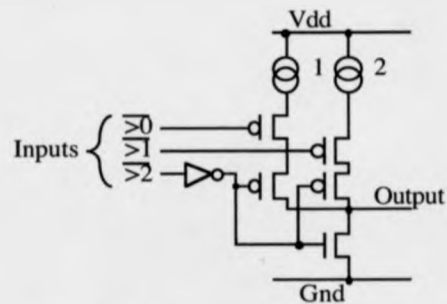


Figure 4.5. The encoder for the maximum function using a grounding output.

		Max			
		0	1	2	3
Current	0				✓
	1	✓			
	2		✓		
	3		✓		
	4			✓	
	5			✓	
	6			✓	
	7			✓	

Table 4.13. The sum of currents vs. output value for the grounding output encoding (including 1 current unit un-switched source).

The sum versus output is given in table 4.13. This shows that the maximum current for this encoding is considerably lower than the encoding given earlier. Indeed the maximum current is only 50% of the maximum current using the previous encoder. The thresholds for this encoding are 0.5, 1.5, and 3.5 current units, which is much lower than with the previous encoding, which is another advantage of the grounding output. The decoder required for the grounding output encoding of the maximum function is shown in figure 4.6. The grounding output encoder can only be used with a specific type of function, because if one encoder grounds the node, there is no way of distinguishing between the different outputs of the other encoder. Hence the grounding output can only be used if one particular input value to one of the encoders always results in the same output regardless of the other encoders input value. In the case of the maximum function, if either encoder has a 3 input to it, then the output must be a 3, and so the input value 3 can be used to ground the summing node.

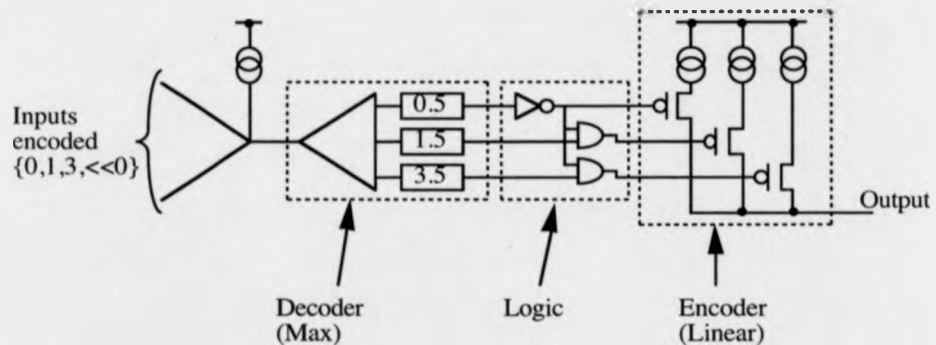


Figure 4.6. The maximum function decoder and logic for use with the grounding output encoders.

The grounding output is clearly very useful, but there is one distinct disadvantage to this kind of encoding. In an earlier chapter, it was mentioned that current mode mirrors can be made considerably faster if they are biased with a small amount of current so that they are always conducting. This is because the transistor gate voltage always stays above the FET's threshold voltage. Charging up to this voltage from 0 volts takes a long time relative to the changing from one current level to another. This

biasing is not possible with grounding output encoders since the bias current would be lost into the encoder.

#### 4.2.5. Multi-stage functions

In a previous chapter, the adder used by Kawahito et al. [Kawahito87] was described. This adder was constructed in two stages as shown in figure 4.7. There is no reason why this form of construction should not be used for other functions. A modulo 4 sum constructed in this way would require almost 50% fewer threshold circuits, no binary logic (as opposed to about 5 gates for the single stage version), and only marginally more switched current sources. The disadvantage is that the resolution of the circuit needs to be higher, since the output from the first stage is subtracted from an un-quantised copy of the input current.

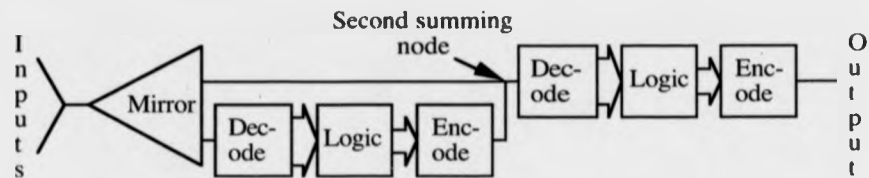


Figure 4.7. A multi stage function.

### 4.3. Basic circuit functions for current mode MVL

Having described how functions can be designed, leaves a major question unanswered, which functions should be implemented? It was noted in an earlier chapter that there are more than 4 billion possible functions of two input variables. Even the number of one input functions (256) makes implementation of all of them impractical. In this section, useful functions are designed, and their application to circuit design described. Functions useful for both arithmetic and logic are described, as both are areas in which efficient basic circuits are important. A quaternary system is assumed, but most of the comments are applicable to a system of any radix. Examination of the quaternary circuits should enable a designer to quickly create a similar circuit in a different radix.

#### 4.3.1. The choice of functions

In choosing a set of functions for quaternary current mode MVL, it is useful to refer to the binary operators in common use, and try to enlarge these functions to higher radices. The three basic operations in Boolean logic are AND, OR, and NOT. There are others such as EXclusive-OR (or EXOR) and composites such as NAND, but just the basic 3 will be considered. A Boolean function is written in the form:

$$F = \bar{A} \cdot B + \bar{B}(C \cdot D + \bar{E})$$

Where a  $\cdot$  indicates an AND operation, and a  $+$  indicates an OR operation. An overhead bar indicates a NOT operation. The Boolean expression above can be expanded out to give the following:

$$F = \bar{A} \cdot B + \bar{B} \cdot C \cdot D + \bar{B} \cdot \bar{E}$$

This is known as the "sum of products" form, which gives an indication of a possible choice of functions, the SUM and PRODUCT functions. In fact, it is known that the modulo 4 SUM and PRODUCT functions can generate any function in quaternary logic, by using generalised Reed-Muller expansions. It should be noted however that the OR Boolean operation is not in fact modulo 2. That is the EXOR operation. Tables of the modulo 4 SUM and PRODUCT functions are given in table 4.14 a, and b respectively.

	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

a)

	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

b)

Table 4.14. Modulo 4 a) SUM and b) PRODUCT functions.

One major advantage of these circuits is that with the addition of a little extra circuitry to give them carry outputs, they can also be used as arithmetic operators with obvious uses in multipliers and adders. In terms of logic, these two functions can be used together to create any logical function [Hurst84].

#### 4.3: Basic circuit functions for current mode MVL

The SUM and PRODUCT extensions of the Boolean AND and OR operations are not the only extensions available. One other common form which allows direct implementation from a "sum of products" type form is the MAX (maximum) and MIN (minimum) functions. The MAX and MIN functions are given in table 4.15.

	0	1	2	3
0	0	1	2	3
1	1	1	2	3
2	2	2	2	3
3	3	3	3	3

a)

	0	1	2	3
0	0	0	0	0
1	0	1	1	1
2	0	1	2	2
3	0	1	2	3

b)

Table 4.15. a) MAX and b) MIN quaternary functions.

The MAX and MIN functions are a simple extension of the binary OR and AND functions respectively. Indeed examination of table 4.15 shows that if binary data were applied to either, they would appear to be the binary function that they are an extension of. The SUM and PRODUCT functions do not possess this property. MAX and MIN functions, however are not enough on their own to implement logical functions. It is necessary to have an equivalent to the Boolean NOT gate. There are a number of different options for the extension to a Boolean NOT gate and some of these are given in table 4.16. These are all single input or 'unary' functions.

Input	Option			
	a	b	c	d
0	1	3	2	3
1	2	0	3	2
2	3	1	0	1
3	0	2	1	0

Table 4.16. Various quaternary extensions to the NOT gate

The first three options are modulo 4. They are a) input+1 (SUCCESSOR), b) input-1, c) input+2. Option d is 3-input (COMPLEMENT). The COMPLEMENT function, with the MAX and MIN functions will allow all logical functions to be produced [Hurst84]. There are other unary functions that are well used such as the literal operator which outputs a zero unless one particular value is input, when it outputs the maximum possible value (3 in the quaternary case). For a quaternary system there are 4 literal operators.

To give a minimum number of logic gates to be designed, the modulo 4 SUM and PRODUCT is the best solution, and the addition of carry output circuitry would give easy arithmetic circuit production. A second choice is MAX, MIN and COMPLEMENT. It was shown earlier that the MAX function had an efficient implementation, and so this set may yield better circuits in terms of time and area than the SUM and PRODUCT circuits. In the following sections, unary functions in general, the MAX and MIN functions, and the SUM and PRODUCT are all shown in further detail.

#### 4.3.2. Encoders as unary functions

In current mode MVL the most useful circuit element is the current summing node. For two input functions, the summing node is used to sum the inputs. However, with unary functions there is only one input anyway, and so no use is made of the sum. A unary function could be constructed from a DLE type of structure, as shown in figure 4.8.

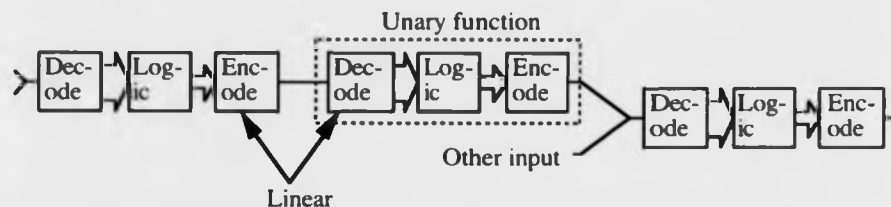


Figure 4.8. A unary function in a circuit in DLE form.

However, the only sensible input encoding would be a simple linear encoding. This would be immediately decoded, and fed to some binary logic. It is therefore a simple matter to omit the output encoder of the previous function, and replace it with the logic and encoder stage of the unary function as shown in figure 4.9. Whereas when the interconnection between the two input (dyadic) function and the unary function is a current mode signal the two circuits could be physically separate from each other without much routing being required, the removal of the encoder and decoder could easily leave the designer with up to 3 wires to route instead of the single current mode

signal. The solution to this is to view the unary function as the output stage of the dyadic function before it. Hence the two functions are placed physically next to each other.

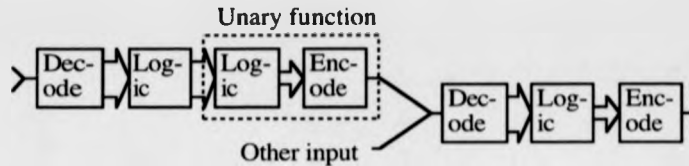


Figure 4.9. A unary function as the output of another multi input function.

The output of the unary function will be fed, most probably to a dyadic gate, so its output must be encoded correctly for that gate. Therefore, each unary function must be designed, with an encoder for each dyadic function.

#### 4.3.3. MAX and MIN

Earlier in this chapter, two designs for MAX functions were shown. The MIN function is very similar indeed. Examination of the MAX and MIN functions given in table 4.15 shows that a MIN function can be constructed from a MAX gate by complementing both of the inputs, and the output. If a COMPLEMENT is represented by an overhead bar, MAX is represented by a  $\vee$ , and MIN is represented by a  $\wedge$ , then this can be written as:

$$x \wedge y = \overline{\overline{x} \vee \overline{y}}$$

This is very similar to De Morgan's theorem in Boolean logic:

$$x \cdot y = \overline{\overline{x} + \overline{y}}$$

It is therefore possible to construct a MIN gate from a MAX gate and 3 COMPLEMENT functions. It is however still interesting to examine the MIN gate designs in order to evaluate whether or not they are more efficient than the MAX COMPLEMENT design. Indeed, there is no reason to suppose that the MAX gate should be more efficient to design than the MIN gate. It is even conceivable, that a MIN gate with its inputs and output complemented could form an efficient MAX gate.



#### 4.3: Basic circuit functions for current mode MVL

Having designed a MAX gate, there is prior knowledge that can be used to help the MIN gate design. It was noted that in the MAX gate design, the encoded current for a value  $x+1$  had to be more than twice the sum of the current for  $x$ . For the MIN encoding, the equivalent requirement is that the current for a value of  $x$  summed with the maximum current encoded is less than twice the encoding of  $x+1$ . In other words, for the same  $A()$  and  $B()$ :

$$A(x) + A(3) < A(x+1) + A(x+1)$$

This ensures that any set of inputs produces a lower summed current than any other set of inputs with a higher minimum. Two encodings that satisfy this criterion are given in table 4.17. Table 4.17a is a simple encoding whilst 4.17b makes use of a grounded input encoder. The tables assume that  $A()$  and  $B()$  are the same.

Input	A(Input)	Input	A(Input)
0	0	0	<<0
1	4	1	0
2	6	2	3
3	7	3	4

a)
b)

Table 4.17. Two encodings for a MIN function.

The encodings shown can be analysed, and put into current sum vs. required output. This is done for the simple encoding in table 4.18, and for the grounded output encoding in table 4.19.

		Min			
		0	1	2	3
C u r r e n t	0	✓			
	4	✓			
	6	✓			
	7	✓			
	8		✓		
	10		✓		
	11		✓		
	12			✓	
	13			✓	
	14				✓

Table 4.18. The simple MIN encoding current sum vs. required output.

#### 4.3: Basic circuit functions for current mode MVL

The simple encoding gives rise to a similar decoding to that used by the MAX function. However, the thresholds for the MIN function would be 7.5, 11.5, and 13.5. This means that the MIN circuits would require high resolution, and consume much more power. Hence at this stage it can be seen that the simply encoded MIN function is not a good choice for direct implementation.

		Mod 4 Sum			
		0	1	2	3
Current	0	✓			
	1		✓		
	4		✓		
	5		✓		
	7			✓	
	8			✓	
	9				✓

Table 4.19. The grounded output MIN encoding current sum vs. required output.

The grounded output encoding suffers similar problems to the simple encoding, and so it is not worth expending effort on the MIN function. Instead, a second encoder that performs a complement function is required that will allow a MAX function to become a MIN.

Figure 4.10 gives a possible encoder for an complemented simple MAX encoding. The inputs are all inverted, and the current weights are reversed to form the complement. It may be possible, depending on technological constraints to replace the inverter and p-type transistor with a single n-type transistor. In this case, the complement function is in effect free, since no extra area would be used relative to a normal encoding.

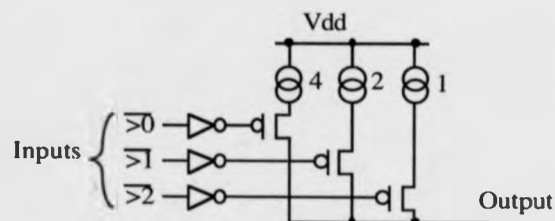


Figure 4.10. A complemented MAX encoder (simple encoding).

#### 4.3: Basic circuit functions for current mode MVL

The grounded output encoding can also be complemented as shown in figure 4.11. As with the simple encoder, the inverters may be omitted if n-type transistors can be used as the current switches. In this case, the complemented encoding would be smaller than the un-complemented encoding, as the grounding transistor must be an n-type transistor. Hence the inverter in the un complemented encoding cannot be removed. Even using p-type switch transistors, the complemented encoder uses only one more inverter (i.e. two transistors) than the un-complemented encoder.

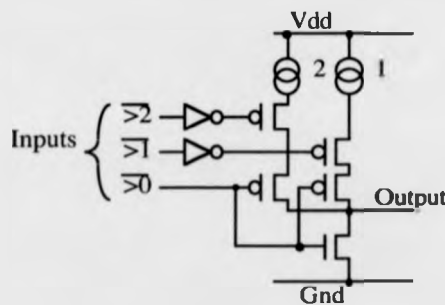


Figure 4.11. A complemented MAX encoder (grounded output encoding).

The MAX function using optionally complemented grounded output encoders provides the ability to produce any logical function, with very low resolution requirements (maximum threshold of 3.5 current units). Its importance as a current mode MVL building block is un-questionable.

#### 4.3.4. SUM and PRODUCT

The modulo 4 sum function has been described earlier in this chapter and in previous chapters. The effect of the modulus is that a maximum number of decoders (6) is required to decode the function. Because no two adjacent sum values give the same output, there is nothing to be gained by using a grounded output encoder.

The modulo 4 product must also be designed to allow logic circuits to be constructed arbitrarily. The product can be simply implemented in two ways:

1. A set of switched copies of the one input could be summed together using modulo 4 sum circuits according to the magnitude of the second input. However, does not take

#### 4.3: Basic circuit functions for current mode MVL

advantage of the current sum to reduce the amount of interconnect between basic blocks. Also, being itself constructed from other building blocks, means that its implementation will be relatively large in comparison to the other basic building blocks.

2. The methodology described earlier can be employed. In this case it would not be unreasonable to start with a linear encoding as was done in the example. However, it is well known that:

$$x \cdot y = \log^{-1}(\log x + \log y)$$

This gives an instant set of functions A(), B(), and C(). The encoders should produce an approximation to the log form. Table 4.20 shows the log of the quaternary values.

Input	log(Input)
0	$-\infty$
1	0
2	0.693...
3	1.098...

Table 4.20. The natural logarithm.

The log of zero could give some cause for concern, since it is minus infinity, however this can be approximated as  $\ll 0$ , and hence the encoding will produce a grounded output style of circuit. The other values can be scaled up such that there is a difference of at least 1 between each output value. The approximation is shown in table 4.21.

Input	A(Input)
0	$\ll 0$
1	0
2	2
3	3

Table 4.21. The log approximation encoding

This encoding can be used as the first guess encoding for the methodology described earlier. The sum of currents versus the required output is shown in table 4.22. The encoding works with a unit current added to the summing node to distinguish between a grounded output and no current flow. The grounded output must be capable of sinking 3 current units from the other encoder, and one current unit that is added to the node, without an appreciable voltage drop.

		Mod 4 Product			
		0	1	2	3
Current	0	✓			
	1		✓		
	3			✓	
	4				✓
	5	✓			
	6			✓	
	7		✓		

Table 4.22. The current sum vs. required output for the log encoding.

The encoding given works, in that a function  $C()$  is realisable. However, the function is not very efficient since related outputs are not grouped together. The function  $C()$  can be taken from the sum of currents table, and is shown in figure 4.12. The circuit consists of six threshold functions and the equivalent of about 6 logic gates. This makes the modulo 4 product roughly the same size as the modulo 4 sum.

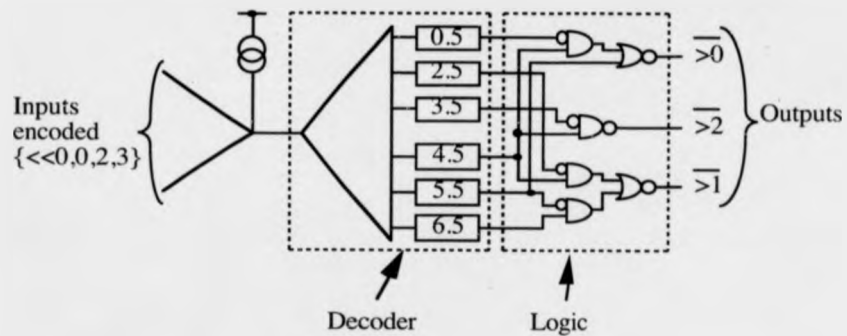


Figure 4.12. The modulo 4 product decoder.

The encoder for the modulo 4 product is shown in figure 4.13a. The encoding can be modified to give a purely positive output current to increase the speed of the circuit. The encoding for the circuit is shown in table 4.23, and the encoder circuit is shown in figure 4.13b.

Input	A(Input)
0	7
1	3
2	1
3	0

Table 4.23. A modulo 4 product encoding using all positive currents.

The encoding that does not make use of the grounded output will clearly use more power, but the resolution required is no higher. The all positive encoding is formed by inverting the value of the encoding, and then finding the minimum value for which the function  $C()$  is realisable. In this case it is necessary for the sum of the encoding for zero to be greater than the sum of any two other encodings. Hence the encoding for zero is 7 current units.

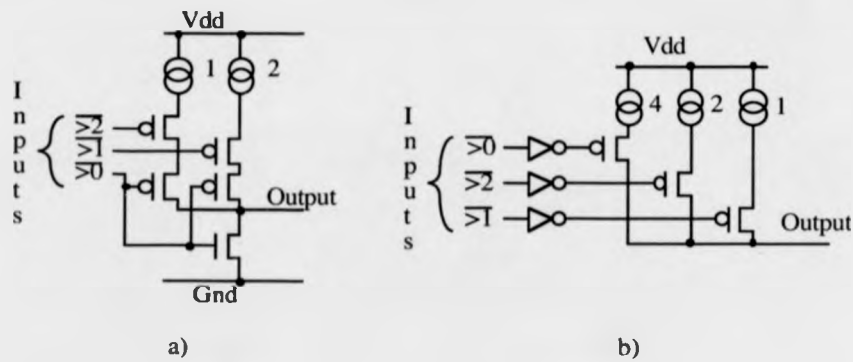


Figure 4.13. Encoders for a modulo 4 product. a) grounded output b) positive current.

The decoder for the positive current encoding is shown in figure 4.14. It is very similar to the grounded output decoder in terms of numbers of thresholds, and gates. The maximum threshold is the same as the grounded output version, since all the currents above 6.5 current units indicate a zero output.

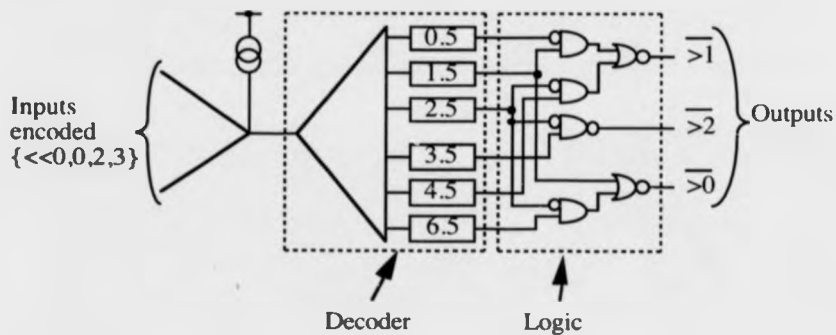


Figure 4.14. A positive current modulo 4 product decoder.

One major area in which the SUM and PRODUCT building blocks are likely to be more efficient than MAX and MIN circuits is in the design of arithmetic units. For example, a multiplier will typically be constructed from a set of partial product generators and adders to sum the partial products together. If carry circuitry is added to the SUM and PRODUCT building blocks, then a multiplier can be constructed from them. The two inputs are put into an array of partial product generators, and the outputs of these generators are then summed to form the product. Two sets of adders are needed in each row because the output from the partial product generator has a carry output which it would not have in the binary case. The array cell is shown in figure 4.15 in block form.

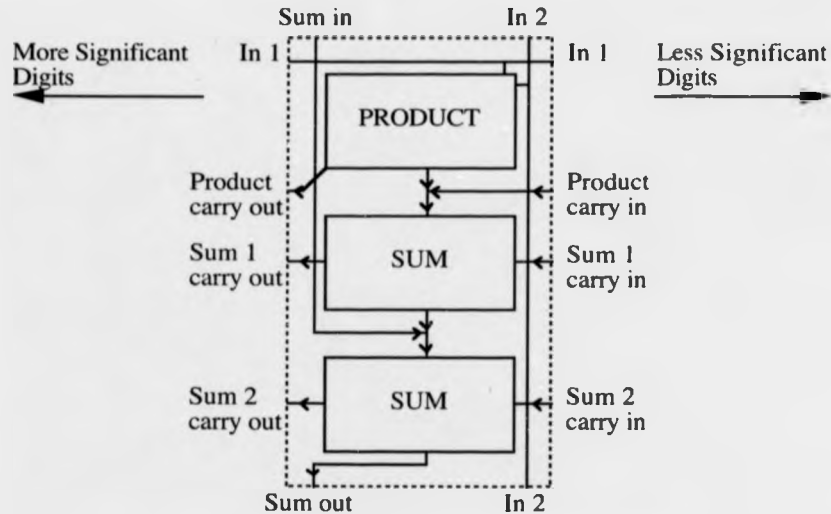


Figure 4.15. A multiplier cell block diagram.

It should be noted, that the input digits IN1 and IN2 must be voltage mode signals. The encoding to a current mode log encoded signal must be made at the multiplier cell. Either a pair of binary signals could be used, or a four level voltage mode encoding could be used as described in an earlier chapter.

#### 4.4. Current mode MVL libraries

When laying out larger circuits that use the building blocks already described, more than one encoding of a particular output, or more than one copy of an encoding may be needed. This is because of the unity fan-out of current mode circuits. A library could be designed so as to have each function as, an input section, and a set of output sections, one output section for each encoding type. It should be possible to fit any number of these output sections onto a single input section. Figure 4.16 shows a schematic for this style of library. The variables are shown with the way they have been encoded in brackets after the variable name. The decoder, and logic are in the SUM or PRODUCT sections, and the encoders (marked linear, and log) are simply slotted on the end. Encoding is logarithmic to perform a product, and a linear manner to perform a sum. This system makes use of the fact that the logic section output is voltage mode, and hence has an infinite fan-out, allowing it to drive as many encoders as necessary. The schematic shown is a logic circuit that has been described in modulo 4 sum and product form. MAX, MIN and COMPLEMENT circuits could also be used.

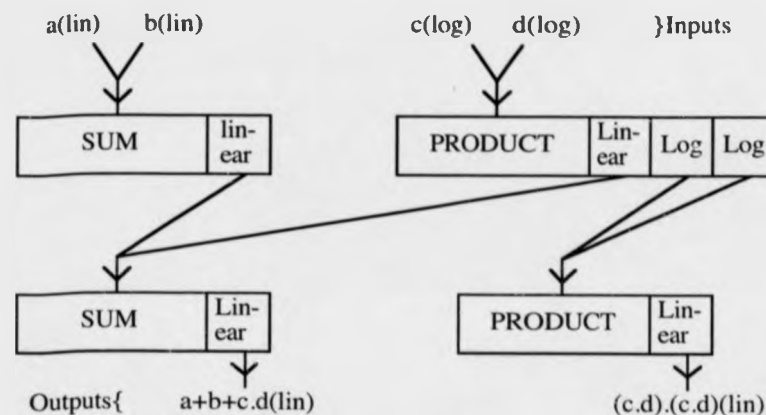


Figure 4.16. An example schematic of a current mode circuit.

Table 4.24 shows the minimum requirements in terms of input and output functions required for a full logic set for both SUM and PRODUCT, and MAX and MIN logic forms. The binary converters allow the circuits to interface easily with conventional



voltage mode binary circuits. Only one type of converter is needed when decoding to binary, since an encoder will have to be assigned to that particular output anyway.

Logic type	Encoders	Decoders
SUM, PRODUCT	Linear (SUM) Log (PRODUCT) Binary to Linear Binary to Log	SUM PRODUCT Linear to binary
MIN, MAX	Max encoding Complemented Max encoding Binary to Max Binary to complemented Max	MAX Max encoding to binary

Table 4.24. The minimum cells required for a library of different logic types.

It will probably be beneficial to individual library designers to incorporate their own special library cells that will aid efficient implementation of the particular operation that may be required by a design.

#### 4.4.1. Abutment

In table 4.24, an assumption has been made. The assumption is that one output encoder can be used in conjunction with a number of decoder and logic circuits. For this to be practical, there needs to be an interface between the two that is structured enough to allow the construction of many interlocking cells that fit easily and efficiently together. Typically, library cells are designed so that they fit neatly next to each other in a row, with the power rails, and possibly clock signals running down the row. All other connections are made from the top and bottom of the cell. With the current mode MVL circuits, the decoders, and encoders need to be physically next to each other, and so it makes sense for the lines connecting them to travel along the row. Indeed, as a number of encoders may be connected to the same decoder, the lines should pass right through the encoder cells. Figure 4.17 shows the perimeter of a typical decoder, and encoder using this design style. To prevent the lines through the encoders of two adjacent circuits from touching (which could happen if one circuit was mirrored about the vertical axis), it may be necessary to have a short dummy cell to place between adjacent circuits. This dummy cell will pass the power lines along

voltage mode binary circuits. Only one type of converter is needed when decoding to binary, since an encoder will have to be assigned to that particular output anyway.

Logic type	Encoders	Decoders
SUM, PRODUCT	Linear (SUM) Log (PRODUCT) Binary to Linear Binary to Log	SUM PRODUCT Linear to binary
MIN, MAX	Max encoding Complemented Max encoding Binary to Max Binary to complemented Max	MAX Max encoding to binary

Table 4.24. The minimum cells required for a library of different logic types.

It will probably be beneficial to individual library designers to incorporate their own special library cells that will aid efficient implementation of the particular operation that may be required by a design.

#### 4.4.1. Abutment

In table 4.24, an assumption has been made. The assumption is that one output encoder can be used in conjunction with a number of decoder and logic circuits. For this to be practical, there needs to be an interface between the two that is structured enough to allow the construction of many interlocking cells that fit easily and efficiently together. Typically, library cells are designed so that they fit neatly next to each other in a row, with the power rails, and possibly clock signals running down the row. All other connections are made from the top and bottom of the cell. With the current mode MVL circuits, the decoders, and encoders need to be physically next to each other, and so it makes sense for the lines connecting them to travel along the row. Indeed, as a number of encoders may be connected to the same decoder, the lines should pass right through the encoder cells. Figure 4.17 shows the perimeter of a typical decoder, and encoder using this design style. To prevent the lines through the encoders of two adjacent circuits from touching (which could happen if one circuit was mirrored about the vertical axis), it may be necessary to have a short dummy cell to place between adjacent circuits. This dummy cell will pass the power lines along

down the row, but nothing else. It is also important to define precisely, what lines will be where at the interface of the encoder and the decoder. This allows all the encoders to line up with all the decoders.

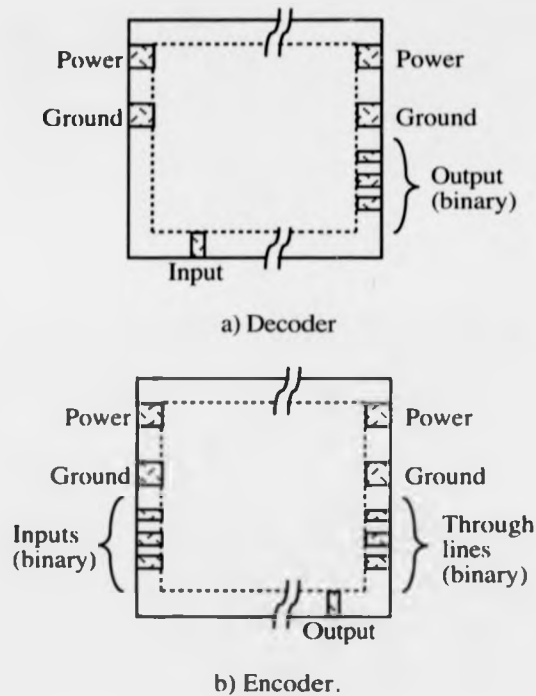


Figure 4.17. The perimeter of library cells.

The choice of what exactly the signals mean is also important. It must be standardised for the system to work, but it must also allow easy implementation of the encoder functions. The  $\overline{>0}$ ,  $\overline{>1}$ , and  $\overline{>2}$  signals have been the choice so far in this chapter. This is because it makes linear encoders, and decoders easy to design, and because other encoders seem to map well to this set of functions. The value of the multiple valued signal for given individual binary levels is shown in table 4.25.

Value	$\overline{>0}$	$\overline{>1}$	$\overline{>2}$
0	1	1	1
1	0	1	1
2	0	0	1
3	0	0	0

Table 4.25. A suggested decoder to encoder signal interface.

It is possible to use two lines instead of three, but this makes both the logic in the decoder, and the encoder more complex. So as the circuits are physically separated by only the smallest of possible distances, it is probably more area efficient to use three lines.

### 4.5. MVL circuits in VLSI

The design of VLSI circuit components is difficult and time consuming. Since a large number of the design tools available are only written to cope with binary signals, the design of an MVL circuit must be done as an analogue circuit. Some of the design tools for analogue circuits are considerably slower than their binary counterparts. For this reason, it is not practical to design large MVL circuits at present.

This section describes the design and testing of three MVL circuits on two test chips. The circuits designed were a current mode modulo 4 sum and product circuit, and a voltage mode binary full adder with a quaternary internal node. Two of these circuits have been previously reported by others (the modulo 4 sum [Current87], and the adder [Schultz89]), and the purpose of fabricating them was to investigate the problems involved with the design and fabrication process.

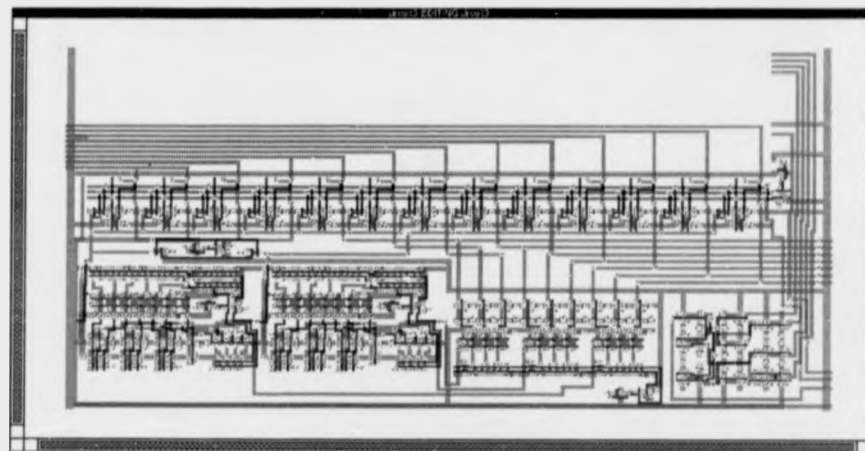
The test circuits were fabricated as additional circuitry on chips used for a final year course for undergraduates in VLSI design. Hence the number of output pins available was severely limited. These circuits were not designed as library cells, enough of the circuit was designed simply to test the current mode sections.

#### 4.5.1. The first test chip

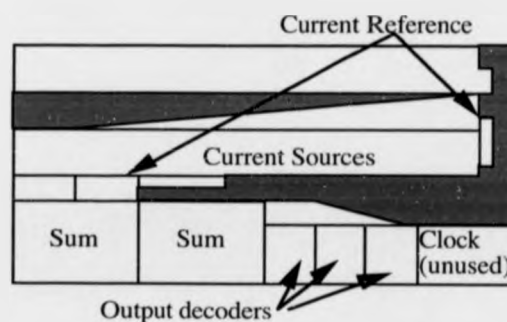
The first test chip was fabricated using a European Silicon Structures (ES2)  $2\mu\text{m}$  basic CMOS process. This process is tuned for binary digital circuits. Four circuits were actually fabricated on this test chip, but problems with the binary interface logic caused three of these to be untestable. The testable circuit was a pair of current mode modulo 4 sum circuits. The circuits had additional carry outputs, and

#### 4.5: MVL circuits in VLSI

were linked together by the carry line, in effect forming a 4 bit adder. The current inputs were formed by 9 inputs to 1 and 2 current unit switched sources. These were summed together to give an input to both of the sum circuits. The current unit used was  $10\mu\text{A}$ , and offset currents were used to speed up the sum circuits. The test circuit is shown in figure 4.18.



a)

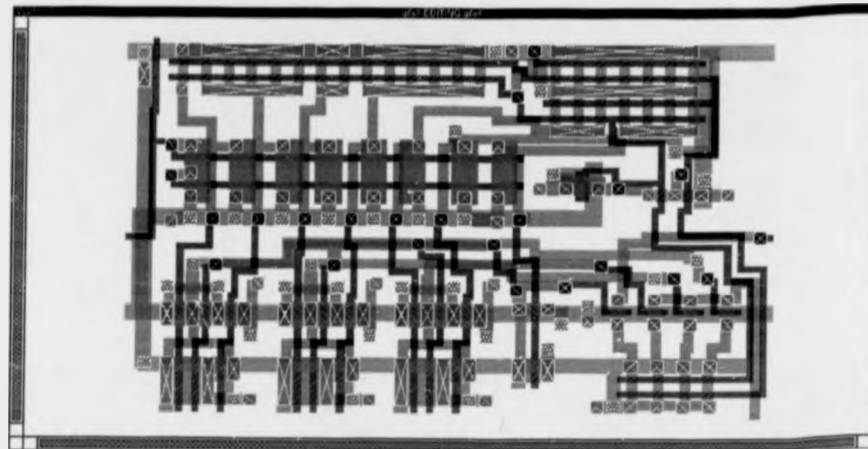


b)

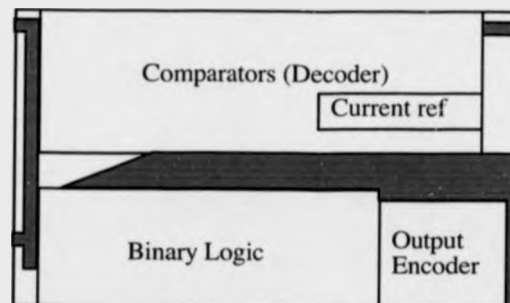
Figure 4.18. The first test chip circuit. a) Circuit layout b) floorplan.

It was not possible to have any analogue connections to the circuit, and so all the inputs and outputs were voltage mode binary signals. This was unfortunate, since it vastly reduced the observability of the circuits, and reduced the number of useful tests possible considerably. The current mirrors used were of the cascode type, and unit

transistor design was employed to increase relative accuracy between the current sources. The modulo 4 sum circuit is shown in figure 4.19.



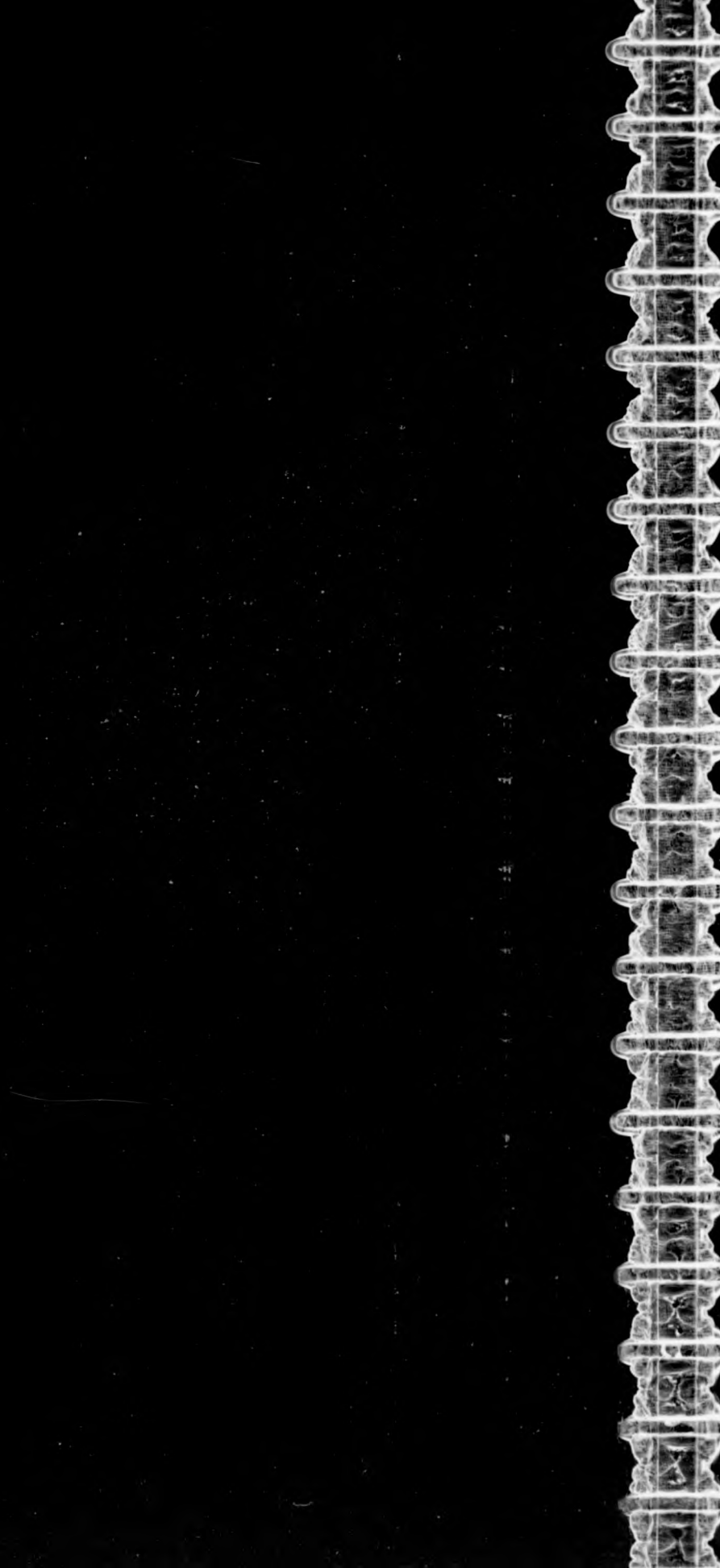
a)



b)

Figure 4.19. Current mode modulo 4 sum. a) Layout, b) floorplan.

The devices were tested, and some but not complete functionality was observed. The different circuits each had their own reference voltage generators, which could potentially cause different parts of the circuit to be producing different current levels. However, since the main errors occurred in the majority of the devices, it is unlikely that transistor parameter variation across the circuit caused these errors. A more likely cause is that the parameters of the N and P type FETs used for the design were different to those on the actual chip fabricated. There are two possible explanations for this:



1. The chips fabricated were fabricated in such a way that they did not have nominal characteristics of the process. This is possible since the upper and lower bounds of the process parameters (the 'skew' parameters) were not tested in the design.
2. The model parameters provided, produce accurate timing results for binary circuits, but do not accurately model the analogue behaviour of the transistors.

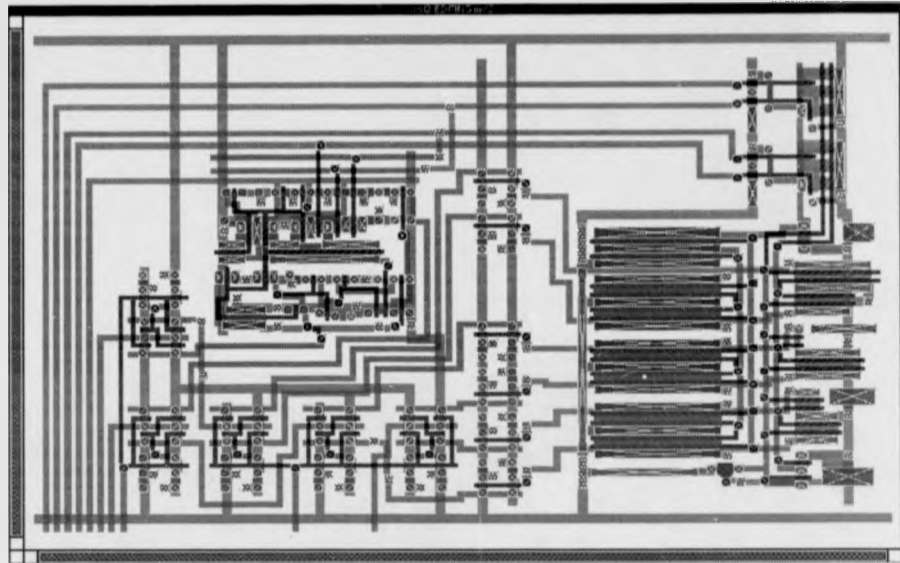
One error that gave insight into the problems with the circuit was the fact that the output circuits never registered an output level of 3 current units. This means that a 3 current unit signal fed to a 2.5 unit threshold circuit was not switching the threshold circuit. The most probable cause of this is an error in the size of the transistors relative to each other. However, a simulation showed the sizes to be correct, and so the simulation was probably at fault. Since the simulator used (SPICE) is well known and highly regarded in the VLSI design community, the only remaining component of the simulation, the model parameters, must be suspected. Despite these problems, the circuits did provide indications that 8 or so current units is not an unreasonable resolution to expect for a circuit of this type.

##### 4.5.2. The second test chip

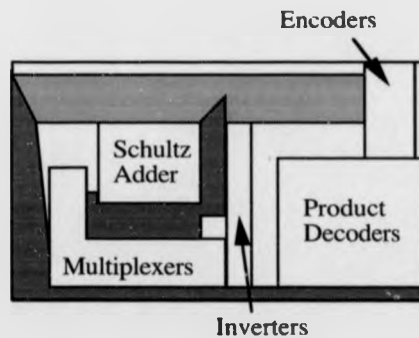
The second test chip was fabricated using an Orbit 1.5 $\mu$ m process. This was also a CMOS process aimed at binary circuits. Both a voltage mode and a current mode circuit were constructed in the process. The voltage mode circuit was the binary full adder described by Schultz [Schultz89]. The transistor sizes were modified so that the circuit operated properly in simulations. The current mode circuit was the encoders and decoders for the grounded output modulo 4 product circuit described earlier. The current unit for the current mode circuits was set at 10 $\mu$ A. The outputs of the circuits were multiplexed together to reduce the number of output pins required. The logic for the product circuit was not included since this section is binary and thus easy to design. Instead, the decoder threshold outputs are fed to inverters that buffer the signal, and then they are fed via the multiplexers to the output. The circuit fabricated on the second test chip is shown in figure 4.20. The mirrors in the current mode



circuit are cascode mirrors, and the N-type mirrors were extended to reduce errors as much as possible.



a)



b)

Figure 4.20. The second test circuit.

The current encoders used in this circuit are not exactly the same as those described earlier. Only two inputs are used, but the 4 possible output values are obtainable. The layout of the current encoder is shown in figure 4.21a, and the circuit schematic is given in figure 4.21b. The N-type transistors only need to be unit sized transistors to sink enough current to form a grounded encoder output. These circuits are not

intended to be used in a library, but simply to allow the form of circuit to be tested. There was the opportunity with the second test circuit to have an unbuffered output, and so the central quaternary voltage mode signal from the adder was connected to an external pin on the chip. This allowed the analogue voltages on this node to be measured, and checked against the simulated values.

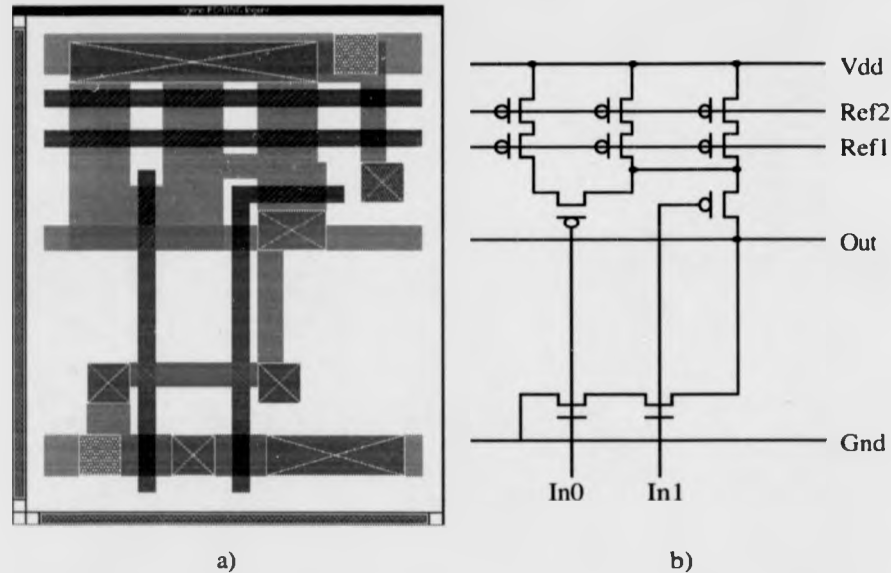


Figure 4.21. The grounded output, 2 input encoder. a) Layout, b) schematic.

Because the second test chip had an analogue output, it was possible to get much more certain results about the differences between the models used in simulation, and the actual circuits that were fabricated. The simulation results were in fact very different to the results actually measured. The voltage mode adder quaternary internal node voltages bore very little resemblance to the simulations. From these test results, the only conclusion that can be drawn is that the simulation models for MVL circuits must be as of as high a standard of accuracy as the models used in analogue simulation. The current mode circuit showed clearly that the grounded output technique works. The grounded signal was easily distinguishable. The other signal levels suffered similar problems to the current mode circuit on the first test chip, again pointing to problems with the models. The models used were investigated, and they

intended to be used in a library, but simply to allow the form of circuit to be tested. There was the opportunity with the second test circuit to have an unbuffered output, and so the central quaternary voltage mode signal from the adder was connected to an external pin on the chip. This allowed the analogue voltages on this node to be measured, and checked against the simulated values.

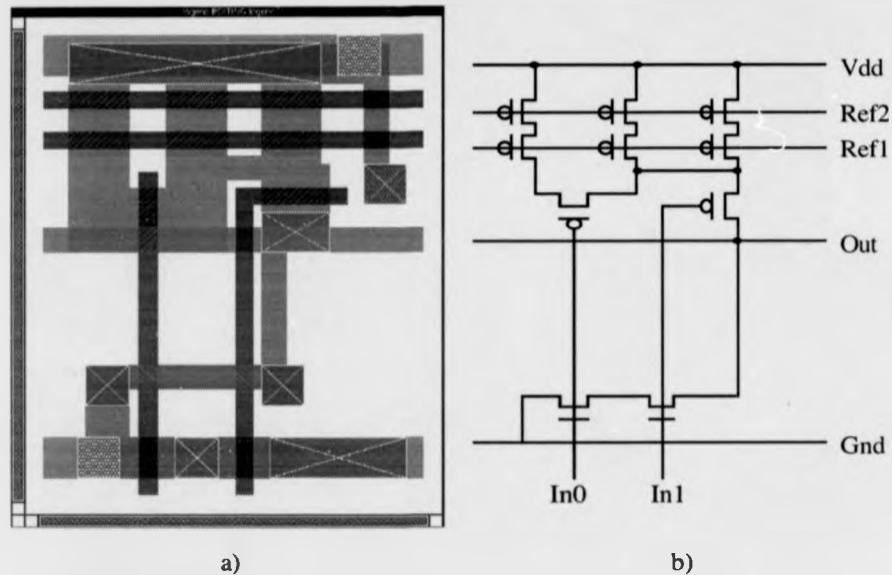


Figure 4.21. The grounded output, 2 input encoder. a) Layout, b) schematic.

Because the second test chip had an analogue output, it was possible to get much more certain results about the differences between the models used in simulation, and the actual circuits that were fabricated. The simulation results were in fact very different to the results actually measured. The voltage mode adder quaternary internal node voltages bore very little resemblance to the simulations. From these test results, the only conclusion that can be drawn is that the simulation models for MVL circuits must be as of as high a standard of accuracy as the models used in analogue simulation. The current mode circuit showed clearly that the grounded output technique works. The grounded signal was easily distinguishable. The other signal levels suffered similar problems to the current mode circuit on the first test chip, again pointing to problems with the models. The models used were investigated, and they

are designed for binary simulation. Better models are clearly the priority, and for practical commercial circuits, the circuits must operate over the full range of the process parameters, and the full temperature and voltage supply range.

#### 4.6. Summary

This chapter has described a methodology for the design of uni-directional current mode MVL circuits that could be easily extended to bi-directional circuits, and has potential for the design of optical logic circuits. The design method has been discussed in detail, and although this methodology is by no means perfect, it gives a good insight into how to design such circuits, and can provide very encouraging results. Single input functions can be implemented as the output stage of a circuit, with little or no overhead. The MAX, MIN and COMPLEMENT multiple valued logical form can be created most efficiently by using just a MAX gate with normal or complemented outputs. This is because the MIN gate is far less easy to implement. Modulo 4 sum and product circuits have also been designed. These circuits are less efficiently implemented because of the modulus effect, but in arithmetic applications they are the obvious basic building blocks. Libraries of current mode MVL circuits can be built up, and if the guidelines given in this chapter are followed, the efficiency of these designs in implementing larger functions should be good.

Despite the trends in CMOS processes, circuits such as the MAX function require so little in terms of resolution, that these circuits may well be viable until CMOS is finally replaced by a new commercial technology such as optical logic. At this point the concepts behind the circuits presented here could be easily transferred across. These designs and concepts allow larger circuits to be considered in a general sense for current mode MVL rather than just for individual applications.

# 5

## Function evaluation in many dimensions.

### 5.1. Introduction

In a previous chapter, the CORDIC algorithm has been described in two, three and four dimensions [Volder59, Delosme90]. There are other ways to operate the CORDIC algorithm in three and more dimensions other than the method already shown. A three dimensional algorithm is developed and this is then generalised to take account of the work of others already shown on the two dimensional algorithm. Following this, a new two dimensional CORDIC system which has a unit scale factor is also shown. A comparison of the various algorithms presented in this chapter is given, which includes a comparison relative to the methods used by others. Finally the layout of a three dimensional pipelined redundant arithmetic processor is described.

### 5.2. CORDIC in more than two dimensions

In this section, the theory behind multi-dimensional vector rotations will be considered. Starting with three dimensions, the CORDIC algorithm is generalised to  $N$  dimensions. It is shown that a three dimensional CORDIC algorithm can be used as a two dimensional algorithm eliminating the need for a scaling factor operation. The unit scale factor is achieved by rotating the vector in three dimensional space in a manner which scales its projection onto the X-Y plane by the reciprocal of the overall scale factor. This new technique takes the same number of cycles as the standard

CORDIC algorithm. The system is shown to be entirely compatible with redundant number system implementations of the CORDIC algorithm and has only a marginally slower cycle time than the redundant system of Takagi [Takagi91]. The extensions given by Walther [Walther71] are not considered, although they could be incorporated into this later without much difficulty. The use of redundant [Takagi91] and high radix [Rodrigues81] number systems is considered, as are hybrid systems.

Unless other wise stated, it will be assumed that a fixed point fractional representation is being used for vector lengths.

### 5.2.1. 3D CORDIC

The CORDIC algorithm is a two dimensional vector rotator. Its function is based upon the fact that a rotation of a vector  $(x,y)$  by an angle  $\alpha$  results in a new vector:

$$(x \cos \alpha - y \sin \alpha, y \cos \alpha + x \sin \alpha) \quad (1)$$

Hence the new vector can be represented by the old vector and a simple function of the rotation angle. In three dimensions, the rotation is defined by two variables. There are a number of choices for these two variables, but the most well known is to have one value representing the rotation away from the z-axis and one representing the rotation about the z-axis. This is a spherical coordinate system. However there are many other choices available such as that taken by Delosme and Hsaio [Delosme90]. The problem with a spherical coordinate system is that a vector  $(x,y,z)$  which is rotated by the angles  $A$  and  $B$  around and away from the z-axis respectively, cannot easily be represented by  $x, y, z$  and the sines or cosines of  $A$  and  $B$ . Ways around this problem are now examined.

#### 5.2.1.1. Basis of extension to three dimensions

A vector in three dimensional space is shown in figure 5.1. It has Cartesian coordinates  $(X, Y, Z)$  and spherical coordinates  $(R, \theta, \phi)$ .

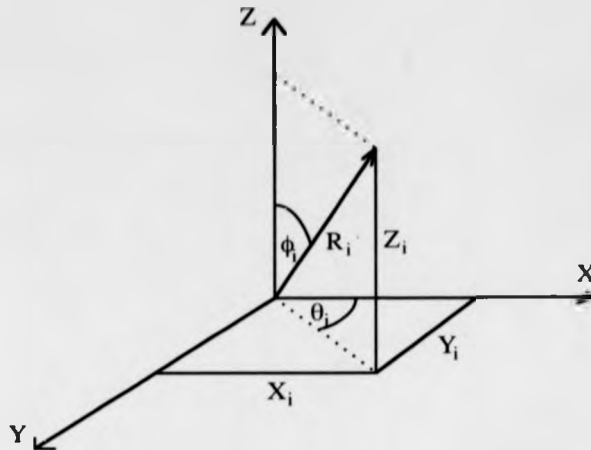


Figure 5.1. A vector in three dimensional space.

The vector can be rotated by an angle  $\alpha_i$  around the z-axis and by an angle  $\beta_i$  away from the z-axis to become a new vector which has Cartesian coordinates  $(X_{i+1}, Y_{i+1}, Z_{i+1})$  and spherical coordinates  $(R_i, \theta_i + \alpha_i, \phi_i + \beta_i)$ . In this case, the relationship between the Cartesian and spherical coordinates of the two vectors are shown in equations 2 to 7.

$$X_i = R_i \cos \theta_i \sin \phi_i \quad (2)$$

$$Y_i = R_i \sin \theta_i \sin \phi_i \quad (3)$$

$$Z_i = R_i \cos \phi_i \quad (4)$$

$$X_{i+1} = R_i \cos(\theta_i + \alpha_i) \sin(\phi_i + \beta_i) \quad (5)$$

$$Y_{i+1} = R_i \sin(\theta_i + \alpha_i) \sin(\phi_i + \beta_i) \quad (6)$$

$$Z_{i+1} = R_i \cos(\phi_i + \beta_i) \quad (7)$$

It is preferable to form an iteration such that allows  $(X_{i+1}, Y_{i+1}, Z_{i+1})$  to be found from knowing  $(X_i, Y_i, Z_i)$  and the sines and cosines of the rotation angles in the same manner as the two dimensional algorithm. However, when equations 5 to 7 are expanded, the result is:

$$\begin{aligned} X_{i+1} = & X_i \cos \alpha_i \cos \beta_i - Y_i \sin \alpha_i \cos \beta_i \\ & + R_i \cos \theta_i \cos \phi_i \cos \alpha_i \sin \beta_i - R_i \sin \theta_i \cos \phi_i \sin \alpha_i \sin \beta_i \end{aligned} \quad (8)$$

$$Y_{i+1} = Y_i \cos \alpha_i \cos \beta_i + X_i \sin \alpha_i \cos \beta_i + R_i \sin \theta_i \cos \phi_i \cos \alpha_i \sin \beta_i + R_i \cos \theta_i \cos \phi_i \sin \alpha_i \sin \beta_i \quad (9)$$

$$Z_{i+1} = Z_i \cos \beta_i - R_i \sin \phi_i \sin \beta_i \quad (10)$$

These are not fully defined in terms of the original vector. However, it is possible to define the following set of scalars:

$$U_i = R_i \cos \theta_i \cos \phi_i \quad (11)$$

$$V_i = R_i \sin \theta_i \cos \phi_i \quad (12)$$

$$W_i = R_i \sin \phi_i \quad (13)$$

$$U_{i+1} = R_i \cos(\theta_i + \alpha_i) \cos(\phi_i + \beta_i) \quad (14)$$

$$V_{i+1} = R_i \sin(\theta_i + \alpha_i) \cos(\phi_i + \beta_i) \quad (15)$$

$$W_{i+1} = R_i \sin(\phi_i + \beta_i) \quad (16)$$

These, in combination with equations 2 to 7 gives a set that will iterate. The  $W$  variable is the magnitude of the vector projected onto the x-y plane, but the physical meaning of the  $U$  and  $V$  variables is unclear. Expanding the left hand side of equations 5 to 7 and 14 to 16 and substituting from equations 2 to 7 and 11 to 13 wherever possible, the following set of iterating equations is found:

$$U_{i+1} = U_i \cos \alpha_i \cos \beta_i - X_i \cos \alpha_i \sin \beta_i - V_i \sin \alpha_i \cos \beta_i + Y_i \sin \alpha_i \sin \beta_i \quad (17)$$

$$V_{i+1} = V_i \cos \alpha_i \cos \beta_i - Y_i \cos \alpha_i \sin \beta_i + U_i \sin \alpha_i \cos \beta_i - X_i \sin \alpha_i \sin \beta_i \quad (18)$$

$$W_{i+1} = W_i \cos \beta_i + Z_i \sin \beta_i \quad (19)$$

$$X_{i+1} = X_i \cos \alpha_i \cos \beta_i + U_i \cos \alpha_i \sin \beta_i - Y_i \sin \alpha_i \cos \beta_i - V_i \sin \alpha_i \sin \beta_i \quad (20)$$

$$Y_{i+1} = Y_i \cos \alpha_i \cos \beta_i + V_i \cos \alpha_i \sin \beta_i + X_i \sin \alpha_i \cos \beta_i + U_i \sin \alpha_i \sin \beta_i \quad (21)$$

$$Z_{i+1} = Z_i \cos \beta_i - W_i \sin \beta_i \quad (22)$$



This set of equations divides any rotation into a set of smaller rotations in the same way as the two dimensional CORDIC algorithm. This is possible since there is now a definition of the rotated vector in terms of the original vector, and the rotations  $\alpha$  and  $\beta$ . The rotation can therefore be divided into a set of smaller rotations.  $\alpha_i$  and  $\beta_i$  have the same magnitude, but their signs may differ. This is simply for convenience. It also means that the accuracy of the algorithm is the same in all dimensions at a given iteration  $i$ . As with the two dimensional CORDIC algorithm,  $\alpha_i = a_i \cdot \arctan 2^{-i}$  and  $\beta_i = b_i \cdot \arctan 2^{-i}$  are chosen, where  $a_i$  and  $b_i$  are  $\in \{-1, 1\}$ . From this, equations 23 to 26 can be derived for the sine and cosine of  $\alpha_i$  and  $\beta_i$ . The  $a_i$  and  $b_i$  factors disappear from the cosine equations because  $\cos \delta = \cos -\delta$ , whereas they are present in the sine equations because  $\sin \delta = -\sin -\delta$ .

$$\cos \beta_i = \frac{1}{\sqrt{1 + 2^{-2i}}} \quad (23)$$

$$\sin \beta_i = \frac{b_i 2^{-i}}{\sqrt{1 + 2^{-2i}}} \quad (24)$$

$$\cos \alpha_i = \frac{1}{\sqrt{1 + 2^{-2i}}} \quad (25)$$

$$\sin \alpha_i = \frac{a_i 2^{-i}}{\sqrt{1 + 2^{-2i}}} \quad (26)$$

The divisor of all these equations is a constant for each iteration step. We will call this constant  $k_i$ . Substituting equations 23 to 26 into equations 17 to 22, gives:

$$U_{i+1} = \frac{1}{k_i^2} (U_i - X_i b_i 2^{-i} - V_i a_i 2^{-i} + Y_i a_i b_i 2^{-2i}) \quad (27)$$

$$V_{i+1} = \frac{1}{k_i^2} (V_i - Y_i b_i 2^{-i} + U_i a_i 2^{-i} - X_i a_i b_i 2^{-2i}) \quad (28)$$

$$W_{i+1} = \frac{1}{k_i} (W_i + Z_i b_i 2^{-i}) \quad (29)$$

$$X_{i+1} = \frac{1}{k_i^2} (X_i + U_i b_i 2^{-i} - Y_i a_i 2^{-i} - V_i a_i b_i 2^{-2i}) \quad (30)$$

$$Y_{i+1} = \frac{1}{k_i} (Y_i + V_i b_i 2^{-i} + X_i a_i 2^{-i} + U_i a_i b_i 2^{-2i}) \quad (31)$$

$$Z_{i+1} = \frac{1}{k_i} (Z_i - W_i b_i 2^{-i}) \quad (32)$$

The scale factor for  $Z_{i+1}$  and  $W_{i+1}$  is different to that of  $U_{i+1}$ ,  $V_{i+1}$ ,  $X_{i+1}$  and  $Y_{i+1}$ . This is not a problem since the two sets of variables do not interact in any way. These equations can be used to rotate a three dimensional vector. The equations 27 to 32 are iterated ignoring  $k_i$ . We can pre-scale the inputs, or post-scale the outputs by the overall scale factor  $K$ , for  $Z$  and  $W$  and  $K^2$  for  $U$ ,  $V$ ,  $X$  and  $Y$ , where:

$$K = \prod_{i=0}^{n-1} \sqrt{1 + 2^{-2i}} \quad (33)$$

#### 5.2.1.2. Convergence and accuracy

In expanding the CORDIC algorithm to more than two dimensions, it is not unreasonable to assume that obtaining an iteration that gives convergent results will be more difficult than in the two dimensional case. Indeed that is exactly what Delosme and Hsaio found [Delosme90]. They used a system similar to that used by Walther for his hyperbolic rotations of repeating a number of iterations. The repetitions roughly follow a  $2k-1$  sequence, but not exactly. However, when the rotation system already described is used, the planes of rotation are orthogonal to each other and hence can be considered separately. The planes of rotation for a polar coordinate system are shown in figure 5.2.

$$Y_{i+1} = \frac{1}{k_i^2} (Y_i + V_i b_i 2^{-i} + X_i a_i 2^{-i} + U_i a_i b_i 2^{-2i}) \quad (31)$$

$$Z_{i+1} = \frac{1}{k_i} (Z_i - W_i b_i 2^{-i}) \quad (32)$$

The scale factor for  $Z_{i+1}$  and  $W_{i+1}$  is different to that of  $U_{i+1}$ ,  $V_{i+1}$ ,  $X_{i+1}$  and  $Y_{i+1}$ . This is not a problem since the two sets of variables do not interact in any way. These equations can be used to rotate a three dimensional vector. The equations 27 to 32 are iterated ignoring  $k_i$ . We can pre-scale the inputs, or post-scale the outputs by the overall scale factor  $K$ , for  $Z$  and  $W$  and  $K^2$  for  $U$ ,  $V$ ,  $X$  and  $Y$ , where:

$$K = \prod_{i=0}^{n-1} \sqrt{1 + 2^{-2i}} \quad (33)$$

#### 5.2.1.2. Convergence and accuracy

In expanding the CORDIC algorithm to more than two dimensions, it is not unreasonable to assume that obtaining an iteration that gives convergent results will be more difficult than in the two dimensional case. Indeed that is exactly what Delosme and Hsaio found [Delosme90]. They used a system similar to that used by Walther for his hyperbolic rotations of repeating a number of iterations. The repetitions roughly follow a  $2k-1$  sequence, but not exactly. However, when the rotation system already described is used, the planes of rotation are orthogonal to each other and hence can be considered separately. The planes of rotation for a polar coordinate system are shown in figure 5.2.

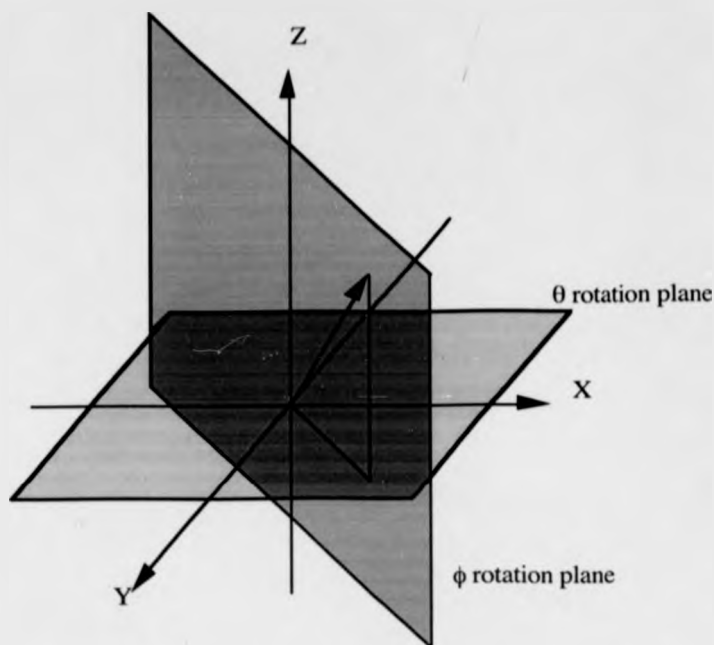


Figure 5.2. Planes of rotation for a polar coordinate system.

Each of the planes of rotation acts on its own, as a simple two dimensional vector rotation and so convergence is assured in just the same way as for the 2D case, except that both planes must be shown to converge. Since the case under investigation is that of circular rotations, the algorithm must converge as it is known that:

$$\frac{\tan^{-1} 2^{-i}}{\tan^{-1} 2^{-i-1}} \leq 2 \quad (34)$$

As with the two dimensional algorithm, there are two sources of errors, the rounding error and the angle approximation error. The iterations for  $W$  and  $Z$  form a two dimensional algorithm for a rotation in the  $\phi$  rotation plane and so the results given by Hu [Hu92] are still valid. The errors for  $U$ ,  $V$ ,  $X$  and  $Y$  however are different. There is an angle approximation error from both the  $\theta$  and  $\phi$  rotations and the rounding errors are different since they are propagated to the output differently.

The angle approximation error of the  $\theta$  and  $\phi$  rotations is  $\leq \tan^{-1} 2^{-n-1}$  in each plane. The planes are at right angles to each other, so the error of the two angles

combined is  $\sqrt{2}$  of the error of a single rotation. The error due to the angle approximation in any of the four variables will be less than:

$$\sqrt{2}R\sin(\tan^{-1}2^{-(n-1)}) \quad (35)$$

Where  $R$  is the vector magnitude. The worst case vector magnitude appears when the vector starts with unit magnitude. This vector is extended by the scaling factor during the iterations, but post scaling then removes this factor. For practical values of  $n$ ,  $2^{-(n-1)}$  is small enough that both the tangent and sine functions can be ignored. Hence the angle approximation error  $E_{angle}$  is given by:

$$E_{angle} \leq 2^{-(n-2)} \quad (36)$$

The rounding error is formed by observing that there is a rounding error at each iteration and that this error is modified by subsequent stages. Assuming an internal resolution of  $b$  bits, the maximum error from one rounding operation on the vector will be  $\leq 2^{-b} \cdot \sqrt{2}$ . The effect of propagating this error (at the  $i$ th iteration) to the final result can be found by considering the transfer matrix of the iteration in matrix form:

$$\frac{1}{1+2^{-2i}} \begin{bmatrix} 1 & -a_i 2^{-i} & -b_i 2^{-i} & a_i b_i 2^{-2i} \\ a_i 2^{-i} & 1 & -a_i b_i 2^{-2i} & -b_i 2^{-i} \\ b_i 2^{-i} & -a_i b_i 2^{-2i} & 1 & -a_i 2^{-i} \\ a_i b_i 2^{-2i} & b_i 2^{-i} & a_i 2^{-i} & 1 \end{bmatrix} \quad (37)$$

The error will be scaled by the determinant of this matrix multiplied by the factor given. Overall the rounding error will be the initial error scaled by the subsequent iterations:

$$\prod_{i=1}^{n-1} (1+2^{-2i})^3 \quad (38)$$

The error due to rounding in the final result  $E_{round}$  will therefore be the sum of the individual rounding errors, each propagated to the result of the final iteration and then post-scaled, i.e.:

$$E_{\text{round}} = \frac{2^{-b+0.5}}{\prod_{i=0}^{n-1} (1 + 2^{-2i})} \cdot \left[ 1 + \sum_{i=0}^{n-2} \left( \prod_{j=i+1}^{n-1} (1 + 2^{-2j})^3 \right) \right] \quad (39)$$

The output accuracy  $b_o$  (given in bits) can be found by the following equation:

$$b_o = -\log_2(E_{\text{angle}} + E_{\text{round}}) \quad (40)$$

The result of this analysis is shown in table 5.1. The table shows the output accuracy  $b_o$  in bits according to the number of iterations performed  $n$  and the number of bits used internally  $b$ . The table shows the region around  $b_o = 16$  bits. The right hand set of figures are taken from a bit level simulation of the algorithm and they show that the theoretical errors are very accurate, although there seems to be a small additional error source. This seems to be due to the post-scaling operation. The bit level simulation performed 10,000 random sized rotations on random vectors. Combinations where  $n > b + 1$  are not shown since the final iteration(s) would be pointless (the rotation angles would be smaller than those representable with the given accuracy). Indeed, during these extra iterations, errors could increase.

		b				
		20	21	22	23	24
n	17	14.4/14	14.8/14	15.1/14	15.3/14	15.4/14
	18	14.7/14	15.3/15	15.8/15	16.1/15	16.3/15
	19	14.9/14	15.6/15	16.3/15	16.8/16	17.1/16
	20	14.9/15	15.8/15	16.6/16	17.3/16	17.7/17
	21	14.9/15	15.9/16	16.8/16	17.6/17	18.2/18
	22	n/a	15.9/16	16.8/16	17.7/17	18.5/18
	23	n/a	n/a	16.8/16	17.8/18	18.6/18
	24	n/a	n/a	n/a	17.8/17	18.7/18

Table 5.1. The result accuracy as a function of the internal resolution and number of iterations used. Theoretical/measured results.

### 5.2.1.3. Initial conditions

When starting, at  $i = 1$ , not only the Cartesian coordinates  $X_1$ ,  $Y_1$  and  $Z_1$  need to be known, but also  $U_1$ ,  $V_1$  and  $W_1$ . These would need to be calculated, which is a significant overhead. One way around this is to constrain the start vector to

lie on an axis, as this would cause  $U_i$ ,  $V_i$  and  $W_i$  to be trivial. Otherwise, it would be necessary to pre-calculate  $U_i$ ,  $V_i$  and  $W_i$  and then store them between operations. This would double the storage requirement for a vector. In addition, it should be noted that it is not possible to sum two vectors simply by summing the corresponding variables in each vector representation. This can be seen by considering  $W$  which is the magnitude of the projection of the vector onto the x-y plane. The magnitude of the sum of two vectors is not necessarily equal to the sum of the individual vector magnitudes, as shown in figure 5.3.

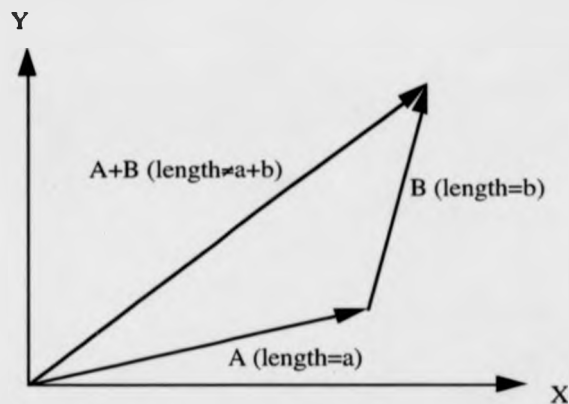


Figure 5.3. The sum of two vectors on the x-y plane.

#### 5.2.1.4. Application to scaleless 2D

It can be seen from the above discussion, that this 3D CORDIC algorithm suffers from a problem at the first iteration. However, when this system is used as a unit scale factor 2D CORDIC, the effect disappears. The X-Y plane is used for the two dimensional rotation. The vector is rotated out of the plane to reduce its projected length on the plane. The rotation out of the plane is chosen to have the opposite effect to that of the vector extension and hence they cancel each other out, giving a unit scale factor. This is shown in figure 5.4. Assume a start vector A, this rotates in the 2D case to B which is extended. Instead, the vector is rotated to C. This vector is also extended, but the projection on the X-Y plane (D) is of the same length as A and rotated by the correct angle in the X-Y plane.

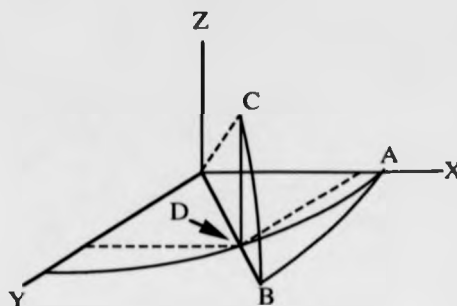


Figure 5.4. Rotating in 3D to remove the 2D scale factor.

Starting with  $Z_1 = 0$ ,  $U_1$  and  $V_1$  will also be 0 since  $\phi_1$  must be  $\frac{\pi}{2}$ . Also, to calculate  $X_n$  and  $Y_n$ , neither  $W_n$  nor  $Z_n$  need be calculated and hence  $W_1$  and  $Z_1$  are not needed. All that needs to be done is to set  $X_1$  and  $Y_1$  to the initial vector and,  $U_1$  and  $V_1$  to 0. To perform a rotation with no scale factor,  $\theta$  is rotated by the required vector rotation and  $\phi$  is rotated by the angle  $\arccos \frac{1}{K^2}$ . The overall effect of the two rotations is a single unscaled rotation.

It is worth noting at this point, that the rotation out of the plane is a constant value and so the signs of the rotations  $b_i$  can be pre-calculated. This makes the form of the iteration similar to that given by Ahmed [Ahmed82]. Indeed, the method described here is a more general version of that given by Ahmed.

#### 5.2.1.5. Redundant numbers

Redundant number arithmetic can be incorporated into this scaleless algorithm without difficulty. To use the double rotation method, the rotations in each dimension are split into two sub-rotations as shown in figure 5.6. Figure 5.5 shows the 3D CORDIC algorithm given earlier in the same form for comparison.



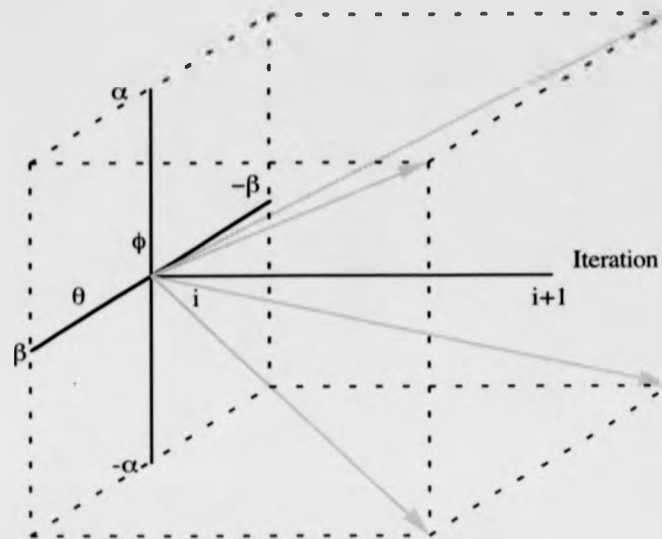


Figure 5.5. A 3D rotation

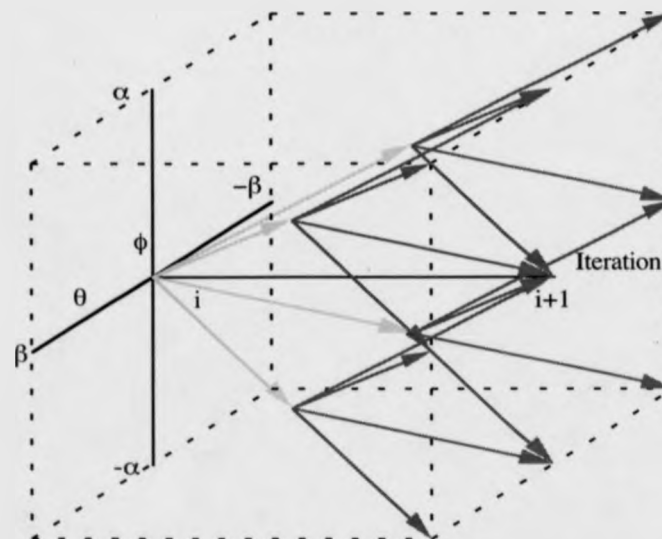


Figure 5.6. The double rotation method for a 3D rotation

It can be seen from figure 5.6, that it is possible to perform a positive, negative or no rotation in  $\theta$  and  $\phi$  independently. This means that the convergence is assured in the same way as in the two dimensional case. The selection of the rotation direction, can be made independently and in parallel for the two rotations. The iteration is now:

$$U_{i+1} = U_i S_1 - X_i S_2 - V_i S_3 + Y_i S_4 \quad (41)$$

$$V_{i+1} = V_i S_1 - Y_i S_2 + U_i S_3 - X_i S_4 \quad (42)$$

$$W_{i+1} = W_i (1 - b_i d_i 2^{-2i-2}) + Z_i ((b_i + d_i) 2^{-i-1}) \quad (43)$$

$$X_{i+1} = X_i S_1 + U_i S_2 - Y_i S_3 - V_i S_4 \quad (44)$$

$$Y_{i+1} = Y_i S_1 + V_i S_2 + X_i S_3 + U_i S_4 \quad (45)$$

$$Z_{i+1} = Z_i (1 - b_i d_i 2^{-2i-2}) - W_i ((b_i + d_i) 2^{-i-1}) \quad (46)$$

Where:

$$S_1 = 1 - (a_i c_i + b_i d_i) 2^{-2i-2} + a_i c_i b_i d_i 2^{-4i-4} \quad (47)$$

$$S_2 = (b_i + d_i) 2^{-i-1} - a_i c_i (b_i + d_i) 2^{-3i-3} \quad (48)$$

$$S_3 = (a_i + c_i) 2^{-i-1} - b_i d_i (a_i + c_i) 2^{-3i-3} \quad (49)$$

$$S_4 = (a_i + c_i) (b_i + d_i) 2^{-2i-2} \quad (50)$$

$a_i$ ,  $b_i$ ,  $c_i$  and  $d_i \in \{1, -1\}$  indicate the direction of the subrotations.  $a_i$  and  $c_i$  represent the rotation in the x-y plane,  $b_i$  and  $d_i$  represent the rotation out of the x-y plane. If the  $T$  variable represents the angle to be rotated about the Z axis (i.e. the rotation in the X-Y plane) and the  $R$  variable represents the angle to be rotated away from the Z axis, then  $a_i$ ,  $b_i$ ,  $c_i$  and  $d_i$  are given by the selection equations below. The iteration equations for  $T$  and  $R$  are also given:

$$a_i \quad c_i = \begin{cases} -1 & -1 & \text{if } [t_i^{i-1} t_i^i t_i^{i+1}] < 0 \\ 1 & -1 & \text{if } [t_i^{i-1} t_i^i t_i^{i+1}] = 0 \\ 1 & 1 & \text{if } [t_i^{i-1} t_i^i t_i^{i+1}] > 0 \end{cases} \quad (51)$$

$$b_i \quad d_i = \begin{cases} -1 & -1 & \text{if } [r_i^{i-1} r_i^i r_i^{i+1}] < 0 \\ 1 & -1 & \text{if } [r_i^{i-1} r_i^i r_i^{i+1}] = 0 \\ 1 & 1 & \text{if } [r_i^{i-1} r_i^i r_i^{i+1}] > 0 \end{cases} \quad (52)$$

Where  $t_i^j$  is the  $j$ th bit of  $T_i$  and similarly for  $R$ .

$$T_{i+1} = T_i - (a_i + c_i) \tan^{-1} 2^{-i-1} \quad (53)$$

$$R_{i+1} = R_i - (b_i + d_i) \tan^{-1} 2^{-i-1} \quad (54)$$

Bit level simulations of this algorithm have been used to assess the accuracy of the outputs for varying numbers of iterations and internal accuracies. Each combination of numbers of iterations  $n$  and internal accuracies  $b$  was simulated for 1000 random vectors, rotated through random angles. The results of these simulations is shown in table 5.2.

		b				
		19	20	21	22	23
n	17	14	14	14	15	14
	18	14	15	15	15	16
	19	14	15	16	17	16
	20	14	15	16	17	17
	21	n/a	15	16	17	18
	22	n/a	n/a	16	17	18
	23	n/a	n/a	n/a	17	18

Table 5.2. 3D Redundant CORDIC output accuracy in fractional bits for  $n$  iterations and  $b$  bits internal accuracy.

When the 3D CORDIC is to be used to create a scaleless 2D CORDIC, the rotation of  $\phi$  is known at the design stage and so the double rotation method suggested by Takagi [Takagi91], or the multiple engine system of Duprat [Duprat91] are not necessary for this rotation. If double rotation method is to be used for the rotation of  $\theta$ , then equations 55 to 60 will be used, where  $c_i$  and  $e_i$  are  $\in \{1, -1\}$ , each denoting the sign of a half rotation in the X-Y plane. These two half rotations combine to form a positive, a negative, or no rotation. When there is no rotation there is still an extension of the vector.

$$U_{i+1} = U_i S_1 - X_i S_1 b_i 2^{-i} - V_i S_2 2^{-i} + Y_i S_2 b_i 2^{-2i} \quad (55)$$

$$V_{i+1} = V_i S_1 - Y_i S_1 b_i 2^{-i} + U_i S_2 2^{-i} - Y_i S_2 b_i 2^{-2i} \quad (56)$$

$$X_{i+1} = X_i S_1 + U_i S_1 b_i 2^{-i} - Y_i S_2 2^{-i} - V_i S_2 b_i 2^{-2i} \quad (57)$$

$$Y_{i+1} = Y_i S_1 + V_i S_1 b_i 2^{-i} + X_i S_2 2^{-i} + U_i S_2 b_i 2^{-2i} \quad (58)$$

where:

$$S_1 = 1 - c_i e_i 2^{-2i-2} \quad (59)$$

and:

$$S_2 = \frac{c_i + e_i}{2} \quad (60)$$

The scale factor still needs to be calculated, to find the correcting rotation of the vector off the X-Y plane. The new scaling factor  $K$  in this case is:

$$K = \prod_{i=1}^n (1 + 2^{-2i-2}) \sqrt{1 + 2^{-2i}} \quad (61)$$

The rotation of  $\phi$  will be  $\arccos \frac{1}{K}$ . Note that this scale factor is different to that in the previous section.

So far the issue of making the decision of whether to rotate in a positive or negative direction at any particular stage has not been addressed. The case of rotations of  $\phi$  for scaleless 2D is simple because the angle through which  $B$  will be rotated is known in advance. Hence it is possible to pre-calculate the set of  $b_i$  so that :

$$B = \sum_{i=1}^n b_i \arctan 2^{-i} \quad (62)$$

Once calculated,  $b_i$  can be stored in a very small ROM ( $n$  bits). The calculation of the rotations in the X-Y plane (and away from the X-Y plane for the general 3D system) is done as Takagi suggested. The calculation of  $c_i$  and  $e_i$  is made by evaluating the most significant three digits of  $T_i$  at stage  $i$ . Then  $c_i$  and  $e_i$  are defined as follows:

$$c_i e_i = \begin{cases} -1 & 1 & \text{if } \begin{bmatrix} t_i^{i-1} & t_i^i & t_i^{i+1} \end{bmatrix} < 0 \\ 1 & -1 & \text{if } \begin{bmatrix} t_i^{i-1} & t_i^i & t_i^{i+1} \end{bmatrix} = 0 \\ 1 & 1 & \text{if } \begin{bmatrix} t_i^{i-1} & t_i^i & t_i^{i+1} \end{bmatrix} > 0 \end{cases} \quad (63)$$

Note that  $c_i$  indicates the direction of rotation and  $e_i$  represents whether the second rotation is in the same direction as the first (+1), or not (-1). The new value  $T_{i+1}$  can be found from equation 64.

$$T_{i+1} = T_i - c_i (e_i + 1) \arctan 2^{-i} \quad (64)$$

### 5.2.1.6. Impact of the 2D Scaleless CORDIC Algorithm

The equations for the 'scale factor'-less redundant arithmetic CORDIC system are more complex than the equations suggested by Takagi [Takagi91]. The number of terms is doubled. This would, however only lead to a 50% longer delay in the adder stage due to the increased number of inputs from 3 to 6, as shown in figure 5.7.

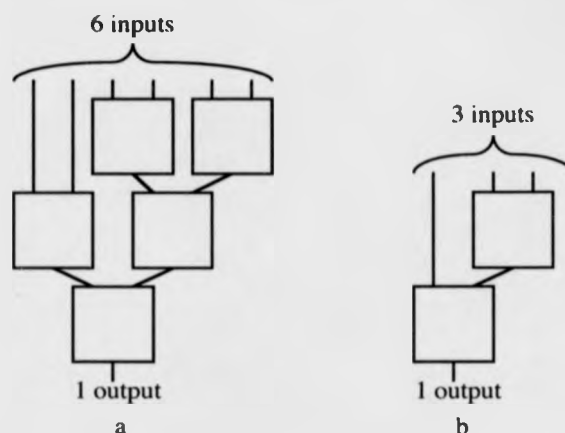


Figure 5.7. Example bit slice of redundant adder layouts for a) A unit scale factor system b) Takagi's suggestion.

As well as the increased need for addition, there would need to be two more registers for U and V and multiple shifters. The addition of more shifters and registers should not greatly affect the clock period of a hardware implementation of the new algorithm because the operations would be performed in parallel. We could use the redundant method suggested by Duprat [Duprat91] and this would result in an adder delay equal to that of the constant scale factor technique given by Takagi. It would also be possible to incorporate this new system into the unified algorithm of Walther [Walther71]. It should be noted however, that when using the unified algorithm, the scale factor is greater than 1 in the circular rotation case, equal to 1 in the linear case and less than 1 in the hyperbolic case. This means that a different correction is

required for each type of rotation. The obvious solution is to arrange for the two rotations to be of the same type. The sets of additions and subtractions necessary to compensate for the scale factor will be different in each case and trivial for the linear case.

### 5.2.1.7. Hybrid systems

The work of Timmermann et. al. and Ahmed [Timmermann89a, Ahmed82] that reduces the number of iterations required for a given accuracy can be extended to three dimensions. The technique relies on the first term of the Taylor series and therefore comes under the heading of a hybrid system.

The approximations used are:

$$\cos \delta \approx 1 \quad (65)$$

$$\sin \delta \approx \delta \quad (66)$$

From this it is possible to see that a final iteration needs to be added to the algorithm in rotation mode that uses these approximations. Clearly, if only  $(x, y, z)$  needs to be evaluated, then it is only necessary to do the final iteration for those equations. The equations for  $u$ ,  $v$  and  $w$ , are not given because it will not normally be necessary to store these variables. If they are required, it would be a simple matter to work out the equations for them.

$$X_n = X_{n-1} - Y_{n-1}r_\theta + U_{n-1}r_\phi - V_{n-1}r_\theta r_\phi \quad (67)$$

$$Y_n = Y_{n-1} + X_{n-1}r_\theta + V_{n-1}r_\phi + U_{n-1}r_\theta r_\phi \quad (68)$$

$$Z_n = Z_{n-1} - W_{n-1}r_\phi \quad (69)$$

The variables  $r_\theta$  and  $r_\phi$  are the remaining angles to be rotated through. They are the remainder stored in the  $R$  and  $T$  registers. To get an overall accuracy of  $m$  bits, it is necessary to use the standard CORDIC algorithm to obtain  $m$  bit variables to an accuracy of  $n$  bits and then one iteration as described above. The relationship between  $m$  and  $n$  is

$$n = \left( \frac{m-1}{2} \right) \quad (70)$$

This result is found directly from the second term of the Taylor series (the others do not need to be taken into account). Bit level simulation accuracy results are shown in table 5.3 for both redundant and conventional arithmetic types over 10,000 random vector rotations for each result. Because the remaining angle must be small and the number of bits to which the angles are stored will be the same as the coordinates, the bits that can be anything other than zero will only be in the lower half of the word. It is therefore possible to use a fairly small multiplier (in the region of  $m$  bits by  $\frac{m}{2}$  bits) for the most of the multiplications.

		b				
		18	19	20	21	22
n	9	13	14	14	14	14
	10	13	15	16	16	16
	11	13	14	16	16	17
	12	13	14	15	16	17
	13	13	14	15	16	17

a)

		b				
		18	19	20	21	22
n	9	13	14	15	15	15
	10	13	14	15	16	16
	11	13	14	15	16	17
	12	13	14	15	16	17
	13	13	14	15	16	17

b)

Table 5.3. The accuracy of a 3D Hybrid CORDIC for a) conventional, and b) redundant arithmetic.

#### 5.2.1.8. Higher radix algorithms

The use of higher radix techniques was shown earlier, [Rodrigues81] and this approach can be considered in the three dimensional case. The basis of the extension in two dimensions is that the scaling factor becomes a stable number to  $m$  bits long before the  $m$ th iteration. This also holds in the three dimensional case and is shown in table 5.4. The underlined figures indicate the limit of accuracy at a particular stage.

Iterations	Scale Factor (Conventional)	Scale Factor (Redundant)
1	10.000000000000000000	1.100100000000000000
2	10.100000000000000000	1.110000111001000000
3	10.101010000000000000	1.11010001110010001011
4	10.101100101010000000	1.11010101011011100001
5	10.1011010101010010101	1.11010110010110001111
6	10.101101011111111111...	1.11010110100100111011
7	10.1011011000101011010...	1.11010110101000100111
8	10.1011011000111000111...	1.11010110101001100001
9	10.1011011000111001100...	1.11010110101001110000
$\infty$	10.1011011000111001110...	1.11010110101001110101...

Table 5.4. The scaling factor accuracy for 3D CORDIC.

For a given desired final accuracy, it is possible to determine how many radix 2 iterations are required before the scale factor becomes, in effect a constant. After this iteration, it is possible to use a radix 4 system to halve the number of remaining iterations. Because the radix 4 system used by Rodrigues et al. uses a redundant form, it fits very well with the redundant CORDIC form, although it is possible to use the radix 4 system in combination with a conventional CORDIC system (this was the technique used by Rodrigues). If the number of radix 2 iterations needed is  $j$  (which is shown for various configurations in table 5.5, then the radix 4 iterations (for the three dimensional non redundant case) are of the form given in equations 71 to 78.

Number of bits	2D		3D	
	Conventional	Redundant	Conventional	Redundant
8	5	6	7	5
16	9	8	10	9
32	17	16	19	17

Table 5.5. The number of radix 4 iterations for the scale factor to become (apparently) constant.

$$U_{i+1} = U_i - X_i b_i \delta_i - V_i a_i \delta_i + Y_i a_i b_i \delta_i^2 \quad (71)$$

$$V_{i+1} = V_i - Y_i b_i \delta_i + U_i a_i \delta_i - X_i a_i b_i \delta_i^2 \quad (72)$$

$$W_{i+1} = W_i + Z_i b_i \delta_i \quad (73)$$



$$X_{i+1} = X_i + U_i b_i \delta_i - Y_i a_i \delta_i - V_i a_i b_i \delta_i^2 \quad (74)$$

$$Y_{i+1} = Y_i + V_i b_i \delta_i + X_i a_i \delta_i + U_i a_i b_i \delta_i^2 \quad (75)$$

$$Z_{i+1} = Z_i - W_i b_i \delta_i \quad (76)$$

Where  $\delta_i = 2^{-i-1}$  and :

$$\delta_{i+1} = 4 \times \delta_i \quad (77)$$

The selection variable  $a_i \in \{-2, -1, 0, 1, 2\}$  is now given by:

$$a_i = \begin{cases} -2 & \text{if } E_3\left(\frac{R_i}{\delta_i}\right) \geq \frac{13}{8} \\ -1 & \text{if } \frac{5}{8} \leq E_3\left(\frac{R_i}{\delta_i}\right) < \frac{13}{8} \\ 0 & \text{if } -\frac{5}{8} < E_3\left(\frac{R_i}{\delta_i}\right) < \frac{5}{8} \\ 1 & \text{if } -\frac{13}{8} < E_3\left(\frac{R_i}{\delta_i}\right) \leq -\frac{5}{8} \\ 2 & \text{if } E_3\left(\frac{R_i}{\delta_i}\right) \leq -\frac{13}{8} \end{cases} \quad (78)$$

$E_j(x)$  is  $x$  to a precision of  $j$  fractional bits. This does not require a full evaluation of  $\frac{S_i}{\delta_i}$  since it can be shown that:

$$-\frac{21}{8} < E_3\left(\frac{S_i}{\delta_i}\right) < \frac{21}{8} \quad (79)$$

The selection process for  $b_i$  relative to  $T_i$  is identical to the selection process of  $a_i$  relative to  $S_i$ .

The above set of equations gives an algorithm with an accuracy of  $n$  fractional bits in less than  $n$  iterations and it would be possible to create a scaleless 2D vector rotator in this manner. The accuracy of this algorithm is shown in table 5.4 for an accuracy of 16 fractional bits. Hence, from table 5.5 there are 10 radix 2 iterations

followed by a number of radix 4 iterations. The results shown in table 5.6 are taken from a bit level simulation of the algorithm.

<i>Iterations</i>			<i>Internal accuracy (fractional bits)</i>				
<i>Radix 2</i>	<i>Radix 4</i>	<i>Total</i>	<i>20</i>	<i>21</i>	<i>22</i>	<i>23</i>	<i>24</i>
10	1	11	9	9	9	9	9
10	2	12	11	11	11	11	11
10	3	13	13	13	13	13	13
10	4	14	14	15	15	15	15
10	5	15	15	16	16	16	16
10	6	16	15	16	16	16	17

Table 5.6. The output accuracy of the conventional arithmetic 3D algorithm with radix 4 iterations.

These techniques can be applied to the equations given by Takagi [Takagi91] for the double rotation method and can then be generalised to three dimensions. The equations for  $U_{i+1}$ ,  $V_{i+1}$ ,  $X_{i+1}$  and  $Y_{i+1}$  are given by equations 41, 42, 44 and 45.  $Z_{i+1}$ ,  $W_{i+1}$  and  $S_{1 \rightarrow 4}$  are given by equations 74 to 81.

$$W_{i+1} = W_i(1 - b_i d_i \delta_i^2) + Z_i((b_i + d_i)\delta_i) \quad (74)$$

$$Z_{i+1} = Z_i(1 - b_i d_i \delta_i^2) - W_i((b_i + d_i)\delta_i) \quad (75)$$

$$S_1 = 1 - (a_i c_i + b_i d_i)\delta_i^2 + a_i c_i b_i d_i \delta_i^4 \quad (76)$$

$$S_2 = (b_i + d_i)\delta_i - a_i c_i (b_i + d_i)\delta_i^3 \quad (77)$$

$$S_3 = (a_i + c_i)\delta_i - b_i d_i (a_i + c_i)\delta_i^3 \quad (78)$$

$$S_4 = (a_i + c_i)(b_i + d_i)\delta_i^2 \quad (79)$$

where:

$$\delta_i = \frac{\delta_{i-1}}{4} \quad (80)$$

and at the first radix 4 iteration, after  $m$  radix 2 iterations:

$$\delta_m = 2^{-m} \quad (81)$$

## 5.2: CORDIC in more than two dimensions

Terms containing  $\delta$  to the power of three or more can in fact be ignored since these will be beyond the precision of the arithmetic being used. The iterations are very similar for radix 2 and radix 4 when using redundant numbers and so very little extra hardware would be needed to allow the radix 4 iterations.

Results of a bit level simulation are shown in table 5.7 for 9 and 10 radix 2 iterations. This gives us an algorithm that can supply vector rotations faster than 1 bit per cycle, with each cycle taking time  $O(1)$ . These can be three dimensional incorporating a scale factor, or two dimensional with no scale factor.

Iterations (n)			Internal Accuracy (b)				
radix 2	radix 4	total	19	20	21	22	23
9	4	13	13	14	14	14	13
10	4	14	14	14	15	15	15
9	5	14	15	15	15	15	15
10	5	15	14	15	16	16	17
9	6	15	n/a	15	15	15	15
10	6	16	n/a	n/a	16	17	17
9	7	16	n/a	n/a	n/a	15	15
10	7	17	n/a	n/a	n/a	n/a	17

Table 5.7. Accuracy results for a 3D redundant CORDIC with radix 4 iterations.

### 5.2.2. Evaluation of 3 Dimensional CORDIC algorithms

A number of variations have now been shown on the original three dimensional CORDIC algorithm given at the beginning of the chapter. No indication has been given on which algorithm is 'best' for a given set of requirements. There are two major areas to consider, the comparison of two and three dimensional algorithms and the comparison of the various three dimensional algorithms. In the following assessment of the different algorithms, only the datapath is considered. Since the datapath is likely to be much larger than the control section of the circuit, the omission of the control section from the analysis is unlikely to cause any major errors in the results. The time taken by the control unit will also be constant for most situations.

### 5.2.2.1. Two versus Three dimensional CORDIC

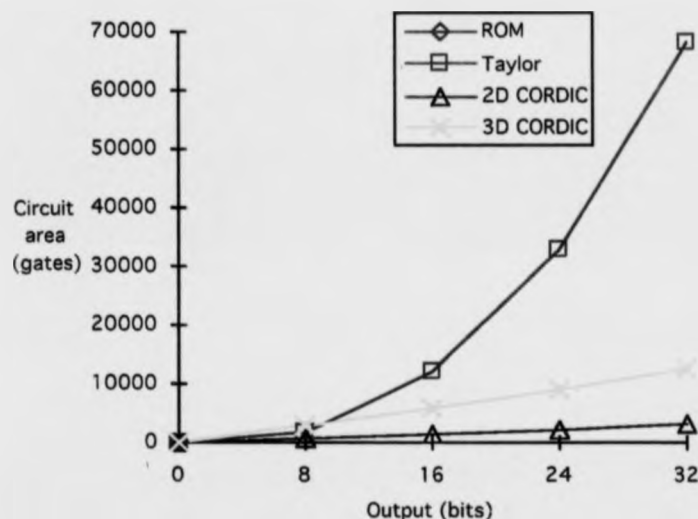
A two dimensional CORDIC engine can be made to perform the same function as a three dimensional engine. For a general rotation of a 3D vector, the 2D CORDIC must rotate the projection of the vector onto the X or Y axis, rotate in the X-Z, or Y-Z plane and then rotate the vector in the X-Y plane again to get it to its final position. In this way, the rotation is restricted to changing only two of the three coordinates at any one time. Hence 3, 2D CORDIC operations are needed. If  $U$ ,  $V$  and  $W$  variables are not known, it will take 2, 2D CORDIC operations to find them before the 3D CORDIC could be used. Hence for a single rotation, there is no benefit to be gained from using a 3D CORDIC system. However, once the variables have been calculated, they can be stored and used again when the vector is next rotated. Unfortunately, if two vectors are summed, then  $U$ ,  $V$  and  $W$  must be re-evaluated. When using the 3D CORDIC to produce a scaleless 2D CORDIC, only one 3D CORDIC operation is required as opposed to two for the 2D CORDIC. The number of CORDIC engines required and the number of sequentially performed operations are given in table 5.8 for a number of different possible uses. Values are given for time efficient and area efficient solutions. In the table  $A_{2D}$  and  $A_{3D}$  are the areas taken up by a 2D and 3D CORDIC unit respectively and  $T_{2D}$  and  $T_{3D}$  are the times taken by a 2D and 3D CORDIC operation respectively.

	Area Efficient				Time Efficient			
	2D		3D		2D		3D	
	Time	Area	Time	Area	Time	Area	Time	Area
2D Scale-less	$2T_{2D}$	$A_{2D}$	$T_{3D}$	$A_{3D}$	$2T_{2D}$	$A_{2D}$	$T_{3D}$	$A_{3D}$
3D without $U$ , $V$ and $W$	$3T_{2D}$	$A_{2D}$	$2T_{2D} + T_{3D}$	$A_{3D}$	$2T_{2D}$	$3A_{2D}$	$2T_{2D} + T_{3D}$	$A_{3D}$
3D with $U$ , $V$ and $W$	$3T_{2D}$	$A_{2D}$	$T_{3D}$	$A_{3D}$	$2T_{2D}$	$3A_{2D}$	$T_{3D}$	$A_{3D}$

Table 5.8. The 2D verses the 3D CORDIC algorithm for various operations.

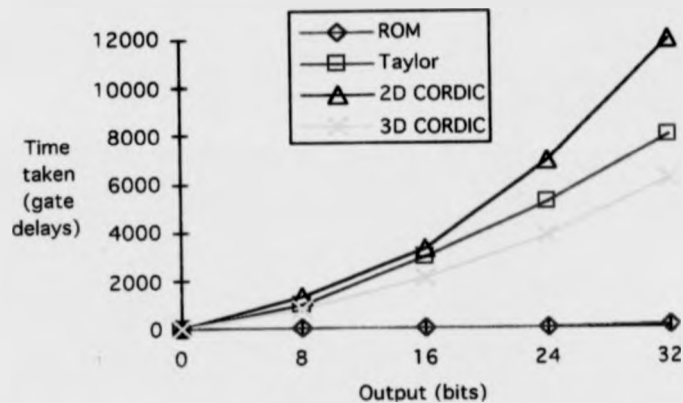
## 5.2: CORDIC in more than two dimensions

Relative values for  $A_{2D}$ ,  $A_{1D}$ ,  $T_{2D}$  and  $T_{3D}$  need to be found. If a 16 fractional bit result is required, then both two and three dimensional algorithms require 19 iterations. However, the three dimensional algorithm requires 23 bits of internal fractional accuracy, 3 overflow bits and a sign bit (total: 27 bits), whereas the two dimensional case requires 21 bits of internal fractional accuracy 2 overflow bits and a sign bit (total: 24 bits). Similar reasoning can be used for other required accuracies. Hence it is possible to find the hardware required for a single unit in terms of basic gates over a range of accuracies. Graph 5.1 shows the variation of gate counts for two and three dimensional CORDIC systems. Graph 5.2 gives corresponding overall execution time. ROM and Taylor series solutions are also shown assuming that the application is a scaleless two dimensional vector rotation. The ROM area estimates are omitted from graph 5.1 because they are so large. With the scale shown, the plotted point for the ROM for a 32 bit accuracy would be of the order of *ten thousand million light years* above the top of the graph! Even at an accuracy of 4 bits the ROM solution would not be on the graph. This is a reflection of the exponential increase in size of ROM style solutions.



Graph 5.1. Gate counts for vector rotations with various required accuracies.

The 2D CORDIC gate counts and execution times given are for a 2D conventional CORDIC with a multiplication using the CORDIC processor. The time taken for an operation could be improved by about 20% by using scale factor compensation. This compares with 35% to 50% improvement by using a 3D CORDIC and a 10% to 35% improvement by using a Taylor's series implementation in the range shown.

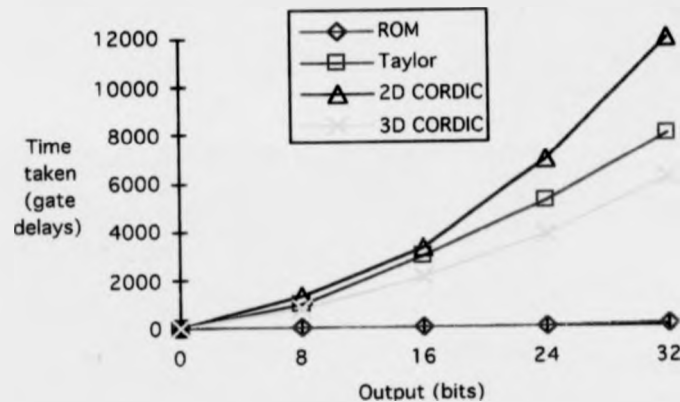


Graph 5.2. Execution times for vector rotations with various required accuracies.

#### 5.2.2.2. Three Dimensional Algorithms

A number of different three dimensional CORDIC algorithms have been discussed. In order to get some kind of understanding for the relative merits of each of these algorithms, area and time estimates will be shown. Time for execution can be divided into two parts: cycle time and number of iterations. Either time parameter, or the product of the two can be important and so all are given. Serial execution and pipelined cases are considered. Figure 5.8 is the legend for graphs 5.3 to 5.9. Graphs 5.3 to 5.6 relate to the different algorithms running in a serial manner. The multiplications used by the hybrid algorithms are implemented using extra iterations in the CORDIC unit.

The 2D CORDIC gate counts and execution times given are for a 2D conventional CORDIC with a multiplication using the CORDIC processor. The time taken for an operation could be improved by about 20% by using scale factor compensation. This compares with 35% to 50% improvement by using a 3D CORDIC and a 10% to 35% improvement by using a Taylor's series implementation in the range shown.



Graph 5.2. Execution times for vector rotations with various required accuracies.

#### 5.2.2.2. Three Dimensional Algorithms

A number of different three dimensional CORDIC algorithms have been discussed. In order to get some kind of understanding for the relative merits of each of these algorithms, area and time estimates will be shown. Time for execution can be divided into two parts: cycle time and number of iterations. Either time parameter, or the product of the two can be important and so all are given. Serial execution and pipelined cases are considered. Figure 5.8 is the legend for graphs 5.3 to 5.9. Graphs 5.3 to 5.6 relate to the different algorithms running in a serial manner. The multiplications used by the hybrid algorithms are implemented using extra iterations in the CORDIC unit.

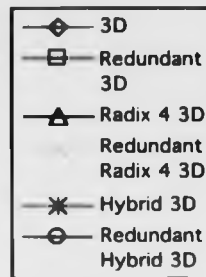
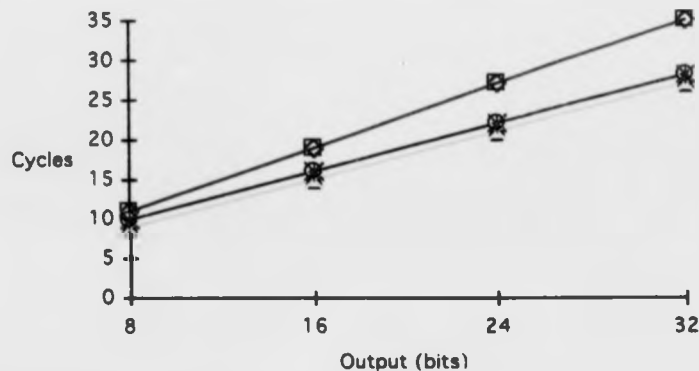


Figure 5.8. Graph Legend.

Graph 5.3 shows the number of clock cycles (or algorithm iterations) required for a given output accuracy. Whether the arithmetic used is conventional or redundant, there is a saving of about 20% gained from the use of either a radix 4 or hybrid algorithm. The hybrid algorithms are shown in the graph as taking slightly more cycles than the radix 4 approach. A careful examination of the accuracy required in the multiplication could possibly remove this difference. The radix 4 algorithms provide a convergence of better than 1 bit per cycle for more than 8 bit output accuracies, but it should be noted that the unmodified algorithms do not quite produce 1 bit per cycle.



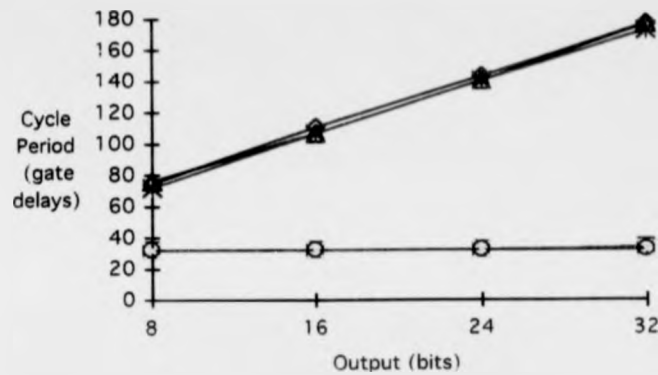
Graph 5.3. Number of cycles taken in various 3D CORDIC algorithms (serial execution).

Graph 5.4 shows the cycle period of the algorithms. As would be expected, the algorithms using conventional arithmetic show a linear dependence on word length, whereas the redundant arithmetic algorithms are largely unaffected by the word length.



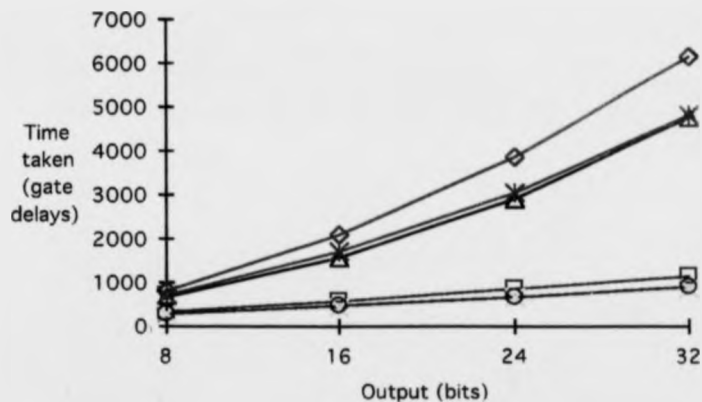
## 5.2: CORDIC in more than two dimensions

Some parts of the redundant algorithms such as the shifters are affected by the word length, but their delay is small in comparison to the other components that make up the majority of the delay.



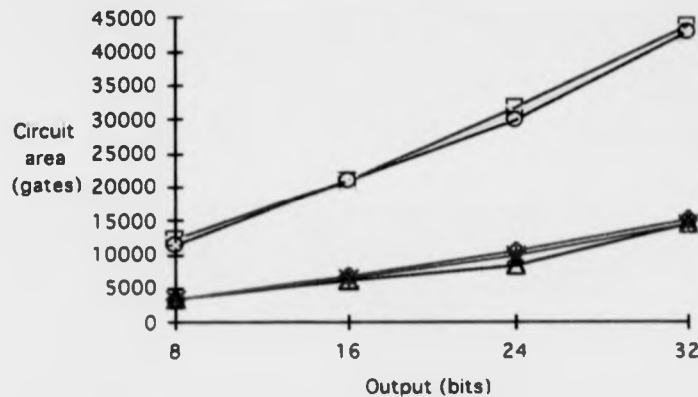
Graph 5.4. Cycle period in gate delays for various 3D CORDIC algorithms (serial execution).

Graph 5.5 shows the overall time taken (the number of cycles multiplied by the cycle period) to execute the algorithm. This is the important value when throughput is the most important design factor. At 32 bits output accuracy, the redundant algorithm is 5 times faster than the conventional 3D algorithm and the radix 4 version is 7 times faster.



Graph 5.5. Overall time taken for various 3D CORDIC algorithms (serial execution).

Graph 5.6 shows the area required to construct the circuits. The area required for a redundant arithmetic circuit is about 3 times that of a conventional arithmetic circuit. The area requirement of both types of algorithms grows slightly more than linearly with required accuracy due to the area required by the shifters. However, the redundant arithmetic algorithms are less affected by the shifter area.



Graph 5.6. Area required for various 3D CORDIC algorithms (serial execution).

In figure 5.9 the area used by the two unmodified algorithms is split up so as to show the proportions of the area used for different parts of the circuit. Redundant arithmetic CORDIC circuits are less heavily affected by the size of the shifter circuits because they use fewer in proportion to the rest of the circuit. This is due to the fact that each shifter output can be used to produce more than one adder input. This reduces the number of shifters required. The 'Other' section refers to multiplexers, conditional inverters and other similar parts of the circuit. These are much more predominant in the redundant arithmetic circuit because adder word inputs may be zero as well as positive or negative versions of the register values.

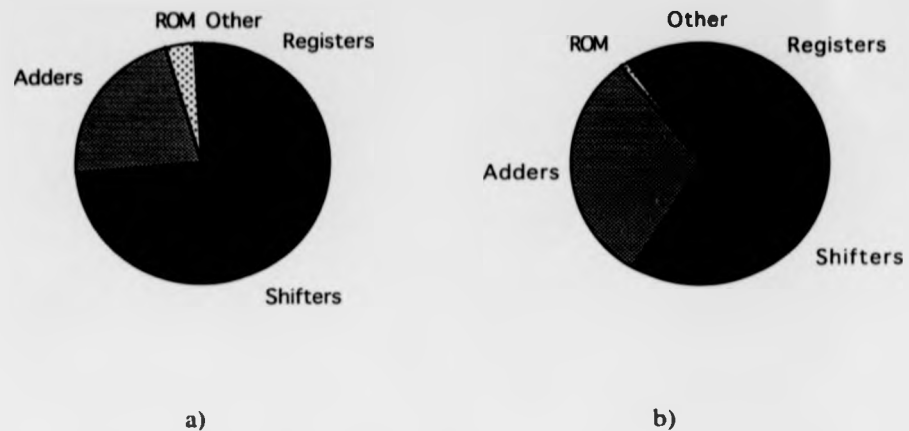
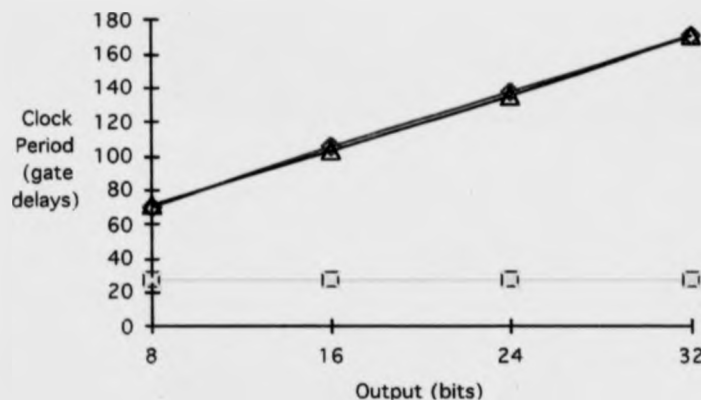


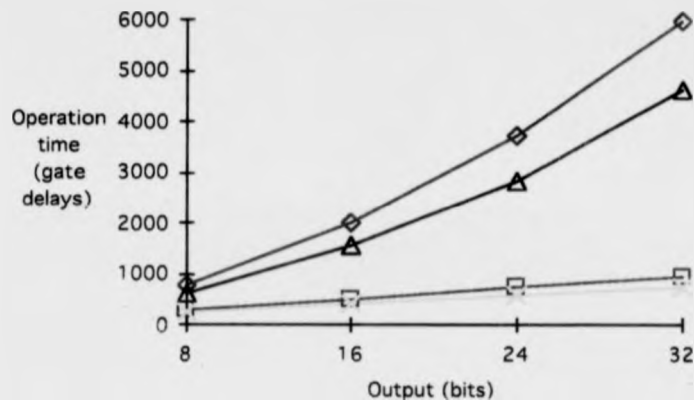
Figure 5.9. The fractional area used by the different parts of a) conventional and b) redundant arithmetic CORDIC processors.

The three dimensional CORDIC algorithms can also be implemented as pipelined structures. In this case, the most significant part of the serial circuits; the shifters, can be hard wired. It has been shown that the hybrid algorithms are almost identical to the radix 4 algorithms in terms of area and time, so only the radix 4 algorithms are plotted. Graph 5.7 shows the clock period of the circuits. As shifters are no longer present, the period of the redundant arithmetic circuits is totally independent of word length.



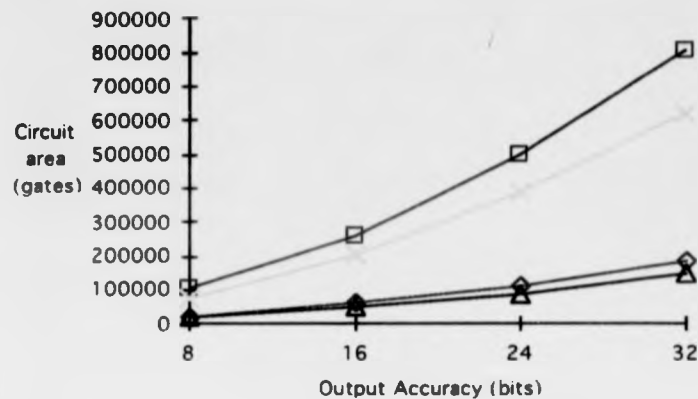
Graph 5.7. Cycle period in gate delays for various pipelined CORDIC algorithms.

Graph 5.8 shows the overall time taken for an operation. It can clearly be seen that the redundant arithmetic circuits offer a much faster operation than the conventional arithmetic circuits. It should also be noted, that at any one time, a number of operations are in progress so that the operation time is an indication of the latency of the circuit, whilst the reciprocal of the period indicates the throughput of the circuit.



Graph 5.8. Overall time in gate delays for various pipelined CORDIC algorithms.

Graph 5.9. shows the area requirements of the different pipelined versions of the algorithms. The redundant algorithms take in the region of 4 times as much area as the conventional arithmetic, whereas it can be seen that they are more than 6 times faster. It should be remembered, however that if a binary result is required, then the redundant numbers will need conversion. The time required would be similar to the difference between the unmodified and the radix 4 or hybrid algorithms. However, the extra area required would be very small due to the fact that most of the variables used in the algorithm would not be needed. The redundant algorithms would still be about 6 times faster.



Graph 5.9. Area required in gates for various pipelined CORDIC algorithms.

### 5.3. 3D CORDIC in VLSI

The area estimates given in the previous section do not allow for two major factors in VLSI design. The first factor is wiring which can take up very considerable areas, and is not necessarily related to the number of gates. The second factor is wasted area. Very irregularly shaped circuits do not fit neatly next to other circuits incorporated on the same device, or need space left around them if they are the only circuit on the device. However, for a set of similar circuits, such as pipelined 3D CORDIC circuits, the layout, and hence the wiring overheads and wasted space will be roughly the same for the different circuits relative to their own gate counts. For example, it can be seen in graph 5.9, that the gate count for a 32 bit pipelined radix four 3D CORDIC using redundant arithmetic is about twice that of a 24 bit pipelined radix two 3D CORDIC using conventional arithmetic. Wasted space and wiring will also take up area in roughly the same proportions. To assess the amount of wasted space, and wiring area in the 3D CORDIC circuits, a 32 bit pipelined radix 4 3D CORDIC using redundant arithmetic has been floorplanned, and partially laid out. The fabrication process assumed was a double metal CMOS  $1.5\mu\text{m}$  process, and the circuit was laid out using the design rules for an Orbit process of this type. Transistor sizing for high speed operation was not considered since this would not affect the circuit area

greatly. The work involved in fully laying out and testing a design such as this is huge, and as only a small amount of time was available, only a portion of the circuit was laid out. In addition, no attempt was made to optimise pipeline sections to take account of parts of the standard section not used at that point in the pipeline. This section was however, significant enough to allow a confident assessment of the area of the whole circuit.

The floorplan for the circuit is shown in figure 5.10. The circuit has been split into three major sections: the angle register pipeline, the WZ pipeline, and the UVXY pipeline. Redundant number to binary converters are placed at the end of the circuit to convert X, Y, and Z back into binary. Each pipeline has fifteen stages, ten radix two, and five radix four, and the length of each pipeline is the same. This causes the circuit to be rectangular which reduces the amount of wasted area at the edge of the circuit. A similar floorplan would be applicable to all 3D CORDIC pipelined circuits.

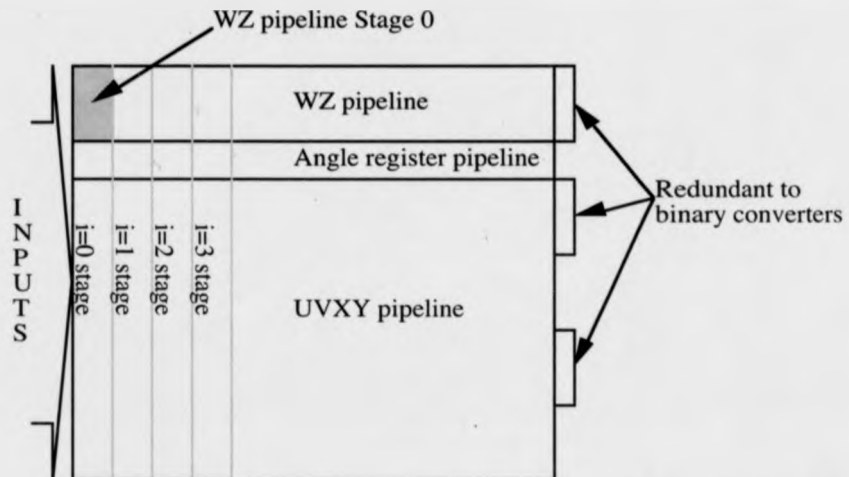


Figure 5.10. The Floorplan for a pipelined 3D CORDIC circuit (Redundant radix 4).

Both the WZ pipeline, and the angle register pipeline have been fully laid out. Twenty four bit pipelines were needed to provide a 16 bit result, as can be seen by referring to table 5.7, Twenty one fractional bits are needed for accuracy, and three non fractional bits are needed to prevent overflow due to scaling. In the following sections, the basic building blocks for the circuit are described, followed by a description of the two

pipelines laid out. Finally, the area of the overall circuit is estimated, and compared with the estimates given earlier. The wiring area, and wasted area is also assessed. The circuit areas are given in units of  $\lambda^2$ .  $\lambda$  is a layout parameter based on the process dimensions. In this case,  $\lambda$  is  $0.8\mu\text{m}$ , and hence  $\lambda^2$  is  $0.64\mu\text{m}^2$ .

### 5.3.1. Basic building blocks

The pipelined 3D CORDIC algorithm consists of a set of stages, each of which is a sum of shifted register outputs. The two major components of the circuit are therefore registers and adders. Both must cope with redundant arithmetic. The shifting of variables is constant for a particular stage in the pipeline, and so it is simply hardwired. Some additional circuitry was also required to conditionally invert the digit, and other similar operations. These additional circuits are not shown here. The redundant number representation chosen was two bits, one indicating a positive value, and the other indicating a negative value. The representation is given in table 5.9.

$A^+$	$A^-$	Value
0	0	0
0	1	1
1	0	-1
1	1	X

Table 5.9. The Redundant number representation.

To reduce wiring space, it was decided that the adder circuits should be designed using only the first metal layer, to allow the hardwired shifting operation to be overlaid on top of the adders. The adder block is similar to that used by McQuillan et. al [McQuillan91], and is shown in block form in figure 5.11. Because the  $t$  and  $t'$  variables change adder stages as they move up the adder, no carry propagation is more than 2 digits long.

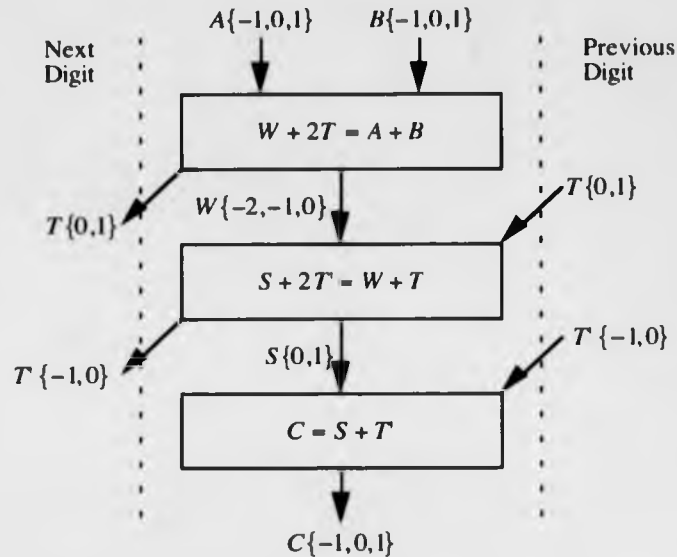


Figure 5.11. Block diagram of a single digit redundant full adder.

There must be an additional stage at the most significant end of the adder to prevent unnecessary overflow. This circuitry is not shown here, but was included in the design of the adders. The single metal layout of a redundant full adder is shown in figure 5.12. The circuit has one slight modification to a conventional adder. The adders cells placed in a column produce two adders with bits of equivalent significance next to each other. This is used in the WZ pipeline as will be shown later. The area of the adder is about  $50,000\lambda^2$ , which is much higher than the estimate used in an earlier section. The increase in area is due to three major reasons. First, the circuit is produced in single metal to allow over-routing, and is being compared to circuits laid out in two metal layers. Secondly, due to time constraints, really compact design was not achieved. Thirdly, a number of signals had to be routed through the circuit, adding to the wiring area.



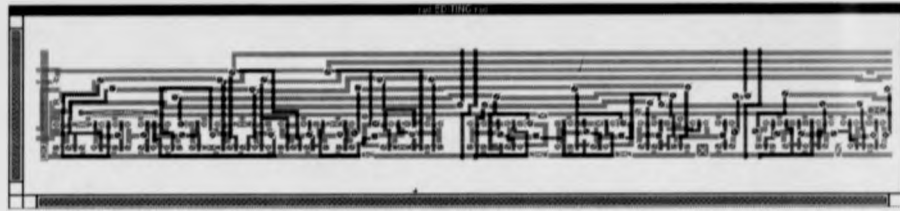


Figure 5.12. The redundant full adder single digit section.

The register circuit used was a simple one. The circuit schematic for a single register bit is shown in figure 5.13. For each digit, two of these were required, one for the positive bit, and one for the negative bit. In the schematic,  $\phi_1$  and  $\phi_2$  are the two phases of a non-overlapping clock. The data is output to the rest of the circuit on the rising edge of  $\phi_2$  and computation must be complete by the falling edge of  $\phi_1$ . This gives almost the whole cycle for the computation.

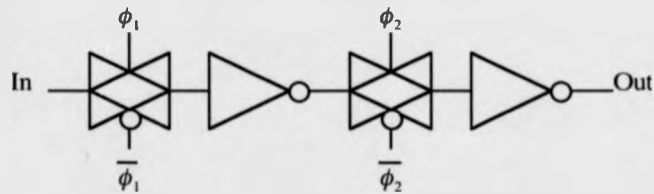


Figure 5.13. Bit register schematic.

Two single bit registers are shown in figure 5.14. They form a single digit register that can be replicated vertically to form a word register. The area of the register was about  $3000\lambda^2$ .

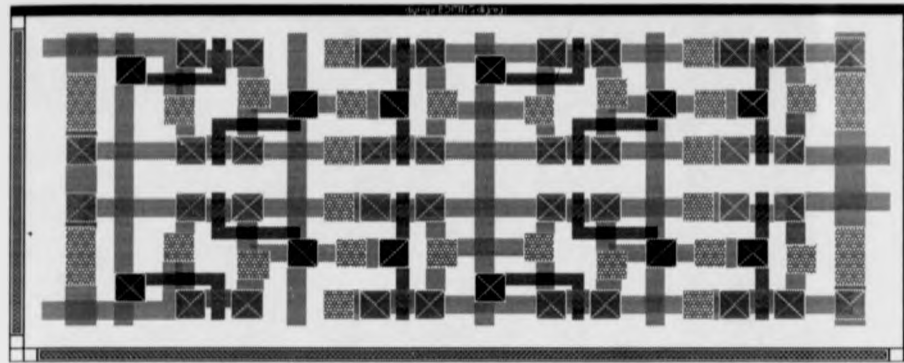


Figure 5.14. Single digit register layout

For comparison purposes, a two input NAND gate was laid out and is shown in figure 5.15. The circuit uses both metal layers and has an area of approximately  $1000\lambda^2$ . This is the standard gate referred to in the previous area estimates.

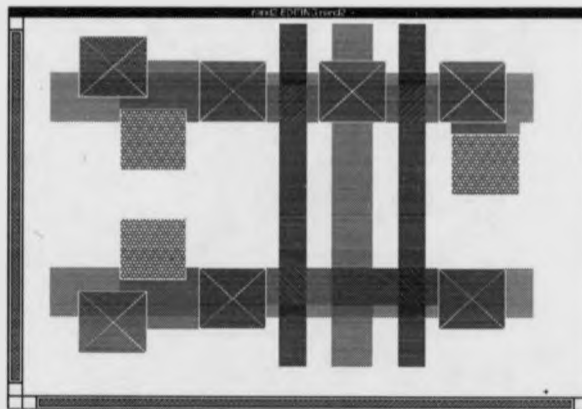


Figure 5.15. A double metal NAND gate layout

### 5.3.2. The angle register pipeline

The angle register pipeline takes two rotation angles as inputs and outputs control signals to the WZ, and UVXY pipelines on the directions and magnitudes of each rotation stage in the pipeline. At each stage in the pipeline, there are two operations performed on each of the angle variables. The first operation is the evaluation of the most significant few bits, and the second is the addition or subtraction of a constant according to the result of the evaluation. The evaluation stage

also provides the control signals for the other pipelines. The radix two evaluation circuitry is shown in figure 5.16. The radix four circuit was a little more complex. The area of this circuit is comparable with the area of a single digit redundant adder circuit, but there are only two of these circuits per stage as opposed to forty eight adder cells in the angle pipeline alone. Consequently, the evaluation circuit area is not significant in the overall circuit area.

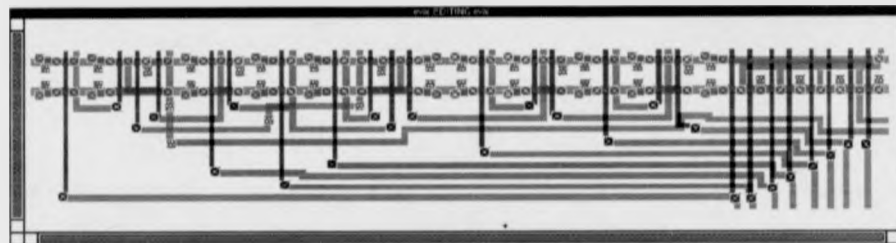


Figure 5.16. Radix two angle evaluation circuit.

At first sight, it would appear that the most sensible way to set out the angle register pipeline would be to have two identical pipelines, one for each angle variable, placed next to each other. However, if this is done, large amounts of padding would be required to lengthen the angle register pipeline to fit with the other pipelines. By combining the two pipelines as shown in figure 5.17, the angle register pipeline can be made to match the WZ pipeline. The two angle pipeline stages are placed next to each other, and each stage has lines to route through the data for the other stage.

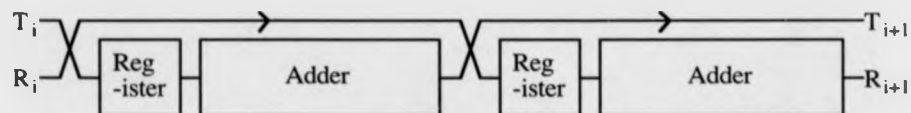


Figure 5.17. The angle register pipeline organisation (bit slice).

At each stage in the pipeline, the angle value gets smaller. This would mean a different wiring to the evaluation circuitry at each stage in the pipeline. To remove this need the angle value is simply shifted up one place during each of the radix two stages, and two places at each of the radix four stages. Digits shifted into the bottom of the word are always zero, and digits shifted out of the top of the bit are tested for overflow (which

would only occur in the case of error in the circuit design). The constant that is added to or subtracted from the angle variables is different in the different stages of the pipeline, but is the same for the two angle variables (although its sign may be different). A simple one word ROM circuit is placed in between the two angle register circuits in each stage of the pipeline. The output from this ROM is then used by both of the angle register circuits. The layout of a radix two angle register single pipeline stage is shown in figure 5.18. Only the wiring layers are shown for clarity. The area of the radix two circuit is about  $4.1M\lambda^2$ . The area of the radix four circuit is about 10% more.

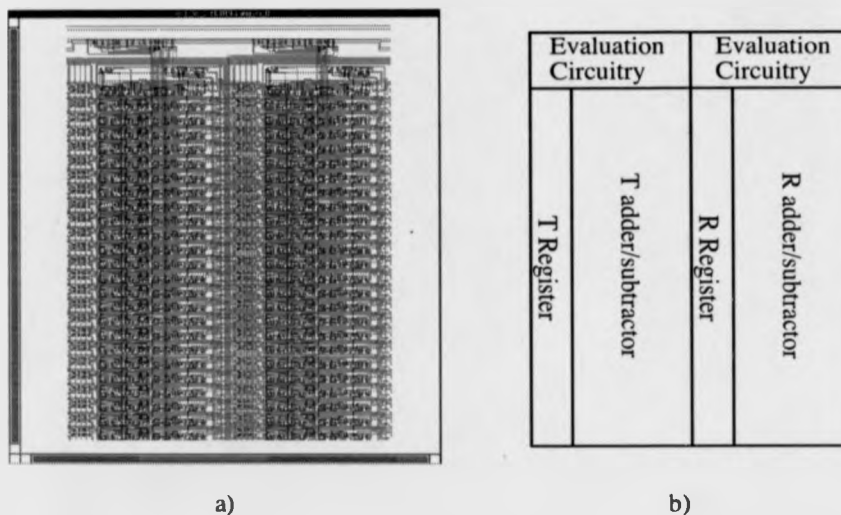


Figure 5.18. a) A radix two angle register single pipeline stage. b) Circuit block diagram.

### 5.3.3. The WZ pipeline

The WZ pipeline performs a fairly simple iteration. Both variables are summed with a shifted version of themselves and each other. The shifted versions may be zeroed, or negated. The data is stored in a register, and added together in two adders placed horizontally next to each other. The variables are shifted using metal layer 2 over the adders. If the adders for the two variables were placed vertically above each other, the shifting wires would be very difficult to lay out efficiently. However by interleaving the adders as shown in figure 5.19, the wiring is made much easier. The

reason for this is that with interleaved registers, at any particular stage, the wiring of one shifted digit is almost identical to all the others, and there is no significant need to have one signal cross over another (which would require the use of metal layer 1 as well).

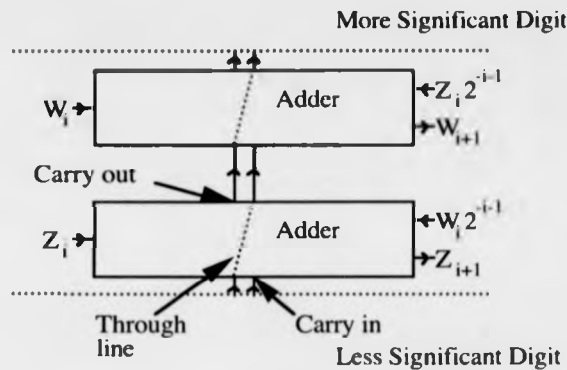


Figure 5.19. The layout of the WZ pipeline adders

The WZ pipeline has been laid out and tested, and a single radix two pipeline stage is shown in figure 5.20. The area of a radix two stage is about  $7.4M\lambda^2$ . The gate count for the pipeline as a whole is higher than estimated by 50%. This seems like a large error, but is mainly due to the larger than estimated number of transistors in the redundant adder circuit. A more thorough design of this circuit would reduce this factor very considerably.

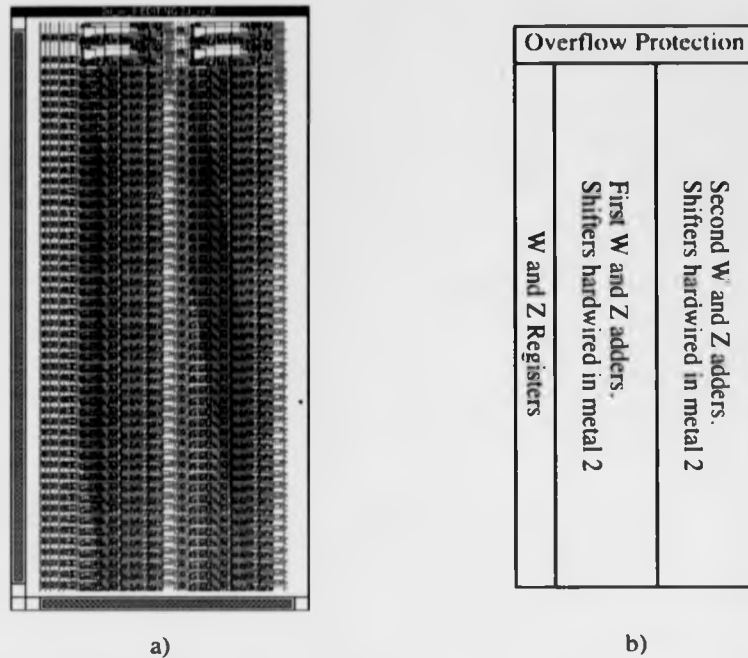


Figure 5.20. a) A radix two stage of the WZ pipeline. b) Circuit block diagram.

#### 5.3.4. The complete pipeline

The complete pipeline has not been laid out. However, a significant proportion (20% in terms of gates) has, and is shown in figure 5.21. This section of the design is shorter than the UVXY pipeline, and would be stretched so that the pipeline stages of the WZ and angle register pipeline were the same length as the UVXY pipeline stages. This introduces wasted area into the design, but the regularity of the layout would be much better, and the extra length of the UVXY stages would only be apparent in the first few stages as will be explained later.



Figure 5.21. The WZ, and angle register pipelines.

The UVXY pipeline would be very similar to the WZ pipeline, but eight interleaved adders would be needed. This is because up to seven adders would be needed per stage, and these would be put in two rows for each register and there are four registers. Any more rows would be pointless as there would not be enough room in the width of the adders for all the metal 2 shift lines that are needed. In all but the first few stages, the terms that are scaled by  $2^{-4i-4}$ , and  $2^{-3i-3}$  can be neglected since the number of digits shifted is greater than the number of digits in the register. Hence the result of the shift must be zero. In subsequent stages, the adders used for these terms can be omitted leading to a shorter pipeline stage. The area of the circuit including wiring and wasted space is estimated to be about 2.5 times its basic area given by the gate count of circuit multiplied by the area of a basic gate. For a circuit of this kind, a factor of 2.5 is very reasonable. In the  $1.5\mu\text{m}$  process that it was designed in, the circuit would measure approximately 20mm, by 30mm. This is very large, but scaled to a more modern process with a  $0.5\mu\text{m}$  line width, this becomes 7mm by 10mm, and this would be a suitable size for fabrication on a single Integrated Circuit (IC). The removal of adders that sum terms scaled by  $2^{-2i-2}$  once they are obviously zero (after the eleventh stage in this design) and the possibility of merging the WZ and angle register pipelines could reduce this area still further. These area estimates are based upon the redundant adder designed, and are heavily dependant upon it. Any decrease in the area of this one cell would result in significant overall area savings provided the cell was kept long enough to allow over-routing of the shifted variables.

#### 5.4. Summary

This chapter has described and examined an extension to the CORDIC algorithm [Volder59] which allows us to remove the need for scaling. The extension also has applications in the conversion of spherical polar coordinates into Cartesian coordinates. The removal of the need for scaling significantly improves the CORDIC algorithm and increases its potential throughput. This algorithm is entirely compatible with Walther's unified algorithm [Walther71] and the various redundant arithmetic schemes [Takagi91, Duprat91] for the CORDIC algorithm.

When used as a 3D rotator it has been shown that the throughput per unit area can be significantly increased by the use of higher radices. This is true even when the circuit is implemented in binary and with binary inputs and outputs. In short, higher radices can out perform binary even in a heavily binary biased environment.

The area and floorplanning of these circuits has been examined, and it has been shown that circuits of reasonable accuracy and speed are possible. The layout of these circuits is a large task, but the important basic cells have been identified, which allows effort to be put into parts of the circuit that give maximum area reduction benefits.



# 6

## Conclusions and Future Work

In this thesis, the design of circuits that operate in a non-binary manner has been examined. These MVL circuits are either constructed using a direct approach using multiple levels on a single interconnection, or using an indirect approach, having a single multiple-value signal communicated by a number of binary interconnections. The new concept of non-linear current encoding has been explained, and a methodology for the design of circuits using this concept has been described. Novel extensions have also been shown to the CORDIC vector rotation algorithm. This chapter concludes the thesis by drawing together the major points made in the thesis. Following this, suggestions are made for possible areas for further research.

### 6.1. Conclusions

The conclusions drawn from this thesis can be split into three broad categories: direct MVL implementation, indirect MVL implementation, and extensions to the CORDIC algorithm. In this section conclusions are drawn from the work described in earlier chapters.

#### 6.1.1. Direct MVL implementation

The direct implementation of MVL in CMOS has been examined in detail. The trends in CMOS production technology indicate that MVL circuits will become more

difficult as the process sizes shrink. The channel modulation effect, and variations in transistor threshold voltages make the smaller processes less predictable for non-binary circuits. For simulation purposes, MVL circuits need to be considered as non-linear analogue circuits to get reasonable results.

It is possible to add extra processing steps in order to produce devices that give good MVL performance. For example the low threshold variation depletion mode transistor used by Kawahito et. al. [Kawahito87]. However, this will inevitably affect the yield of the production process as extra processing steps are involved. In addition, the use of a non-standard process precludes a circuits use as a part of any integrated circuit not using that process. Even new binary processes can take many years to become well used. For example, the BiCMOS process has not supplanted CMOS yet [Alvarez91]. Bearing in mind, that binary logic is almost universal in its use, it would be foolish to attempt to supplant it completely. MVL circuits must therefore compete with binary circuits using in a standard binary VLSI process to have a reasonable chance of being used. This has been done for example in the case of the ROM on the INTEL 8087 [Stark81], but as has already been pointed out, this will become less likely in the CMOS process at least.

The use of non-linear encoding of current mode signals has been shown to allow the construction of efficient functions that don't contain an obvious input sum. Other functions are produced by summing non-linear functions of the input values, and then applying another non-linear function to the sum, to form the result. The non-linear encoding technique which was first described by the author of this thesis in [Summerfield92] has been extended, and a new methodology for the design of these circuits has been described. The methodology gives an indication at an early stage of the area and resolution required to implement a given function. It has been shown that a modulo 4 sum is not very well suited to design in current mode MVL as a single stage function, because of the modulus part of the function. the maximum function has been shown to be well suited to this kind of implementation, especially when a highly non-linear 'grounded output' encoding is used. The design of libraries of current

mode MVL functions has been examined, and guidelines explained. The guidelines take into account the specific advantages and disadvantages of current mode MVL.

The idea that bandwidth can be increased by the use of voltage mode MVL input and output on VLSI devices has been shown to be false. If the comparison is made for differential signalling, MVL can show improvements [Baltus90]. However, power consumption of MVL input and output drivers has potential for being much lower than binary I/O when the same power supply voltage is used. In addition, the actual increase in power from increasing the maximum voltage supply in an voltage mode MVL circuit in order to increase the radix is far less than would be expected.

Bipolar technology offers better characteristics for MVL than binary, but its use is declining. This means that MVL bipolar circuits will become appropriate for fewer and fewer designs. BiCMOS, however is a technology that is becoming more widely accepted, and so BiCMOS designs could have potential in the future. The process offers high density CMOS transistors, and low parameter tolerance bipolar transistors. The bipolar transistors could offer MVL designers with an opportunity to produce circuits more easily in a standard binary logic process that is likely to be useful for a considerable period of time.

In the future, optical logic circuits may become a major logic technology. Current mode techniques for logic implementation would transfer well to this technology. Current mode techniques have been developed in this thesis for the general design of logic circuits. BiCMOS circuits could also make use of these designs. There is the potential to implement algorithms such as the 3D CORDIC algorithm presented in this thesis using circuits based upon the novel design methodology described.

### 6.1.2. Indirect MVL implementation

The indirect implementation of MVL algorithms and techniques in binary has been shown to have advantages over binary algorithms. Indeed some of the best binary algorithms are really MVL algorithms. For example, carry save addition

[Wallace64] is really a redundant radix two addition with digits  $\in \{0,1,2,3\}$ , as the carry and sum output for each digit can be viewed as a single four valued signal. In this thesis, the use of redundant radix two arithmetic, and higher radix arithmetic has been shown to give algorithms that have better performance than traditional binary algorithms. The specific case of the three dimensional CORDIC algorithm has been used in this thesis. The implementation of this algorithm is by no means a trivial task, but indirect implementation of a Multiple-Valued version of the algorithm has been shown to have advantages over a conventional arithmetic version.

### 6.1.3. Extensions to the CORDIC algorithm

The CORDIC algorithm [Volder59] has been examined in detail, and significant extensions made. The three dimensional radix four CORDIC algorithm provides the ability to calculate vector rotations at a rate exceeding 1 bit per iteration, without the excessive hardware requirements of a high speed Taylor series evaluation. Pipelined versions using redundant arithmetic would give very high throughputs with a reasonable latency. It is also interesting to note that the redundant number implementations of all of the CORDIC algorithms have stable outputs MSB first. This would make them ideal for connection to many other redundant number circuits [McQuillan91]. In this case, care should be taken to ensure that a sufficient latency is allowed to ensure the stability of the digits. The three dimensional algorithm is useful in the conversion of spherical polar coordinates into Cartesian coordinates, and general three dimensional vector rotations. However the general three dimensional vector rotations can only be performed on a vector that starts on one axis, or a vector for which the extra components U, V and W are known.

## 6.2. Suggestions for future work

The use of MVL has, so far, been split into two distinct sections: direct implementations of MVL circuits, and indirect implementations of MVL algorithms. However, the combination of the MVL circuits and MVL algorithms is clearly an area

for further research. This section examines possible further work in MVL circuits, algorithms, and combinations of the two.

### 6.2.1. MVL circuits

The direct implementation of MVL in a standard CMOS process is becoming less practical as the process advances. However, the use of a small number of bipolar transistors could help to alleviate the problems with MVL design. For this reason, BiCMOS technology offers the potential for easy implementation of MVL circuits, although scaling issues must still be addressed. Another technology that should be investigated in detail is optical technology. The ability to sum light sources is identical to Kirchhoff's current law, making optical technology ideal for extending the concepts of current mode logic. As optical technology is quite different to the current design technologies, it may be that designers will accept more readily the concept of MVL design as they shift to using this new technology.

### 6.2.2. MVL algorithms

The implementation of MVL in an indirect form has been shown to have very definite benefits over conventional binary algorithms. Further work in this area could focus on Walther's extensions [Walther64], and extending them to the three dimensional case. It should be possible to produce a scaleless two dimensional hyperbolic rotation by performing a three dimensional hyperbolic rotation in the same way as the scaleless two dimensional circular rotation was performed. It may also be possible to combine rotation types to, for example, get  $\cosh A \times \sin B$ . Higher numbers of dimensions should also be possible, although the number of variables used will roughly double for each extra dimension. This is because each extra dimension adds an extra cosine or sine function to each variable. The rotated vector will expand out to twice the number of terms half relating to the sine, and half to the cosine of the angle of the vector in the extra dimension. This increase is large, but as many of the extra terms will be shifted by large amounts, they will quickly become

## 6.2: Suggestions for future work

negligible. Only the first few iterations would be much more complicated than with fewer dimensions.

### 6.2.3. The combination of MVL circuits and algorithms

It is clear that circuits constructed using MVL building blocks will have MVL algorithms. Circuits of this kind have been reported for example Kawahito et al produced a radix four multiplier based on current mode MVL adders [Kawahito87]. Redundant arithmetic (even when the radix is two) uses non-binary signals. All redundant arithmetic circuits have potential for implementation in MVL. A clear area for further research would be an investigation of the effect of using MVL circuits to implement the CORDIC algorithms presented earlier.

# Bibliography

- [Ahmed82]: "Signal Processing Algorithms and Architectures", H.M. Ahmed, PhD thesis, Stanford University, June 1982.
- [Alvarez91]: "BiCMOS-has the promise been fulfilled?", A.R. Alvarez, IEEE International Electron Devices Meeting 1991 Technical Digest, 1991, pp. 355-358.
- [An92]: "Integrated optical nand gate", X. An, K.M. Geib, M.J. Hafich, L.M. Woods, S.A. Feld, F.R. Beyette, Jun., G.Y. Robinson and C.W. Wilemsen, Electronics Letters, Vol. 28, No. 16, July 1992, pp. 1545-1546.
- [Aragaki92]: "A Multiple-Valued Content-Addressable Memory Using Logic-Value Conversion and Threshold Functions", S. Aragaki, T. Hanyu and T. Higuchi, Proceedings IEEE 23rd International Symposium on Multiple Valued Logic, May 1993, pp. 170-175.
- [Aytac87]: "Ternary logic based on a novel MOS building block circuit", H.M. Aytac, International Journal of Electronics, Vol. 63, No. 2, 1987, pp. 241-251.
- [Babbage1837]: "On the mathematical powers of the calculating engine" C. Babbage, Unpublished manuscript, Dec. 1837, Reprinted in: The Origins of Digital Computers Selected Papers, Springer-Verlag, 1982, pp. 19-54.
- [Baltus90]: "An Efficient Multi-Level Multi-Wire Differential Interface", P. Baltus, P. van der Meulen and R. Morley, Proceedings IEEE 20th International Symposium on Multiple Valued Logic, May 1990, pp. 181-188.
- [Barr88]: "The incremental-cost approach for synthesis of CCD 4-valued unary functions", M.H. Abd-El-Barr, T.D. Hoang and Z.G. Vranesic, Proceedings IEEE 18th International Symposium on Multiple Valued Logic, May 1988, pp. 82-89.
- [Barr89]: "Programmable realisation of multi-valued multi-threshold functions using CCDs", M.H. Abd-El-Barr, T.D. Hoang and Z.G. Vranesic, Proceedings IEEE 19th International Symposium on Multiple Valued Logic, May 1989, pp. 42-53.
- [Barr90]: "On the synthesis of MVMT functions for PLA implementation Using CCDs", M.H. Abd-El-Barr and H. Choy, Proceedings IEEE 20th International Symposium on Multiple Valued Logic, May 1990, pp. 316-323.
- [Bastiaansen91]: "A 10-b 40-MHz 0.8 $\mu$ m CMOS Current-Output D/A Converter", C.A.A. Bastiaansen, D.W.J. Groeneveld, H.J. Schouwenaars and H.A.H. Termee, IEEE Journal of Solid State Circuits, Vol. 26, No. 7, July 1991, 917-921.

- [Bhattacharya90]: "Binary to quaternary encoding on clocked CMOS circuits using weak buffer", D. Bhattacharya, Proceedings IEEE 20th International Symposium on Multiple Valued Logic, May 1990, pp. 174-180.
- [Boyle70]: "Charge Coupled Semiconductor Devices", W.S. Boyle and G.E. Smith, The Bell System Technical Journal, April 1970, pp. 587-593.
- [Butler88]: "Multiple-Valued CCD Circuits", J.T. Butler and H.G. Kerkhoff, IEEE Computer, April 1988, pp. 58-68.
- [Cavallaro87]: "CORDIC Arithmetic for an SVD Processor", J.R. Cavallaro and F.T. Luk, Proceedings IEEE 8th Symposium on Computer Arithmetic, 1987, pp. 113-120.
- [Chew87]: "On the design of CMOS ternary logic circuits using T-gates", B.P. Chew and H.T. Mouftah, International Journal of Electronics, Vol. 63, No. 2, 1987, pp. 229-239.
- [Cho88]: "A CMOS ternary ROM chip", Y.H. Cho and H.T. Mouftah, Proceedings IEEE 18th International Symposium on Multiple Valued Logic, May 1988, pp. 358-363.
- [Chown90a]: "VLSI Design of a Pipelined CORDIC Processor", P. Chown, D.W. Walton and G.R.Nudd, Research Report 164, Department of Computer Science, University of Warwick, October 1990.
- [Chown90b]: "Notes on the Design of a Barrel Shifter for the Warwick Pipelined CORDIC processor", P. Chown, D.W. Walton and G.R.Nudd, Research Report 161, Department of Computer Science, University of Warwick, August 1990.
- [Clarke92a]: "A Redundant Arithmetic CORDIC System With A Unit Scale Factor", C.T. Clarke, 3rd IMA Conference on Mathematics in Signal Processing, December 1992. To be published in proceedings.
- [Clarke92b]: "A Redundant Arithmetic CORDIC System With A Unit Scale Factor", C.T. Clarke, Research Report RR234, University of Warwick Dept of Computer Science, December 1992.
- [Clarke94]: "Current Mode Techniques for Multiple Valued Arithmetic and Logic", C.T. Clarke, G.R.Nudd, and S. Summerfield, Submitted for IEEE International Symposium on Circuits and Systems 94, 1994.
- [Current87]: "A CMOS multiple valued logic test chip", K.W. Current, F. Edwards and D. Freitas, Proceedings IEEE 17th International Symposium on Multiple Valued Logic, May 1987, pp. 16-19.
- [Current88]: "A proposed DCT/IDCT chip design using quaternary logic", K.W. Current, Proceedings IEEE 18th International Symposium on Multiple Valued Logic, May 1988, pp. 40-44.



- [Current89]: "A CMOS quaternary latch", K.W. Current, Proceedings IEEE 19th International Symposium on Multiple Valued Logic, May 1989, pp. 54-57.
- [Current 91]: "A Bi-directional Current-Mode CMOS Multiple Valued Logic Memory Circuit", K.W. Current and M.E. Hurlston, Proceedings IEEE 21st International Symposium on Multiple Valued Logic, May 1991, pp. 196-202.
- [Daggett59]: "Decimal-Binary Conversions in CORDIC", D. H. Daggett, IRE Transactions on Electronic Computers, Vol. EC-8, Issue 3, 1959, pp. 335-339.
- [Delosme89]: "CORDIC algorithms: Theory and Extensions" J.M. Delosme, SPIE Vol. 1152 Advanced Algorithms and Architectures for Signal Processing IV, 1989, pp. 131-145.
- [Delosme90]: "CORDIC algorithms in four dimensions", J.M. Delosme and S.F. Hsiao, SPIE Vol. 1348 Advanced Signal-Processing Algorithms, Architectures and Implementations, 1990, pp. 349-360.
- [Despain74]: "Fourier Transform Computers Using CORDIC Iterations", A.M. Despain, IEEE Transactions on Computers, Vol. C-23, No. 10, October 1974, pp. 993-1001.
- [Dewilde92]: "Standard Algebraic Problems And Bitwise Parallelism", P. Dewilde and G. Hekstra, Proceedings IEEE International Symposium on Circuits and Systems, May 1992, pp. 1634-1635.
- [Dimauro93]: "A New Technique for Fast Number Comparison in the Residue Number System", G. Dimauro, S. Impedvo and G. Pirlo, IEEE Transactions on Computers, Vol. 42, No. 5, May 1993, pp. 608-612.
- [Dixon90]: "An Array Processor Implementation of the CORDIC Algorithm", G. Dixon, Proceedings of IEE Colloquium on VLSI Signal Processing Architectures", May 1990, pp. 5/1-8.
- [Dube90]: "Alleviating memory bottlenecks using multi-level memory", D. Dube and A.V. Mayrhauser, Proceedings IEEE 20th International Symposium on Multiple Valued Logic, May 1990, pp. 102-109.
- [Duprat91]: "Fast VLSI Implementation of CORDIC using Redundancy", J. Duprat and J-M. Muller, Algorithms and Parallel VLSI Architectures Volume B: Proceedings, 1991, Elsevier Science Publishers, pp. 155-164.
- [Edirisooriya92]: "Aliasing in Multiple-Valued Test Data Compaction", G. Edirisooriya and J.P. Robinson, Proceedings IEEE 22nd International Symposium on Multiple Valued Logic, May 1992, pp. 43-50.
- [Epstien93]: "Multiple-Valued Logic Design: An Introduction", G. Epstein, IOP Publishing Ltd, Bristol, 1993.

- [Ercegovac73]: "Radix-16 Evaluation of Certain Elementary Functions", M.D. Ercegovac, IEEE Transactions on Computers, Vol. C-22, 1973, pp. 561-566.
- [Ercegovac90]: "Redundant and On-Line CORDIC: Application to Matrix Triangularization and SVD", M.D. Ercegovac and T. Lang, IEEE Transactions on Computers, Vol. 39, No. 6, June 1990, pp. 725-740.
- [Etiemble80]: "On the realisation of multiple-valued flip-flops", D. Etiemble and M. Israel, Proceedings IEEE 10th International Symposium on Multiple Valued Logic, 1980, pp. 16-23.
- [Etiemble90]: "4-valued BiCMOS Circuits for the Transmission System of a Massively Parallel Architecture", D. Etiemble, C. Chanussot and V. Neri, Proceedings IEEE 20th International Symposium on Multiple Valued Logic, May 1990, pp. 348-353.
- [Etiemble92]: "On the Performance of Multivalued Integrated Circuits: Past, Present and Future", D. Etiemble, Proceedings IEEE 22nd International Symposium on Multiple Valued Logic, May 1992, pp. 156-164.
- [Farmwald81]: "High Bandwidth Evaluation of Elementary Functions", P.M. Farmwald, Proceedings IEEE 5th Symposium on Computer Arithmetic, 1981, pp. 139-142.
- [Freitas83]: "CMOS current comparator circuit", D.A. Freitas and K.W. Current, Electronics Letters, Vol. 19, No. 17, August 1983, pp. 695-697.
- [Fulcher89]: "CORDIC Survey", J. Fulcher, Preprint No. 89/1, Dept of Computing Science, University of Wollongong, N.S.W., Australia.
- [Ghest71]: "Multiplying made easy for digital assemblies", C. Ghest, Electronics, Vol. 44, November 1971, pp. 56-61.
- [Ginderdeuren85]: "CORDIC Based HIFI Digital FM Demodulator Algorithm for Compact VLSI Implementation", J. Van Ginderdeuren, L. Van Paepegem, J. Lecocq, R. Govaerts, F. Catthoor, P. Vandebroek, S. Slock, T.A.C.M. Claasen and H. De Man, Electronics Letters, Vol. 21, No. 25/26, December 1985, pp. 1227-1229.
- [Gise86]: "Modern semiconductor fabrication technology", P. Gise and R. Blanchard, Prentice-Hall, 1986.
- [Hanyu87]: "Quaternary gate array for pattern matching and its application to knowledge information processing system", T. Hanyu, M. Kameyama and T. Higuchi, Proceedings IEEE 17th International Symposium on Multiple Valued Logic, May 1987, pp. 181-187.
- [Hanyu88]: "Design of a highly parallel AI processor using new multiple-valued MOS devices", T. Hanyu, and T. Higuchi, Proceedings IEEE 18th International Symposium on Multiple Valued Logic, May 1988, pp. 300-306.

- [Hanyu90]: "Design of a high-density multiple-valued content-addressable memory based on floating-gate MOS devices", T. Hanyu, and T. Higuchi, Proceedings IEEE 20th International Symposium on Multiple Valued Logic, May 1990, pp. 18-23.
- [Harding91]: "A Comparison of Redundant CORDIC Rotation Engines", J.A. Harding, T. Lang and J.A. Lee, IEEE Conference on Computer Design, VLSI in Computers & Processors, Cambridge, MA, October 1991, pp. 556-559.
- [Haviland80]: "A CORDIC Arithmetic Processor Chip", G.L. Haviland and A.A. Tuszynski, IEEE Transactions on Computers, Vol. c-29, No. 2, February 1980, pp. 68-79.
- [Haznedar91]: "Digital Microelectronics", H. Haznedar, The Benjamin/Cummings Publishing Company, 1991.
- [Hekstra93]: "Floating Point CORDIC", G.J. Hekstra, and E.F.A. Deprettere, IEEE 11th conference on computer arithmetic, 1993, pp. 130-137.
- [Ho89]: "Switched capacitor circuits in the implementation of multiple-valued logic", H.L. Ho and K.C. Smith, Proceedings IEEE 19th International Symposium on Multiple Valued Logic, May 1989, pp. 202-209.
- [Horiguchi88]: "An Experimental Large-Capacity Semiconductor File Memory Using 16-Levels/Cell Storage", M. Horiguchi, M. Aoki, Y. Nakagome, S. Ikenaga and K. Shimohigashi, IEEE Journal of Solid State Circuits, Vol. 23, No. 1, February 1988, pp. 27-33.
- [Horowitz80]: "The Art of Electronics", P. Horowitz and W. Hill, Cambridge University Press, 1980.
- [Hsiao91]: "The CORDIC Householder Algorithm", S.F. Hsiao and J.M. Delsome, Proceedings IEEE 10th Symposium on Computer Arithmetic, 1991, pp. 256-263.
- [Hu86]: "Ternary Scan Design for VLSI Testability", M. Hu and K.C. Smith, IEEE Transactions on Computers, Vol. C-35, No. 2, February 1986, pp. 167-170.
- [Hu90]: "A Novel Implementation of a Chirp Z-Transform Using a CORDIC Processor", Y.H. Hu and S. Naganathan, IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. 38, No. 2, February 1990, pp. 352-354.
- [Hu91]: "Expanding the Range of Convergence of the CORDIC Algorithm", X. Hu, R.G. Harber and S.C. Bass, IEEE Transactions on Computers, Vol. 40, No. 1, January 1991, pp. 13-21.
- [Hu92]: "The Quantization Effects of the CORDIC Algorithm", Y.H. Hu, IEEE Transactions on Signal Processing, Vol. 40, No. 4, April 1992, pp. 834-844.

- [Huertas81]: "Low-power CMOS implementation of some operators", J.L. Huertas and G. Sanchez-Gomez, Proceedings IEEE 11th International Symposium on Multiple Valued Logic, 1981, pp. 196-199.
- [Hurst80]: "Fibreoptics, a multiple-valued interconnection means?", S.L. Hurst, Proceedings IEEE 10th International Symposium on Multiple Valued Logic, 1980, pp. 115-119.
- [Hurst84]: "Multiple-Valued logic - Its Status and its Future", S.L. Hurst, IEEE Transactions on Computers, December 1984, pp. 1160-1179.
- [Hwang87]: "Evaluating Elementary Functions With Chebyshev Polynomials On Pipeline Nets", K. Hwang, H.C. Wang and Z. Xu, Proceedings IEEE 8th Symposium on Computer Arithmetic, May 1987, pp. 121-128.
- [Ishizuka90]: "On design of multiple-valued static random-access-memory", O. Ishizuka, Z. Tang and H. Matsumoto, Proceedings IEEE 20th International Symposium on Multiple Valued Logic, May 1990, pp. 11-17.
- [Kaliman88]: "Ternary physical protocol for MARILAN a multiple access ring local area network", R.J. Kaliman and C.B. Silio Jr, Proceedings IEEE 18th International Symposium on Multiple Valued Logic, May 1988, pp. 14-20.
- [Kamiura92]: "Easily Testable Multiple-Valued Cellular Arrays", N. Kamiura, Y. Hata, F. Miyawaki and K. Yamoto, Proceedings IEEE 22nd International Symposium on Multiple Valued Logic, May 1992, pp. 36-42.
- [Kamiura93]: "A Repairable and Diagnosable Cellular Array on Multiple-Valued Logic", N. Kamiura, Y. Hata and K. Yamato, Proceedings IEEE 23rd International Symposium on Multiple Valued Logic, May 1993, pp. 92-97.
- [Karasawa93]: "Design and Examination of a Multiple-Valued Flip-Flop Circuit with Stair Shaped I-V Curved Device as a Coupling Element", S. Karasawa and K. Yamanouchi, Proceedings IEEE 23rd International Symposium on Multiple Valued Logic, May 1993, pp. 152-163.
- [Katter90]: "A new CMOS gate - the balanced gate for detecting physical failures", O.E. Katter Jr and H.M. Razavi, Proceedings IEEE 20th International Symposium on Multiple Valued Logic, May 1990, pp. 25-31.
- [Kawahito87]: "A high-speed compact multiplier based on multiple-valued bi-directional current-mode circuits", S. Kawahito, M. Kameyama, T. Higuchi and H. Yamada, Proceedings IEEE 17th International Symposium on Multiple Valued Logic, May 1987, pp. 172-180.

- [Kawahito91]: "Multiple-Valued Current-Mode Arithmetic Circuits Based on Redundant Positive-Digit Number Representations", S. Kawahito, K. Mizuno and T. Nakamura, Proceedings IEEE 21st International Symposium on Multiple Valued Logic, May 1991, pp. 330-339.
- [Kameyama80]: "Design of radix 4 signed-digit arithmetic circuits for digital filtering", M. Kameyama and T. Higuchi, Proceedings IEEE International Symposium on Multiple Valued Logic, June 1980, pp. 272-277.
- [Kameyama81]: "Signed-digit arithmetic circuits based on multiple-valued logic and its applications", M. Kameyama and T. Higuchi, Proceedings IEEE International Symposium on Multiple Valued Logic, June 1981, pp. 41-47.
- [Kameyama88]: "A Multiplier Chip with Multiple-Valued Bidirectional Current-Mode Logic Circuits", M. Kameyama, S. Kawahito and T. Higuchi, Computer, IEEE Computer Society, April 1988, pp. 43-56.
- [Lee91]: "SVD by Constant-Factor-Redundant-CORDIC", J.A. Lee and T. Lang, Proceedings IEEE 10th Symposium on Computer Arithmetic, June 1991, pp. 264-271.
- [Lee92]: "Dynamic Current-Mode Multi-Valued MOS Memory With Error Correction", E.K.F. Lee and P.G. Gulak, Proceedings IEEE 22nd International Symposium on Multiple Valued Logic, May 1992, pp. 208-215.
- [Lin90]: "On-Line CORDIC Algorithms", H.X. Lin and H.J. Sips, IEEE Transactions on Computers, Vol. 39, No. 8, August 1990, pp. 1038-1052.
- [MacSorley61]: "High-Speed Arithmetic in Binary Computers", O.L. MacSorley, Proceedings of the IRE, Vol. 49, 1961, pp. 67-91.
- [Mangin86]: "Characteristics of Prototype CMOS Quaternary Logic Encoder-Decoder Circuits", J.L. Mangin and K.W. Current, IEEE transactions on computers, Vol. C-35, No. 2, February 1986, pp. 157-161.
- [Manzoul87]: "Binary addition via MVL-CCD carry-look-ahead circuit", M.A. Manzoul and M. Ashraf, Proceedings IEEE 17th International Symposium on Multiple Valued Logic, May 1987, pp. 210-214.
- [Manzoul88]: "Quaternary logic for carry-look-ahead binary addition", M.A. Manzoul and A. Bommireddy, Proceedings IEEE 18th International Symposium on Multiple Valued Logic, May 1988, pp. 294-299.
- [McCluskey80]: "Logic design of MOS ternary logic", E.J. McCluskey, Proceedings IEEE International Symposium on Multiple Valued Logic, 1980, pp. 1-5.

- [McCluskey82]: "A discussion of multiple-valued logic circuits", E.J. McCluskey, Proceedings IEEE International Symposium on Multiple Valued Logic, 1982, pp. 200-205.
- [McQuillan91]: "A VLSI Architecture for Multiplication, Division and Square Root", S.E. McQuillan and J.V. McCanny, Proceedings ICASSP91, May 1991.
- [Millman87]: "Microelectronics", J. Millman and A. Grabel, McGraw Hill, Singapore, 1987.
- [Muzio86]: "Multiple Valued Switching Theory", J.C. Muzio and T.C. Wesselkamper, Adam Hilger Ltd, Bristol, 1986.
- [Nairn88]: "High-resolution, current-mode A/D converters using active current mirrors", D.G. Nairn and C.A.T. Salama, Electronics Letters, Vol. 24, No. 21, October 1988, pp. 1331-1332.
- [Nakamura91]: "A 10-b 70-MS/s CMOS D/A Converter", Y. Nakamura, T. Miki, A. Maeda, H. Kondoh and N. Yazawa, IEEE Journal of Solid State Circuits, Vol. 26, No. 4, April 1991, pp. 637-642.
- [Nash82]: "Combinatorial Digital Logic Using Charge-Coupled Devices", J.G. Nash, IEEE Journal of Solid State Circuits, Vol. SC-17, No. 5, October 1982, pp. 957-963.
- [Ohhashi85]: "High-Speed Computation of Unary Functions", M. Ohhashi and R.F. Schnieder, Proceedings of the IEEE Symposium on Computer Arithmetic, 1985, pp. 82-85.
- [Osawa92]: "Two-dimensional optical and/or logic operations using an optoelectronic integrated functional device (OFD) array", Y. Osawa, K. Yamaguchi, H. Kondo and S. Satoh, Electronics Letters, Vol. 28, No. 22, October 1992, pp. 2084-2085.
- [Prieto88]: "The design of decoders for q-valued logic circuits", A. Prieto, P. Martin-Smith, F. Pelayo and A. Lloris, Proceedings IEEE 18th International Symposium on Multiple Valued Logic, May 1988, pp. 32-39.
- [Rich86]: "A Survey of Multivalued Memories", D.A. Rich, IEEE Transactions on Computers, Vol. C-35, No. 2, February 1986, pp. 99-106.
- [Rodrigues81]: "Hardware Evaluation of Mathematical Functions", M.R.D. Rodrigues, J.H.P. Zurawski and J.B. Gosling, IEE Proceedings Part E, Vol. 128, No. 4, July 1981, pp. 155-164.
- [Rozon88]: "Pseudo-Random Testing of CMOS Ternary Logic Circuits", C. Rozon and H.T. Mouftah, Proceedings IEEE 18th International Symposium on Multiple Valued Logic, May 1988, pp. 316-320.
- [Rozon90]: "Realisation of a Three-Valued Logic Built-in Testing Structure", C.N. Rozon and H.T. Mouftah, IEEE Journal of Solid-State Circuits, Vol. 25, No. 3, June 1990, pp. 814-820.

- [Rozon91]: "Testability Analysis of CMOS Ternary Circuits", C. Rozon and H.T. Mouftah, Proceedings IEEE 21st International Symposium on Multiple Valued Logic, May 1991, pp. 158-165.
- [Sansen88]: "A CMOS Temperature-Compensated Current Reference", W.M. Sansen, F.O. Eynde and M. Steyaert, IEEE Journal of solid-state circuits, Vol. 23, No. 3, June 1988, pp. 821-824.
- [Schultz89]: "A CMOS Binary Adder Using a Quaternary Ganged-Logic Internal Node", K.J. Schultz and K.C. Smith, Proceedings IEEE 19th International Symposium on Multiple Valued Logic, May 1989, pp. 356-359.
- [Seymour88]: "Electronic Devices and Components", J. Seymour, Longman Scientific and Technical, 1988.
- [Shanbhag90]: "Quaternary Logic Circuits in 2 $\mu$ m CMOS Technology", N.R. Shanbhag, D. Nagchoudhuri, R.E. Siferd and G.S. Visweswaran, IEEE Journal of Solid-State Circuits, Vol. 25, No. 3, June 1990, pp. 790-799.
- [Shieh93]: "Series Resonant Tunnelling Diodes as a Two-Dimensional Memory Cell", M.H. Shieh and H.C. Lin, Proceedings IEEE 23rd International Symposium on Multiple Valued Logic, May 1993, pp. 158-163.
- [Shyu84]: "Random Error Effects in Matched MOS Capacitors and Current Sources", J.B. Shyu, G.C. Temes, and F. Krummenacher, IEEE Journal of Solid State Circuits, Vol. SC-19, No. 6, December 1984, pp. 948-955.
- [Silio83]: "Some device count comparisons for reduced control stores using multiple-valued MOS circuits", C.B. Silio and J.H. Pugsley, Proceedings IEEE 13th International Symposium on Multiple Valued Logic, 1983, pp. 249-254.
- [Singh87a]: "Four-valued interface circuits for NMOS VLSI", A.D. Singh, International Journal of Electronics, Vol. 63, No. 2, 1987, pp. 269-279.
- [Singh87b]: "Four valued buses for clocked CMOS VLSI systems", A.D. Singh, Proceedings IEEE 17th International Symposium on Multiple Valued Logic, 1987, pp. 128-133.
- [Specker65]: "A Class of Algorithms for  $\text{Ln } x$ ,  $\text{Exp } x$ ,  $\text{Sin } x$ ,  $\text{Cos } x$ ,  $\text{Tan}^{-1}x$  and  $\text{Cot}^{-1}x$ ", W.H. Specker, IEEE Transactions Electron. Comput. Vol. EC-14, 1965, pp. 85-86.
- [Stark81]: "Two bits per cell ROM", M. Stark, Proceedings of COMPCON, 1981, pp. 209-212.
- [Steer77]: "Digital Hardware for Sine-Cosine Function", D.G. Steer and S.R. Penstone, IEEE Transactions on Computers, Vol. 26, No. 12, December 1977, pp. 1283-1286.

- [Summerfield92] "VLSI Arithmetic With Current Mode Multiple Valued Logic", S. Summerfield, C.T. Clarke and G.R. Nudd, Proceedings IEEE International Symposium on Circuits and Systems 92, May 1992, pp. 3001-3004.
- [Swartzlander75]: "The Sign/Logarithm Number System", E.E. Swartzlander Jr. and A.G. Alexopoulos, IEEE Transactions on Computers, Vol. C-24, 1975, pp. 1238-1242.
- [Takagi91]: "Redundant CORDIC Methods with a Constant Scale Factor for Sine and Cosine Computation", N. Takagi, T. Asada and S. Yajima, IEEE Transactions on Computers, Vol. 40, No. 9, September 1991, pp. 989-995.
- [Texas]: "The 9900 Family Data Book", The Engineering Staff of Texas Instruments Semiconductor Group, Texas Instruments.
- [Timmermann89a]: "Modified CORDIC Algorithm with Reduced Iterations", D. Timmermann, H. Hahn, B. Hostika, Electronics Letters, Vol. 25, No. 15, July 1989, pp. 950-951.
- [Timmermann89b]: "Hough Transform Using CORDIC Method", D. Timmermann, H. Hahn and B.J. Hostika, Electronics Letters, Vol. 25, No. 3, February 1989, pp. 205-206.
- [Timmermann91]: "A Programmable CORDIC Chip for Digital Signal Processing Applications", D. Timmermann, H. Hahn, B.J. Hostika and G. Schmidt, IEEE Journal of Solid State Circuits, Vol. 29, No. 9, September 1991, pp. 1317-1321.
- [Tzou85]: "The Temperature Dependence of Threshold Voltages in Submicrometer CMOS", J.J. Tzou, C.C. Yao, R. Cheung and H. Chan, IEEE Electron Device Letters, Vol. EDL-6, No. 5, May 1985, pp. 250-252.
- [Ueno89]: "A programmable switched-capacitor circuit for multivalued-to-binary and binary-to-multivalued conversions", F. Ueno, T. Inoue, K. Sugitani and S. Araki, Proceedings IEEE 19th International Symposium on Multiple Valued Logic, May 1989, pp. 195-201.
- [Vaudin87]: "3 $\mu$ m VLSI Processing Element Using The CORDIC Algorithm", G.J. Vaudin and G.R. Nudd, Electronics Letters, Vol. 23, No. 21, October 1987, pp. 1164-1166.
- [Viswanathan85]: "Threshold Voltage in Short-Channel MOS Devices", C.R. Viswanathan, B.C. Burkey, G. Lubberts, T.J. Tredwell, IEEE Transactions on Electron Devices, Vol. ED-32, No. 5, May 1985, pp. 932-940.
- [Volder59]: "The CORDIC Trigonometric Computing Technique", J.E. Volder, IRE Trans. Electron. Comput. EC-8, pp. 330-334.



- [Wah93]: "Report on Workshop on High Performance Computing and Communications for Grand Challenge Applications: Computer Vision, Speech and Natural Language Processing, and Artificial Intelligence", B.W. Wah, T.S. Huang, A.K. Joshi, D. Moldovan, J. Aloimonos, R.K. Bajcsy, D. Ballard, D. DeGroot, K. DeJong, C.R. Dyer, S.E. Fahlman, R. Grishman, L. Hirschman, R.E. Korf, S.E. Levinson, D.P. Miranker, N.H. Morgan, S. Nirenburg, T. Poggio, E.M. Riseman, C. Stanfill, S.J. Stolfo, S.L. Tanimoto and C. Weems, IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No.1, February 1993, pp. 138-154.
- [Wallace64]: "A Suggestion for a Fast Multiplier", C.S. Wallace, IEEE Transactions Electron. Comput., Vol. EC-13, 1964, pp. 14-17.
- [Walther71]: "A unified algorithm for elementary functions", J.S. Walther, Spring Joint Computer Conf., 1971, pp. 379-385.
- [Watanabe87]: "New logical-sum and logical-product circuits using CMOS transistors and their applications to four-valued combinational circuits", T. Watanabe, M. Matsumoto and T. Li, International Journal of Electronics, Vol. 63, No. 2, 1987, pp. 215-227.
- [Watanabe88]: "CMOS Four-Valued Logic Circuits Using Charge-Control Technique", T. Watanabe, M. Matsumoto, T. Li and T. Hirayama, Proceedings IEEE 18th International Symposium on Multiple Valued Logic, May 1988, pp. 90-97.
- [Webb91]: "Hybrid higher radix JK flipflop sequencer with ASIC implementation potential", J.A.C. Webb, S.M.N. Forbes, J. Wilson and S.J. Lavery, Electronics Letters, Vol. 27, No. 21, October 1991, pp. 1933-1935.
- [Wei91]: "Multiple Peak Resonant Diode for Multi-Valued Memory", S.J. Wei and H.C. Lin, Proceedings IEEE 21st International Symposium on Multiple Valued Logic, 1991, pp. 190-195.
- [Whitney88]: "Decisive Differences and Partial Differences for Stuck-at Fault Detection in MVL Circuits", M. Whitney and J. Muzio, Proceedings IEEE 18th International Symposium on Multiple Valued Logic, May 1988, pp. 321-328.
- [Wu88]: "Ternary CMOS Sequential Circuits", X. Wu and F. Prosser, Proceedings IEEE 18th International Symposium on Multiple Valued Logic, May 1988, pp. 307-313.
- [Wu89]: "An investigation into quaternary CMOS full-adder based on transmission function theory", X. Wu, X. Chen and F. Prosser, IEEE 19th International Symposium on Multiple Valued Logic, May 1989, pp. 58-62.
- [Wu90]: "CMOS ternary logic circuits", X.W. Wu and F.P. Prosser, IEE Proceedings, Pt. G, Vol. 137, No. 1, February 1990, pp. 21-27.

- [Wu93]: "Novel CMOS Scan Design for VLSI Testability", H. Wu, N. Zhuang and M.A. Perkowski, Proceedings IEEE 23rd International Symposium on Multiple Valued Logic, May 1993, pp. 82-91.
- [Xu88]: "Three-Valued System Diagnosis and Parallel Recovery", J. Xu, T. Chen, J. Xu and S. Huang, Proceedings IEEE 18th International Symposium on Multiple Valued Logic, May 1988, pp. 329-335.
- [Xu89]: "A new current-mode multi-valued storage circuit", X. Xu, S. Li and Z. Cui, Proceedings IEEE 19th International Symposium on Multiple Valued Logic, May 1989, pp. 368-375.
- [Yamakawa86]: "The Current Mode Fuzzy Logic Integrated Circuits Fabricated by the Standard CMOS Process", T. Yamakawa and T. Miki, IEEE Transactions on Computers, Vol. C-35, No. 2, February 1986, pp. 161-167.
- [Yuen88]: "INTEL's Floating-Point Processors", A.K. Yuen, Proceedings of Electro/88, 1988, pp. 48/5/1-6.