WAVE COMPONENT SAMPLING METHOD FOR HIGH PERFORMANCE PIPELINED CIRCUITS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

REFİK SEVER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
ELECTRICAL AND ELECTRONICS ENG. DEPT., METU

SEPTEMBER 2011
Approval of the Thesis:

WAVE COMPONENT SAMPLING METHOD FOR HIGH PERFORMANCE PIPELINED CIRCUITS

Submitted by REFİK SEVER in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical and Electronics Engineering by,

Prof. Dr. Canan ÖZGEN
Dean, Graduate School of Natural And Applied Sciences

Prof. Dr. İsmet ERKMEN
Head of Department, Electrical and Electronics Engineering

Prof. Dr. Murat AŞKAR
Supervisor, Electrical and Electronics Engineering Dept., METU

Examinining Committee Members:

Prof. Dr. Semih BİLGEN
Electrical and Electronics Engineering Dept., METU

Prof. Dr. Murat AŞKAR
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Haluk KÜLAH
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Özgür AKTAŞ
Electrical and Electronics Engineering Dept., BILKENT

Assoc. Prof. Dr. Cüneyt BAZLAMAÇCI
Electrical and Electronics Engineering Dept., METU

Date: 23.09.2011
I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: REFİK SEVER

Signature :
ABSTRACT

WAVE COMPONENT SAMPLING METHOD FOR HIGH PERFORMANCE PIPELINED CIRCUITS

SEVER, Refik
Ph.D., Department of Electrical and Electronics Engineering
Supervisor : Prof. Dr. Murat AŞKAR

September 2011, 126 pages

In all of the previous pipelining methods such as conventional pipelining, wave pipelining, and mesochronous pipelining, a data wave propagating on the combinational circuit is sampled whenever it arrives to a synchronization stage. In this study, a new wave-pipelining methodology named as Wave Component Sampling Method (WCSM), is proposed. In this method, only the component of a wave, whose maximum and minimum delay difference exceeds the tolerable value, is sampled, and the other components continue to propagate on the circuit. Therefore, the total number of registers required for synchronization decreases significantly. For demonstrating the effectiveness of the proposed WCSM, an 8x8 bit carry save adder (CSA) multiplier is implemented using 0.18µm CMOS technology. A generic transmission gate logic block with optimized output delay variation depending on the input pattern is designed and used in all of the sub blocks of the multiplier. Post layout simulation results show that, this multiplier can operate at a speed of 3GHz, using only 70 latches. Comparing with the mesochronous pipelining scheme, the number of the registers is decreased by 41% and the total power of the chip is also decreased by 9.5% without any performance loss. An ultra high speed full pipelined CSA multiplier with an operating frequency of 5GHz is also implemented with WCSM. The number of registers is decreased by 45%, and the power consumption of the circuit is decreased by 18.4% comparing with conventional or mesochronous pipelining methods. WCSM is also applied to different multiplier structures employing booth encoders, Wallace trees, and carry look-ahead adders. Comparing full pipelined 8x8 bit WCSM multiplier with the conventional pipelined multiplier, the number of registers in the implementation of booth encoder, Wallace tree, and carry look-ahead adder is decreased by 30%, 51%, and %62, respectively.
Keywords: Wave-pipelining; high performance multiplier; very high speed integrated circuits; pipeline processing; very large scale integrated circuits.
ÖZ

YÜKSEK PERFORMANSLI BORU HATTI MİMARİLİ DEVRELER İÇİN DALGA ELEMANI ÖRNEKLEME METODU

SEVER, Refik
Doktora, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Murat AŞKAR

Eylül 2011, 126 sayfa

Konvansiyonel boruhattı, dalga boruhattı ya da mesokron boru hattı gibi önceki boruhattı mimarilerinin tamamında, kombinezonal devrede ilerleyen bir veri dalgası, senkronizasyon bölümüne ulaştığı anda örneklenmektedir. Bu çalışmada, Dalga Elemanı Örnekleme Metodu (WCSM) olarak adlandırılan yeni bir dalga boruhattı metodu önerilmektedir. Bu metoddada, yalnızca en az ve en çok gecikme farkı tahammül edilen sınıra ulaşan dalga elemanı örneklenmekte, diğer dalga elemanları devrede ilerlemeye devam etmektedir. Bundan dolayı, senkronizasyon için gereken flip-flop sayısı önemli oranda azalmaktadır. Önerilen metodun etkinliğini göstermek amacıyla, 8x8 bitlik çarpıcı bloğu elde saklama metoduyla ve 0.18µm CMOS teknolojisi kullanılarak gerçeklenmiştir. Genel bir transmisyon kapılı mantık bloğu, çıkışındaki gecikme farkları giriş very diziliminden en az etkilenecek şekilde tasarlanmıştır. Serim sonrası simulasyonlar göstermiştir ki, bu çarpıcı 3GHz çalışma frekansında ve sadece 70 tane kayıt elemani kullanarak çalışabilmektedir. Mesokron boruhattı mimarisine kıyasla, herhangi bir performans kaybı olmadan toplam kayıt elemani sayısı %41 ve toplam güç tüketimi de %9.5 oranında azalmıştır. 5GHz çalışma frekansına sahip çok yüksek hızlı bir çarpıcı bloğu da WCSM metodu kullanılarak tasarlanmıştır. Konvansiyonel boruhattı ya da mesokron boru hattı metodlarına kıyasla, toplam kayıt elemani sayısı %45 ve toplam güç tüketimi de %18.4 oranında azalmıştır. WCSM metodu, booth kodlayıcı, Wallace ağacı ve elde öngörüülü toplayıcı gibi farklı çarpıcı yapılarına da uygulanmıştır. Boruhattı mimarisi her mantıksal işlemin sonunda bir kayıt elemanı olacak şekilde kullanıldığından, WCSM metodu konvansiyonel boruhattı metoduna kıyasla booth kodlayıcıda %30, Wallace ağacında %51 ve elde öngörüülü toplayıcıda %62 oranında kayıt elemani tasarrufu sağlamaktadır.
Anahtar Kelimeler: Dalga boruhattı metodu; yüksek performanslı çarşısı; çok yüksek hızlı entegre devreler; boruhattı işleme; çok büyük ölçekli entegre devreler.
I would like to dedicate this work to my dear wife, Aslı, and my dear children, Ece and Metehan.
ACKNOWLEDGMENTS

I would like to express my deepest gratitude for my supervisor Prof. Dr. Murat AŞKAR for his guidance, advice, supervision, and encouragements throughout the research.

I would also like to thank my supervising committee members Assoc Prof. Dr. Haluk KÜLAH and Assist. Prof. Dr. Ö zgür AKTAŞ for their valuable suggestions, comments and guidance.

I also wish to express my gratitude for committee members Prof. Dr. Semih BİLGEN, and Assoc. Prof. Dr. Cüneyt BAZLAMAÇCI.

I also wish to thank TÜBİTAK-UZAY and Electrical and Electronics Department of Akdeniz University for the facilities and environment provided to me throughout the researchs.

I would like to thank to my family for their great patience, sacrifice, and encouragements throughout my studies.
TABLE OF CONTENTS

ABSTRACT .......................................................................................................................... iv
ÖZ ........................................................................................................................................ v
ACKNOWLEDGMENTS ...................................................................................................... viii
TABLE OF CONTENTS ...................................................................................................... ix
LIST OF ABBREVIATIONS ............................................................................................... xii
LIST OF FIGURES ............................................................................................................ xiii
LIST OF TABLES ................................................................................................................ xiv
CHAPTERS
1 INTRODUCTION .............................................................................................................. 1

2 PIPELINING METHODS ................................................................................................. 8
   2.1 Conventional Pipelining ............................................................................................. 8
   2.2 Wave pipelining ....................................................................................................... 11
       2.2.1 Timing constraints ......................................................................................... 13
       2.2.2 Sources of Delay Differences ....................................................................... 14
       2.2.3 Logic Restructuring ..................................................................................... 14
       2.2.4 Delay Insertion ............................................................................................ 15
       2.2.5 The advantages and disadvantages of wave-pipelining ............................... 16
   2.3 Hybrid Wave Pipelining .......................................................................................... 16
   2.4 Mesochronous Pipelining Scheme ......................................................................... 17

3 MULTIPLIER STRUCTURES ............................................................................................ 18
   3.1 Carry Save Adder (CSA) Multiplier ......................................................................... 20
   3.2 Wallace Trees ......................................................................................................... 22
   3.3 Booth Encoding ...................................................................................................... 25
       3.3.1 Booth 2 Algorithm ..................................................................................... 25
       3.3.2 Booth 3 Algorithm ..................................................................................... 27
       3.3.3 Redundant Booth Algorithm ...................................................................... 28
   3.4 Carry Propagate Addition ..................................................................................... 30

4 WAVE COMPONENT SAMPLING METHOD (WCSM) .................................................. 35
   4.1 Principles of Wave Component Sampling Method ............................................... 36
   4.2 Advantages and Disadvantages of Wave Component Sampling Method ............ 42
APPLICATION OF WCSM TO MULTIPLIER STRUCTURES

5.1 Logic selection.................................................................44
5.2 Delay Balancing................................................................53
5.3 Simultaneous generation of complementary outputs ............55
5.4 Implementation of Multiplier Blocks ..................................58
   5.4.1 Half adder design.........................................................58
   5.4.2 Full adder design..........................................................63
   5.4.3 Partial Product Generation ............................................66
   5.4.4 Sampling of the signals ...............................................70
5.5 Implementation of 8x8 bit CSA multiplier ............................73
   5.5.1 Schematic design ...........................................................73
   5.5.2 Operating Frequency .....................................................75
   5.5.3 Layout Implementation of the multipliers .......................76
   5.5.4 Simulations of multipliers .............................................77
   5.5.5 Performance comparison of the multipliers .....................80
5.6 A 5GHz WCSM-Multiplier ................................................81
5.7 IMPLEMENTATION OF THE OTHER MULTIPLIER STRUCTURES ....85
   5.7.1 Booth encoder design ....................................................85
   5.7.2 Wallace tree design .......................................................88
   5.7.3 Carry Lookahead adder design .....................................91
   5.7.4 CLA Adder Design with 4-input logic gates ....................100
5.8 Delay analysis of logic blocks with 4 inputs ..........................102

6 CONCLUSION ........................................................................107

BIBLIOGRAPHY .......................................................................110

APPENDIX A............................................................................113

HDL CODES OF MULTIPLIER BLOCKS ....................................113
   A.1 Verilog HDL code of top module (booth3_bias_wallace_CLA16) ....113
   A.2 Verilog-HDL code of KplusM ...........................................115
   A.3 Verilog-HDL code of Kplus2M .........................................116
   A.4 Verilog-HDL code of Kplus3M .........................................116
   A.5 Verilog-HDL Code of Kplus4M .......................................117
   A.6 Verilog-HDL Code of Booth3Mux ....................................117
   A.7 Verilog-HDL code of wallace_booth3_bias_8x8 ....................119
A.8 Verilog-HDL code of half adder ................................................................. 120
A.9 Verilog-HDL code of full adder ................................................................. 121
A.10 Verilog-HDL code of 16 bit Carry lookahead adder ............................. 121
A.11 Verilog-HDL code of 4-bit carry lookahead adder ................................ 122
A.12 Verilog-HDL code of full-adder in CLA adder ..................................... 122
A.13 Verilog-HDL code of testbench ............................................................. 123
CURRICULUM VITAE ................................................................................. 125
## LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPAR</td>
<td>Bit Plane Associative Router</td>
</tr>
<tr>
<td>CAD</td>
<td>Computer Aided Design</td>
</tr>
<tr>
<td>CLA</td>
<td>Carry Look-ahead Adder</td>
</tr>
<tr>
<td>C^3MOS</td>
<td>Clocked CMOS</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complimentary Metal-Oxide Semiconductor</td>
</tr>
<tr>
<td>CSA</td>
<td>Carry Save Adder</td>
</tr>
<tr>
<td>NPCPL</td>
<td>Normal Process Complementary Pass Transistor Logic</td>
</tr>
<tr>
<td>MOS</td>
<td>Metal-Oxide Semiconductor</td>
</tr>
<tr>
<td>MOSFET</td>
<td>Metal-Oxide Semiconductor Field-Effect Transistor</td>
</tr>
<tr>
<td>nMOS</td>
<td>n-Channel MOSFET</td>
</tr>
<tr>
<td>pMOS</td>
<td>p-Channel MOSFET</td>
</tr>
<tr>
<td>PLL</td>
<td>Phase Locked Loop</td>
</tr>
<tr>
<td>SAFF</td>
<td>Sense Amplifier Based Flip-Flop</td>
</tr>
<tr>
<td>SOC</td>
<td>System on Chip</td>
</tr>
<tr>
<td>UMC</td>
<td>United Microelectronics Company</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very Large Scale Integrated Circuits</td>
</tr>
<tr>
<td>WCSM</td>
<td>Wave Component Sampling Method</td>
</tr>
<tr>
<td>WPM</td>
<td>Wave Pipeline Multiplexed</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

FIGURES

Figure 2.1 Combinational logic circuit and input-output registers .................................................. 9
Figure 2.2 Conventional pipelining scheme ....................................................................................... 9
Figure 2.3 Temporal/spatial diagram for conventional pipelining ................................................... 10
Figure 2.4 Propagating waves on the same combinational circuit .................................................. 12
Figure 2.5 Temporal/spatial diagram of wave-pipeline operation .................................................... 12
Figure 2.6 Logic restructuring ......................................................................................................... 15
Figure 2.7 A combinational logic circuit ........................................................................................ 15
Figure 2.8 A combination logic circuit with balanced paths ............................................................ 16
Figure 2.9 TS diagram of mesochronous pipelining ...................................................................... 17
Figure 3.1 Partial product generation .............................................................................................. 19
Figure 3.2 Simple addition of partial products .................................................................................. 19
Figure 3.3 CSA multiplier with vector merging adder .................................................................... 21
Figure 3.4 CSA Multiplier with half adder tree ............................................................................... 22
Figure 3.5 Reduction of bits using half adder and full adder .............................................................. 23
Figure 3.6 Partial product reduction of 8x8 bit multiplier using Wallace tree [33] ......................... 24
Figure 3.7 Partial product reduction using (4,2) compressors [33] .................................................. 25
Figure 3.8 Modified Booth Algorithm (2 bit shift) [33] .................................................................. 26
Figure 3.9 Modified Booth Algorithm (3 bit shift) [33] .................................................................. 28
Figure 3.10 Partially redundant addition [33] .................................................................................. 29
Figure 3.11 Negative multiple generation [33] ................................................................................ 30
Figure 3.12 Carry look-ahead design of a 4-bit group ..................................................................... 32
Figure 3.13 16 bit carry-look-ahead adder ...................................................................................... 33
Figure 4.1 Proposed wave component sampling method (WCSM) ................................................... 36
Figure 4.2 Input-output delay of AND gate ...................................................................................... 36
Figure 4.3 An imaginary combinational circuit ............................................................................... 37
Figure 4.4 Data transition regions after 1st logic operation ............................................................ 38
Figure 4.5 Flip-flops inserted using mesochronous pipelining scheme ........................................ 39
Figure 4.6 Flip-flops inserted using proposed WCSM ...................................................................... 40
Figure 4.7 Flow chart of register insertion of WCSM ..................................................................... 41
Figure 5.1 Schematic diagram of 2 input NAND gate ...................................................................... 45
Figure 5.2 Schematic diagram of CMOS NAND2 with current limiting transistors ...................... 45
Figure 5.3 Generic transmission gate logic..................................................46
Figure 5.4 Transmission gate logic with 2 inverter cascades at the output........46
Figure 5.5 Block diagram of the simulation setup....................................47
Figure 5.6 Simulation of transmission gate logic block..............................48
Figure 5.7 Parametric sweep analysis of transistor sizes.............................49
Figure 5.8 Parametric sweep analysis of inverter ratio................................49
Figure 5.9 Glitch generation between transitions .....................................50
Figure 5.10 Graphical representation of output delay values.........................53
Figure 5.11 A positive pulse and its delayed version ................................54
Figure 5.12 Inverter cascade with different W/L ratios ..............................54
Figure 5.13 Signals obtained with normal and tuned inverter cascades ..........55
Figure 5.14 Generation of complementary outputs by using separate logic ....55
Figure 5.15 Simultaneous generation of complementary and normal output signals ....57
Figure 5.16 Simulation of generating normal and complementary outputs simultaneously ..58
Figure 5.17 Schematic diagram of half adder block ..................................60
Figure 5.18 A simulation example of half adder .......................................61
Figure 5.19 Graphical representation of the output delay values of half adder .....62
Figure 5.20 Layout of the half adder block .............................................62
Figure 5.21 Schematic diagram of full adder block ...................................64
Figure 5.22 Graphical representation of output delay values of full adder........66
Figure 5.23 Layout diagram of the full adder block ..................................66
Figure 5.24 Schematic diagram of AND gate .........................................67
Figure 5.25 Block diagram of partial product generator ............................68
Figure 5.26 Simulation of partial product generator ..................................69
Figure 5.27 Layout diagram of partial product generator ............................70
Figure 5.28 Schematic diagram of C$^2$MOS latch.................................71
Figure 5.29 Simulations of internal signals of the latch ..............................72
Figure 5.30 Propagating waves before sampling .....................................72
Figure 5.31 Propagating waves after sampling .......................................73
Figure 5.32 Block diagram of mesochronous multiplier ............................74
Figure 5.33 Block diagram of 8x8 bit multiplier using WCSM ....................75
Figure 5.34 Layout view of 8x8 bit multiplier using mesochronous pipelining ....77
Figure 5.35 Propagating waves after first partial product generator .............78
Figure 5.36 Propagating waves after the first half adder layer .....................78
Figure 5.37 Propagating waves after the first full adder layer .....................79
Figure 5.38 Propagating waves after the second full adder layer ........................................ 79
Figure 5.39 Propagating waves after sampling ................................................................. 80
Figure 5.40 Full pipelined multiplier .................................................................................. 82
Figure 5.41 Propagating waves before sampling ............................................................... 83
Figure 5.42 Design of Mux6x1 using generic transmission gate logic ............................... 86
Figure 5.43 Block diagram of 8xmux module ..................................................................... 87
Figure 5.44 Top level block diagram of Booth Encoder module ....................................... 88
Figure 5.45 Block diagram of wallace tree with modified Booth-2 algorithm ..................... 90
Figure 5.46 Schematic design of 4-bit carry look-ahead adder circuit ................................ 91
Figure 5.47 Carry generation logic ..................................................................................... 91
Figure 5.48 Modified carry generation circuit ................................................................. 92
Figure 5.49 Carry generation circuit of first stage with 5 levels of logic ......................... 92
Figure 5.50 Carry generation circuit of first stage with 4 levels of logic ......................... 93
Figure 5.51 16 bit adder with carry look-ahead logic ....................................................... 94
Figure 5.52 13 bit adder with carry look-ahead logic ....................................................... 95
Figure 5.53 16 bit carry save adder tree ............................................................................ 96
Figure 5.54 Block diagram of 16-bit CLA adder with balanced paths ............................. 99
Figure 5.55 16-bit carry lookahead adder with 4-input logic gates .................................. 101
Figure 5.56 4-input AND gate with serial current limiting transistor ............................. 102
Figure 5.57 2-input CMOS AND gate .............................................................................. 104
Figure 5.58 4-input CMOS And gate constructed with 2-input gates ............................. 105
LIST OF TABLES

TABLES

Table 3-1 Partial Product Selection of modified Booth-2 algorithm .................................................. 26
Table 3-2 Partial product selection table of Booth-3 algorithm .............................................................. 27
Table 5-1 Parameters of transistors ........................................................................................................ 48
Table 5-2 Optimized transistor parameters ............................................................................................ 51
Table 5-3 Output propagation times for all transitions of inputs ............................................................ 52
Table 5-4 Input combinations of generic trans. gate logic for NAND and AND gates ......................... 56
Table 5-5 Output propagation times of Q and NQ signals ..................................................................... 57
Table 5-6 Truth table of half adder ......................................................................................................... 58
Table 5-7 Output delay values of half adder for all input transitions .................................................... 61
Table 5-8 The truth table of full adder ...................................................................................................... 63
Table 5-9 Output delay values of full adder ............................................................................................ 65
Table 5-10 Transistor sizes of C2MOS latch .......................................................................................... 71
Table 5-11 Comparison between mesochronous and WCSM multiplier ............................................... 81
Table 5-12 Comparison between 5GHz mesochronous and WCSM multipliers .................................... 84
Table 5-13 Comparison of the multipliers with random inputs .............................................................. 85
Table 5-14 Wallace tree construction of Modified Booth-2 with 3 levels ............................................. 89
Table 5-15 Comparison of Wallace trees with mesochronous pipelining and WCSM ......................... 90
Table 5-16 Transistor count of 16-bit carry save adder implementation ............................................. 96
Table 5-17 Transistor count of 16-bit carry lookahead adder implementation .................................... 97
Table 5-18 Transistor count of 13-bit carry lookahead adder implementation .................................... 97
Table 5-19 Overall comparison of the multipliers .................................................................................. 100
Table 5-20 Delays of 4-input CMOS with a limiting pmos of W=900nm at the top ......................... 103
Table 5-21 CMOS_AND4_v1_delays with no limiting transistor at the top ........................................ 103
Table 5-22 Delay values of CMOS_AND2_v1 .................................................................................... 104
Table 5-23 Delay values of CMOS_AND4_v2 .................................................................................... 105
Table 5-24 Comparison of CMOS_AND4_v1 and CMOS_AND4_v2 ................................................. 106
CHAPTER 1

INTRODUCTION

In today’s high performance digital systems, pipelining technique is widely used to increase the operating frequency of a logic circuit. In conventional pipelining technique, the combinational logic circuit is divided into several sub-stages. Between these sub-stages, synchronization registers are inserted. Since the computation time between the synchronization registers is decreased, the overall operating speed of the logic circuit increases. It is possible to increase the operating frequency of the logic up to N times by using N levels of equally separated pipeline stages. However, the clocking overheads such as clock skew and setup-hold time requirements of the registers generally limit the operating frequency improvements. Also, the clock distribution and the power consumption of the synchronization registers are the other major drawbacks of the conventional pipelining.

Wave-pipelining [1] is another pipelining method in which the pipeline registers or latches are removed and the capacitances of the internal logic gates act as virtual storage elements. In wave-pipelining method, an input data vector is applied to the logic circuit, and before it arrives to the end of the logic circuit, another input data vector is sent. Therefore, multiple data vectors, which are also named as data waves, propagate on the circuit simultaneously. The important concept in wave-pipelining is that the circuit must be designed properly so that the cascading data waves do not collapse with each other. Therefore, the minimum and maximum delay variation of all the paths must be balanced in order to achieve wave-pipeline operation.

Previous work on wave pipelining is summarized as follows:

Wave pipelining method was first used in the design of the IBM System/360 Model 91 floating point unit [2], where the operating frequency of the chip was 2 times the normal frequency. Then, Cotton [3] formalized the wave pipelining method, and named it as
“maximal rate clocking”. Ekroot [4] developed linear programs which automatically insert delay elements to equalize the propagating waves.

Fishburn [5] investigated the performance improvements achieved by adjusting the path delays of the clock signal distributed to the flip-flops. He investigated the effects by both trying to minimize the clock period while avoiding clock hazards and maximizing the minimum safety margin for a given period.

In [6], a 63 bit bipolar population counter is designed by using wave pipelining. The circuit was operated at a frequency which is 2.5 times the normal operating frequency; therefore 2 or 3 waves propagate on the logic simultaneously.

Sakallah et al [7] developed timing models for multiphase synchronous clocking. They proposed a special class of clock schedules named as coincident multiphase clocks, which provide lower bound on the optimal clock cycle time.

Joy and Ciesielski [8] presented a methodology for minimizing the clock period for a given data path. They developed a linear program which minimizes the clock period by adjusting the clock delays to the input and output flip-flops for a logic block. Their method allows simultaneous signals to propagate in the logic without interference; therefore the clock period reduces significantly.

Wong et al [9] presented algorithms for automatically equalizing delays in combinational logic circuits to achieve wave pipelining. Their algorithms insert minimal number of active delay elements for balancing the input-output path lengths. The algorithms not only minimize the number of delay elements, but also optimize the power under delay constraints.

Gray et al [10] presented a method for high resolution sampling of a high speed data signal. Instead of using a high speed latch with a high speed clock signal, they used active delay elements to simultaneously propagate clock and data signals. Therefore, the resolution is controlled by the difference between clock and data signals. They implemented an integrated circuit, in which the delay is externally adjusted with a resolution of 25ps between 0 and 250ps.
In a different study by Gray et al. [11], the timing constraints for single and multiple stage systems with arbitrary feedback were presented. It is demonstrated that feedback loops impose additional constraints on the minimal and maximal clock period. A linear program was also used to optimize the minimum clock period.

A 250-MHz adder in 2-µm CMOS technology is presented in [12]. 16-bit parallel adder was designed using wave pipelining concept, and it has a wave pipelining degree of 9. They developed a biased CMOS cross-coupled NAND gate in a custom layout, which has minimal input data dependency at the outputs.

Ghosh and Nandy [13] designed a high performance wave pipelined 8x8 bit multiplier using CMOS. They used a single generic block in normal process complementary pass transistor logic (NPCPL) for equalizing the propagation paths in the design. The multiplier was implemented using 0.8µm CMOS technology. It operates at a speed of 400MHz, and dissipates a total power of 0.6W.

In [14], a 16-Mb BiCMOS SRAM is designed using 0.4µm BiCMOS process. This SRAM, which has a total size of 512Kw*8b*4, includes a PLL self-timing generator and incorporates 2 stage wave pipeline operation.

A 4-Mb synchronous wave pipeline SRAM was designed and fabricated by using 0.25µm CMOS technology in [15]. This multiplier operates at a speed of 300MHz, resulting in a bandwidth of 2.4GB/s.

In [16], a wave pipelined SRAM of 16kb with dual sensing latch circuit was implemented using 0.25µm CMOS technology. This SRAM has an access time of 2.6ns at 2.5V supply voltage.

In [17], wave pipelining concept is reviewed with special emphasis on CMOS. The effects of temperature, voltage and process parameters on CMOS wave-pipelining are explained. The conventional pipelining considers only the worst case timing constraints; however in wave pipelining both the worst case and the best case timing constraints depending on temperature must be handled. A dynamically adaptive clocking mechanism is proposed, which compensates the effects of environmental fluctuations and process parameter deviations. A
a dynamically adaptive power supply is also proposed. The dependency of output delay on input pattern in conventional CMOS design was analyzed in detail. A biased CMOS gate is also proposed for reducing the input dependency at the output.

In [18], valid clocking frequencies of wave pipelining are investigated. They used a new representation named as timed boolean functions and derived analytical expressions for valid clocking intervals.

Boemo et al [19] studied wave pipelining on FPGA’s. They showed that wave pipelining can be achieved by using automatic place and route, if the circuit has same number of Look-up-tables (LUTs) in all paths.

In [20], an excellent tutorial on wave pipelining is given. They explained the principles of wave pipelining in detail, including the timing constraints, circuit and timing models, internal node constrains etc. The sources of delay variations and the Computer Aided Design (CAD) tools developed for synthesis and placement-routing of wave pipelined circuits are also explained.

In [21], hybrid wave pipelining method is proposed. In hybrid wave pipelining method, wave pipelined sub stages are composed to form pipeline stages. A bit plane associative router (BPAR) is designed with hybrid wave pipelining method using 0.5µm CMOS technology.

Wave Pipeline Multiplexed (WPM) routing technique is proposed in [22] in which multiple signals are sent in a single wire interconnect within a clock period. They suggested that WPM routing technique can be applied to both inter-core and intra-core interconnects in any system-on-chip (SoC) or microprocessor design. The number of total routing channels can be reduced by 50% without any performance loss in the throughput. They analyzed the application of WPM routing technique to a design including 40 million transistors, and they showed that total number of metal layers is decreased by 20% with only 4% increase at the dynamic power without any loss in the throughput.

A study in [23] showed that, the power dissipation in long global wires is significantly reduced by adding wave pipeline stages to global wires and by lowering the supply voltage of repeaters, without any performance loss.
In [24] a novel pipelining scheme named mesochronous pipelining is proposed. In this method, data and clock signals propagate together, and when the minimum and maximum delay difference of a path reaches the tolerable value, then the signals in this logic depth are all sampled. They implemented an 8x8 bit multiplier to compare their method with conventional pipelining scheme, and a speedup of 1.7 was achieved by using fewer pipeline stages and pipeline registers.

In [25] a new pipeline method, named as MOUSETRAP, is proposed. This method uses simple latches and control structures with an efficient event driven protocol. They claim that this pipelining method has a performance comparable to that of wave pipelining with much less design complexity.

In [26] a pipelining method named as surfing pipelines is proposed. This method is similar to the wave pipelining, however in this method timing events are propagated along the pipeline and events in the data path are matched with the timing events. Therefore, timing uncertainty is reduced.

Voltage scaling, wire sizing, and repeater insertion are simultaneously applied in [27] for achieving high performance, low power, and low area on wave-pipelined interconnect circuits. They found that optimal supply voltage is twice the threshold voltage for low power applications. The throughput-per-energy-area in their method is 10% lower than that of low-voltage differential signaling (LVDS).

Schinkel et al [28] used wave pipelining in a network on chip design, and demonstrated that the link power is reduced by a factor of 3.3 and data rate is increased by 80%.

In [29] a double data rate, wave-pipelined interconnect for asynchronous network on chips is proposed. They used interleaved lines, misaligned repeaters and clock gating for low power and high speed chip interconnects.

In [30] a synchronizing logic gate, which has an almost constant gate delay, is proposed for wave pipelining. This logic gate is used as an intermediate latch for synchronizing data.
paths. An 8x8 bit multiplier is designed using 90nm technology, and it has an operating speed of 3.57 GHz.

All the conventional pipelining, wave-pipelining, hybrid pipelining and mesochronous pipelining methods have a common property: A data wave is sampled whenever it reaches to the synchronization stage, which is composed of flip-flops or latches for sampling the data waves. In fact, a data wave is composed of several signal components, and all of these components may have different maximum and minimum delay differences.

In this thesis, a new wave-pipelining methodology, which is named as Wave Component Sampling Method (WCSM), is developed. This method permits individual sampling of the signal components of a wave. Only the component of a wave, whose minimum and maximum delay difference value exceeds the tolerable value, is sampled. The other components of the wave, whose minimum and maximum delay differences do not reach the tolerable value, continue to propagate on the combinational circuit without being sampled. Therefore, the number of synchronization registers is decreased significantly in this proposed method. The area and power consumption due to these synchronization registers, and the associated power of the clock distribution are also decreased.

The organization of this dissertation is as follows:

Chapter 2 describes the theoretical background of current pipelining methods. In Chapter 3, different multiplier structures including Wallace trees, booth encoders and carry look-ahead adders are overviewed.

Chapter 4 describes the details of the proposed WCSM. The advantages and disadvantages of the proposed method compared with the other pipelining methods are also given.

In Chapter 5, the application of WCSM to different multiplier structures are analyzed. Two 8x8 bit carry save adder multipliers are implemented using mesochronous pipelining scheme and WCSM, for comparing the methods. WCSM is also applied to other multiplier structures including booth encoder, Wallace tree and carry look-ahead adder. The optimization of the sub blocks and the performance gain of WCSM are described in detail.
In Chapter 6 the thesis work is summarized, and concluding remarks are given. Some suggestions are made for future improvements and possible utilizations of the proposed wave component sampling method.
CHAPTER 2

PIPELINING METHODS

The following parameters are used to explain the timing constraints for obtaining the maximum operating frequency for different pipelining methodologies:

- $D_{\text{MIN}}$: Minimum propagation time in the combinational circuit.
- $D_{\text{MAX}}$: Maximum propagation time in the combinational circuit.
- $T_{\text{CLK}}$: Minimum clock period.
- $\Delta C$: Constructive clock skew.
- $\Delta U$: Unconstructive clock skew.
- $T_S, T_H$: Setup-hold times of registers.
- $D_R$: Propagation delay of a register.

2.1 Conventional Pipelining

A combinational logic circuit with its input and output registers are shown in Figure 2.1. An input data is sent to the combinational circuit with the rising or falling edge of the clock. Before another data is applied, the combinational circuit must complete the logical operation. Considering the propagation delay of the input registers, the setup time requirement of the output registers, and the clock skew between the input and the output registers, the minimum clock period for that circuit is shown in Equation (1).

$$T_{\text{CLK}} \geq D_{\text{MAX}} + D_R + T_S + \Delta U \quad (1)$$
Figure 2.1 Combinational logic circuit and input-output registers

Generally, $D_R$, $T_S$ and $\Delta U$ cannot be decreased further; therefore the only way for decreasing the clock period is to decrease $D_{MAX}$. In conventional pipelining method, pipeline registers or latches are inserted to increase the operating frequency by decreasing the maximum propagation time, $D_{MAX}$. Figure 2.2 shows the $N$ stage pipelined version of the same combinational circuit. If the pipeline registers are separated with equal propagation delays, then the propagation delay between consecutive pipeline registers becomes $D_{MAX}/N$. In this case, the minimum clock frequency can be expressed by

$$T_{CLK} \geq D_{MAX}/N + D_R + T_S + \Delta U$$

(2)

At every rising or falling edge of the clock signal, a new input data vector is applied to the circuit. At the end of $N^{th}$ clock cycle, the first input vector reaches to the output, therefore the latency between input and output is $N$. Assume that the data throughput is continuous, such that at every clock edge a different input vector is applied. After the initial latency of $N$ clock cycles, a new output is obtained at every clock cycle. Therefore, the clocking overheads such as setup-hold times, clocking skew, and register propagation delay are ignored, then the data throughput increases up to $N$ times.

A temporal-spatial diagram shows the transition times of the signals at different locations of the combinational circuit. The “Y” axis represents the logic depth of the combinational
circuit, and the “X” axis represents time. Figure 2.3 shows the temporal/spatial diagram for conventional pipelining. The fastest signals arrive at the output after $D_{\text{MIN}}$ seconds, and the slowest signals arrive at the output after $D_{\text{MAX}}$ seconds. The shaded region, which is between $D_{\text{MIN}}$ and $D_{\text{MAX}}$, is the transition region, where the combinational logic blocks change their state and the data is unstable. In the other regions, the combinational circuits are idle, keeping their states. In a conventional pipelined system, operating clock frequency is limited by the slowest path in the logic stages. As it is seen from Figure 2.3, a new input data vector is accepted after all the combinational operations are calculated by the logic stages, i.e. a new data can only be launched after the slowest signal arrives at the output register. The setup time and clock uncertainty must also be handled.

![Temporal/spatial diagram for conventional pipelining](image)

Figure 2.3 Temporal/spatial diagram for conventional pipelining

As it is seen from Figure 2.3, the combinational logic blocks are idle for the vast majority of time. Although the fast signals arrive early at the output, they must wait for the slowest signal for sampling.

The disadvantages of conventional pipelining can be listed as:

- Pipeline registers increase the area and power consumption of the circuit.
- Clock distribution to the pipeline registers with minimal skew is a challenging task.
• The combinational logic blocks are idle for the vast majority of time, therefore logic utilization is small.

• The slowest path determines the operating speed of the entire circuit.

The advantages of the conventional pipelining:

• The design complexity is lower than wave-pipelining.

• Since only worst case timing is considered, it is less sensitive to temperature and process parameter variations.

2.2 Wave pipelining

In wave-pipelining method, the pipeline registers are removed from the circuit. The internal capacitances of the logic gates act as virtual storage elements, which store the states of the pipelining data. An input data wave is sent to the combinational circuit, and before it reaches to the output, another data wave is sent. Therefore, multiple data waves propagate on the logic circuit simultaneously. While propagating, those waves encounter with different delays. For proper operation, the fastest signal component of a data wave should not catch the slowest signal component of the preceding wave. Therefore, in wave-pipelining method, the minimum and maximum delays of the waves are tried to be made equal by slowing down the fast components.

Figure 2.4 shows multiple waves propagating on the same combinational circuit. There are 5 different data waves propagating on this circuit from left to right. The shaded regions between the waves are data transition regions in which the data is unstable. As seen in Figure 4, the width of the transition region increases and the width of the stable wave decreases while propagating on the logic circuit. The waves must be sampled, before the width becomes too small, which creates setup and hold time violations.
Figure 2.4 Propagating waves on the same combinational circuit.

Figure 2.5 shows the temporal/spatial diagram of the wave-pipelining operation. Similar to Figure 2.4, the shaded regions are transition regions and the width of those transition regions increases while propagating on the direction of logic depth. As it is seen from Figure 2.5, before a data wave reaches to the output register, another wave is launched. Therefore, multiple data waves propagate on the combinational logic circuit simultaneously. Analogues with the eye diagram of telecommunication theory, an adequate aperture is needed for proper sampling of the data waves at the output.

Figure 2.5 Temporal/spatial diagram of wave-pipeline operation
2.2.1 Timing constraints

There are two constraints for sufficient aperture for proper sampling. The first constraint comes from setup time requirement of the sampling flip-flops: The slowest signal component of a data wave must arrive at least $T_S$ seconds before the sampling edge of the clock signal. The second constraint comes from the hold time requirement: The fastest signal component of the previously launched data wave arrives at time $T_{CLK} + D_{MIN} + D_R$. This value must be larger than the hold time requirement of the flip-flops. Let $T_L$ be the latching time of the data waves at the output, where

$$T_L = N \cdot T_{CLK} + \Delta C$$

(3)

$N$ represents the number of clock cycles passed during the propagation of a data wave from the input register to the output register. It also represents the degree of wave-pipelining, i.e. the number of data waves propagating on the combinational logic simultaneously.

Equation (4) describes the lower bound on the latching time, which comes from the setup time requirement of the register:

$$T_L > D_R + D_{MAX} + T_S + \Delta U$$

(4)

Equation (5) describes the upper bound on the latching time, which comes from the hold time requirement due to the fastest signal component of the succeeding wave.

$$T_L < T_{CLK} + D_{MIN} + D_R - (\Delta U + T_H)$$

(5)

Combining (4) and (5), the constraint on the operating frequency can be obtained as

$$T_{CLK} > (D_{MAX} - D_{MIN}) + T_S + T_H + 2\Delta U$$

(6)

Equation (6) shows that the minimum clock period depends on $D_{MAX} - D_{MIN}$ rather than $D_{MAX}$. Therefore, to increase the operating frequency, $D_{MIN}$ and $D_{MAX}$ are tried to be balanced.
2.2.2 Sources of Delay Differences

The major sources of delay differences can be given as:

1. **Data dependent delay variation**
   A combinational logic gate has a propagation delay between its inputs and outputs. This delay is not constant; rather it depends on the input pattern. Consider a 2 input NAND gate designed with static CMOS logic. If one of its inputs is at logic-0 and the other one is at logic-1, then there will be one path for pulling up the output load. If both of the inputs are at logic-0, then there will be 2 paths for pull-up, which creates double driving strength. Therefore, the output delay is much lower. If a CMOS logic gate with 3 or more inputs is used, then the variation of the output delay becomes higher.

2. **Process dependent delay variation**
   The delays of the gates are strictly dependent on the process parameters. The circuits produced at different manufacturing runs will have different delay values. Furthermore, the circuits produced at the same wafer will also have different delay values at the output.

3. **Temperature dependent delay variation**
   The delays of the gates depend on the temperature. The wave-pipelined circuits must be properly designed to compensate for the delay variation due to temperature.

4. **Delay variation due to the supply noise**
   The noise in the power supply will produce additional delay variation at the outputs of the logic gates. Also, the coupling capacitances between adjacent wires will produce delay variation.

2.2.3 Logic Restructuring

For equalizing the minimum and maximum delays in the logic circuit, it may be re-structured. The main idea is to balance the logic so that all the signals encounter with the same number of logic gates while propagating. Figure 2.6 shows the logic restructuring
technique. Both of the circuits have same function. In the upper circuit, D input arrives to the output early. By restructuring the logic, all the paths are balanced, which can be seen in Figure 2.6. In some situations, logic restructuring increases the number of logic blocks required to obtain same logic operation.

![Fig 2.6 Logic restructuring](image)

### 2.2.4 Delay Insertion

After the logic restructuring, there can still be unbalanced paths. For balancing such paths, inverters or buffers are inserted to slow down the fast paths. Figure 2.7 shows a combinational logic and Figure 2.8 shows the delay buffer inserted for slowing down the fast path.

![Fig 2.7 A combinational logic circuit](image)
2.2.5 The advantages and disadvantages of wave-pipelining

The advantages and disadvantages of wave-pipelining can be summarized as [17]:

Advantages of wave-pipelining:
- Very high clock rates can be obtained.
- No partitioning in the combinational logic is performed; therefore unequal partitioning is not a problem.
- Reduced clocking latency overhead.
- Clock distribution problem is reduced because fewer registers are used.
- Simultaneous switching noise is reduced.
- Power consumption and silicon area due to flip-flops and clock buffers are reduced.

Disadvantages of wave-pipelining:
- Design complexity is increased due to delay balancing.
- Power consumption and area increase because of delay balancing.
- Debugging and testing are difficult.
- Process parameters and environmental changes effect much more than conventional pipelining.

2.3 Hybrid Wave Pipelining

Hybrid wave pipelining is another pipelining method proposed in [21]. In this method, wave pipelined sub stages are composed to form pipeline stages. In wave-pipelining the clock cycle time is determined by the delay difference value at the output register. However, in hybrid wave-pipelining combinational logic is partitioned into several stages, and the clock
cycle time is determined by the delay difference value of a stage with the largest delay variation. Therefore, the operating frequency of the logic circuit is increased.

2.4 Mesochronous Pipelining Scheme

Mesochronous pipelining scheme [24] is similar to the hybrid pipelining method. In mesochronous pipelining scheme, clock signal is delayed so that it propagates with the data. Delay elements, which give the same delay value with the corresponding combinational logic stage, are inserted in the clock signal path.

The cascading registers form wave-pipeline regions, therefore multiple data waves propagate on the combinational logic circuit simultaneously. Figure 2.9 shows the temporal-spatial diagram of mesochronous pipeline operation for a three stage pipelined system. The second stage is assumed to have maximum delay variation, therefore the clock cycle time is determined by this stage. Equation (7) gives the requirement on minimum clock period for mesochronous pipelining scheme.

$$D_{\text{MAX}(j)} - D_{\text{MIN}(j)} + T_S + T_H + 2\Delta U < T_{\text{CLK},m}$$  \hspace{1cm} (7)
CHAPTER 3

MULTIPLIER STRUCTURES

Let X and Y be two unsigned binary numbers, which are M and N bits wide. If we express X and Y in their binary representation with $X_i, Y_j \in \{0,1\}$:

\[
X = \sum_{i=0}^{M-1} X_i 2^i \\
Y = \sum_{j=0}^{N-1} Y_j 2^j
\]

Then the multiplication of X and Y is defined as:

\[
Z = X \cdot Y = \left( \sum_{i=0}^{M-1} X_i 2^i \right) \left( \sum_{j=0}^{N-1} Y_j 2^j \right) \\
= \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} X_i Y_j 2^{i+j}
\]

Multiplication operation is composed of generating partial products and addition of those partial products. Partial product generation is the AND operation of a multiplier bit with all the bits of the multiplicand, which can be seen in Figure 3.1.
Partial products and simple addition of those partial products for an 8x8 bit multiplier can be seen in Figure 3.2. In this figure there are 64 dots, which represent the 64 partial products generated by AND operation of the corresponding bits of the multiplicand and the multiplier. The partial products are shifted to the corresponding weight of the multiplicand bit for addition.

For inputs which are M and N bits wide (M≤N), the simplest multiplier can be composed of a single N bit adder with 2-inputs [31]. The partial products are generated and added at every clock cycle, so the multiplication operation is completed at M clock cycles. This multiplier is named as iterative multiplier.

To increase the speed of the multiplier, partial products are generated and added in parallel. For this purpose, adder trees can be used. In adder tree structures, the output delay is log(N),
instead of N. In this architecture, the adders are carry propagate adders. The carry propagate addition operation is very time-consuming which increases the critical path delay of the circuit. To overcome this problem, Wallace trees are generally used in the literature.

3.1 Carry Save Adder (CSA) Multiplier

All the partial product bits must be added in the multiplication. Multiplication result does not change when the carry bits are sent diagonally to the next stages, instead of sending to the right. In the carry save multiplier structure, the carry outputs are sent diagonally to the next stage for addition. At the last stage, a vector merging adder is used to merge the carry and sum outputs. Figure 3.3 shows the block diagram of carry save multiplier with vector merging adder. As it is seen from the figure, the carry output of a full adder is fed back to the carry input of the neighboring full adder. In pipelined designs, the feedback paths must be avoided for increasing the throughput, therefore a half adder tree can be used instead of carry propagate addition. The blocks circulated with dashed line are replaced with a half adder tree, which is shown in Figure 3.4.
Figure 3.3 CSA multiplier with vector merging adder
3.2 Wallace Trees

The carry propagate addition is the most time consuming operation in multiplication. In order to avoid using carry propagate addition, Wallace proposed a method [32]. In this method, by using an adder tree composed of full adders and half adders, any number of partial products can be decreased to 2 numbers without any carry propagate addition. In the last step, these 2 numbers are added using a fast carry propagate adder.

Figure 3.5 shows the reduction of two and three partial product bits using a half adder and a full adder, respectively. The carry output is shifted to the left by one bit; therefore the weight of it is doubled. Figure 3.6 shows the complete partial product reduction of an 8x8 bit multiplier using a Wallace tree. It is seen that in the first stage, 16 full adders and 5 half adders are used. In the second stage, 10 full adders and 6 half adders are used. In the third stage 6 full adders and 6 half adders are used, while in the fourth stage 6 full adders and 5
half adders are used for reducing the partial products. As it is seen from the figure, no carry propagate addition is used and all the partial products are reduced to 2 numbers, which must be added using carry propagate addition.

Full adder is used as a compressor, which compresses 3 bits into 2 bits. Wallace trees using (3,2) compressors suffer from the irregularity in the routing [33]. There are more regular compressors, like (4,2) compressors, which compresses 4 bits into 2 bits. Figure 3.7 shows the partial product reduction using (4,2) compressors, in which the routing is much more simple than Wallace trees.

![Figure 3.5 Reduction of bits using half adder and full adder](image-url)
Figure 3.6 Partial product reduction of 8x8 bit multiplier using Wallace tree [33]
3.3 Booth Encoding

3.3.1 Booth 2 Algorithm

Booth’s algorithm [34] is a well known algorithm which is used to decrease the number of partial products used in the multiplication. In this algorithm, the multiplier bits are grouped into pairs of two bits to select the partial products from the set of \{0, M, 2M, 3M\}, which are pre-calculated. The calculation of 2M is completed by only shifting the multiplier M by one bit to the left. However, the calculation of 3M requires a carry propagate addition of M and 2M. Therefore, 3M is called as a hard multiple. In order to avoid this carry propagate addition, modified Booth algorithm, which selects partial products from the set of \{0, M, 2M, 4M-M\}, is used. In this algorithm, instead of 3M multiple, either 4M or –M is used, depending on the adjacent multiplier groups. Table 3.1 shows the partial product selection table and Figure 3.8 shows the modified booth algorithm. The multiplier bits are grouped
into pairs of 3 bits, and they are used to select multiplicands. Negative multiples can be obtained by using 2’s complement logic, so if the selected partial product is negative, then all the bits are negated and then a 1 (which is shown as S-bit) is added to complete the 2’s complement operation. As it is seen from the figure, the number of rows of partial products is decreased from 8 to 5. In general, n partial products are decreased to the biggest integer which is smaller than or equal to (n+2)/2.

Table 3-1 Partial Product Selection of modified Booth-2 algorithm.

<table>
<thead>
<tr>
<th>Multiplier bits</th>
<th>Selection S</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>+0 0</td>
</tr>
<tr>
<td>001</td>
<td>+M 0</td>
</tr>
<tr>
<td>010</td>
<td>+M 0</td>
</tr>
<tr>
<td>011</td>
<td>+2M 0</td>
</tr>
<tr>
<td>100</td>
<td>-2M 1</td>
</tr>
<tr>
<td>101</td>
<td>-M 1</td>
</tr>
<tr>
<td>110</td>
<td>-M 1</td>
</tr>
<tr>
<td>111</td>
<td>-0 1</td>
</tr>
</tbody>
</table>

Figure 3.8 Modified Booth Algorithm (2 bit shift) [33]
3.3.2 Booth 3 Algorithm

The multiplier bits can be grouped with pairs larger than 3 bits, so the amount of shift operation between partial products can be greater than 2. In booth 3 algorithm, the partial products are selected from the set of \{±0, ±M, ±2M, ±3M, ±4M\}. There is also a hard multiple of 3M in Booth-3 algorithm. Table 3.2 and Figure 3.9 show the partial product selection table of Booth-3 algorithm and the reduction of partial products, respectively. The number of rows of partial products is decreased from 8 to 3.

Table 3-2 Partial product selection table of Booth-3 algorithm.

<table>
<thead>
<tr>
<th>Multiplier bits</th>
<th>Selection</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>+0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>+M</td>
<td>0</td>
</tr>
<tr>
<td>0010</td>
<td>+M</td>
<td>0</td>
</tr>
<tr>
<td>0011</td>
<td>+2M</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>+2M</td>
<td>0</td>
</tr>
<tr>
<td>0101</td>
<td>+3M</td>
<td>0</td>
</tr>
<tr>
<td>0110</td>
<td>+3M</td>
<td>0</td>
</tr>
<tr>
<td>0111</td>
<td>+4M</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>-4M</td>
<td>1</td>
</tr>
<tr>
<td>1001</td>
<td>-3M</td>
<td>1</td>
</tr>
<tr>
<td>1010</td>
<td>-3M</td>
<td>1</td>
</tr>
<tr>
<td>1011</td>
<td>-2M</td>
<td>1</td>
</tr>
<tr>
<td>1100</td>
<td>-2M</td>
<td>1</td>
</tr>
<tr>
<td>1101</td>
<td>-M</td>
<td>1</td>
</tr>
<tr>
<td>1110</td>
<td>-M</td>
<td>1</td>
</tr>
<tr>
<td>1111</td>
<td>-0</td>
<td>1</td>
</tr>
</tbody>
</table>
Booth 4 and higher booth algorithms are also possible, but the partial product selection logic becomes too complicated. Also, hard multiples (5M and 7M) are difficult to obtain, so it is not feasible to use booth4 and higher.

### 3.3.3 Redundant Booth Algorithm

Hard multiple of 3M in the booth 3 algorithm requires carry propagate addition. In order to overcome this problem, fully redundant booth algorithm is used. In this algorithm, all the partial products are represented by their redundant form (i.e. Redundant form of 3M is M+2M). The number of the dots is doubled, because of this redundant representation. Therefore, fully redundant form is not feasible.

Partially redundant booth algorithm is used to compute hard multiple of 3M, in which small length adders are used. Figure 3.10 shows the partially redundant addition of 16 bit M and 2M, using small length adders of 4 bits. Carry propagate addition of 4 bits is performed in the small adders; however no carry signal propagates between these adders. Therefore, the length of the carry propagation is reduced and limited to 4. As it is seen from Figure 3.10, the number of dots is much smaller than the fully redundant representation.

A problem with this partially redundant representation arises when negative partial products are required. As it is seen from Figure 3.11, large gaps of 0’s become large gaps of 1’s when negated. Also considering the addition of 1’s in the LSB’s of partially redundant represented
numbers, in the worst case (all partial products are negative) same hardware of fully redundant representation plus the hardware of small length adders are needed. Therefore, the problem is to obtain negative multiples from positive multiples, or vice versa.

Figure 3.10 Partially redundant addition [33]
3.4 Carry Propagate Addition

The final 2 numbers which are produced by Wallace trees must be added using carry propagate addition. In ripple carry addition, each full adder must wait until the previous carry output has been calculated in order to begin calculation of its carry output and sum. In order to speed up this carry propagate addition, carry look-ahead adders are widely used. In carry look-ahead adders, the carry outputs are generated before the sum outputs.

Let A and B are two n-bit numbers, and S is the summation of A and B. In binary expanded form:

\[
A = \sum_{k=0}^{n-1} a_k 2^k
\]

Let

- \( ab \): Boolean AND operation of a and b
- \( a \| b \): Boolean OR operation of a and b
- \( a \wedge b \): Boolean EXOR operation of a and b
- \( a + b \): Summation of a and b
The addition of A and B and the carry input \(c_0\) can be computed as:

\[
s_k = a_k ^ b_k ^ c_k\\
\]

\[
c_{k+1} = a_k b_k \| a_k c_k \| b_k c_k\\
\]

\[k=0,1,...,n-1\]

The sum and carry outputs can be interpreted using auxiliary signals, \(g_k\) (generate) and \(p_k\) (propagate). If propagate signal \(p_k\) is 1, then the incoming carry signal to that stage is propagated to the next stage. Similarly, if generate signal \(g_k\) is 1, then a carry signal is generated at that stage and it is sent to the next stage. For obtaining the propagate signal, the two equations shown below can be used. They both give the same result while generating carry out.

\[
g_k = a_k b_k\\
\]

\[
p_k = a_k \| b_k\\
\]

\[
= a_k ^ b_k\\
\]

The carry signal can be interpreted using generate and propagate signals and the incoming carry signal, such that:

\[
c_{k+1} = g_k \| p_k c_k\\
\]

Using these equations, a carry output can be calculated in terms of preceding generate and propagate signals and a carry signal at any bit position:

\[
c_{k+1} = g_k \| p_k g_{k-1} \| p_k p_{k-1} g_{k-2} \| p_k p_{k-1} p_{k-2} c_{k-2}\\
\]

This leads to two new functions:

\[
g(j,k) = g_j \| p_j g_{j-1} \| p_j p_{j-1} g_{j-2} \| ... \| p_j p_{j-1} ... p_{k+1} g_k\\
\]

\[
p(j,k) = p_j p_{j-1} ... p_{k+1} p_k\\
\]

Let \(GG\) and \(PG\) be the group generate and group propagate signals of a 4-bit group respectively.

\[
GG = g(3,0) = g_3 \| p_3 g_2 \| p_3 p_2 g_1 \| p_3 p_2 p_1 g_0\\
\]

\[
PG = p(3,0) = p_3 p_2 p_1 p_0\\
\]

The carry output of a 4 bit carry look-ahead adder is expressed using group propagate and group generate signals:
\[ c_i = GG + PG\cdot c_0 \]

GG and PG signals are generated immediately without using carry inputs. When the carry input has come, then the 4 level shifted carry output is generated, therefore the carry propagation is completed with 2 clocks, instead of 4.

Using 4 of these group signals, a super group can be composed. Figure 3.12 shows the carry look-ahead design of a 4-bit group and Figure 3.13 shows the super group signals of 16 bit addition. These super groups can also be combined to make larger groups.

![Figure 3.12 Carry look-ahead design of a 4-bit group](image_url)
For demonstrating the speed improvements using carry look-ahead adders, the timing details of an addition of two 16 bit numbers using 4 carry look-ahead adders of 4 bits and 1 carry look-ahead logic are investigated.

Starting at time 0,

- Individual $p_i$ and $g_i$ signals are calculated at time 1. ($p_i = a_i \parallel b_i$; and $g_i = a_i \oplus b_i$)
- Individual $c_i$ signals are calculated at time 3 for the first CLA. (AND operation is completed at time 2 and then OR operation is completed at time 3)

\[
\begin{align*}
  c_1 &= g_0 + p_0 \cdot c_0 \\
  c_2 &= g_1 + p_1 g_0 + p_1 \cdot p_0 \cdot c_0 \\
  c_3 &= g_2 + p_2 g_1 + p_2 \cdot p_1 g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_0 \\
  c_4 &= g_3 + p_3 g_2 + p_3 \cdot p_2 g_1 + p_3 \cdot p_2 \cdot p_1 g_0 + p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_0
\end{align*}
\]

- Group propagate signals (PG) are calculated at time 2.

\[
\begin{align*}
  PG[1] &= p_3 p_2 p_1 p_0 \\
  PG[2] &= p_7 p_6 p_5 p_4 \\
  PG[3] &= p_11 p_10 p_9 p_8 \\
\end{align*}
\]
- Group generate signals (GG) are calculated at time 3.
  
  \[
  \begin{align*}
  GG[1] &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 \\
  GG[2] &= g_7 + p_7 g_6 + p_7 p_6 g_5 + p_7 p_6 p_5 g_4 \\
  GG[3] &= g_{11} + p_{11} g_{10} + p_{11} p_{10} g_9 + p_{11} p_{10} p_9 g_8 \\
  GG[4] &= g_{15} + p_{15} g_{14} + p_{15} p_{14} g_{13} + p_{15} p_{14} p_9 g_{12}
  \end{align*}
  \]

- Look-ahead Carry Unit (LCU) generates the inputs required by Carry Look-ahead Adder Blocks at:
  
  - Time 0 for the first CLA
  - Time 5 for the second CLA \((c_4 = GG[1] + PG[1]c_0)\)

- Calculation of Sum outputs are calculated at:
  
  - Time 4 for the first CLA \((s_i = a_i b_i c_i \text{ and } c_i \text{ are calculated at time 3 for the first CLA})\)
  - Time 8 for the second CLA (carry input is generated at time 5, and individual \(c_i\) signals are generated at time 7 for the second CLA)
  - Time 8 for the third CLA (carry input is generated at time 5, and individual \(c_i\) signals are generated at time 7 for the third CLA)
  - Time 8 for the fourth CLA (carry input is generated at time 5, and individual \(c_i\) signals are generated at time 7 for the fourth CLA)

- The carry output of the 16 bit adder \((c_{16})\) is calculated at time 5
  
  \[
  \]

- The carry output of the entire adder is generated with a 5 gate-delay and the overall addition is completed with 8 gate delays. If carry look-ahead addition is not used and ripple carry addition is used, then the overall addition takes 31 gate delays.
A data wave propagating on the combinational logic circuit has many signal components. The combinational logic circuit is also composed of many sub stages of logic gates, which have different propagation delays. Therefore, all of the signal components experience different delays while propagating on the logic. The minimum and maximum delays of all the wave components may be different.

In conventional pipelining, wave-pipelining and mesochronous pipelining schemes, all of the components of a propagating wave are sampled at the same time whenever they arrive to a synchronization stage. In the proposed WCSM, only the components of a wave, whose minimum and maximum delay differences reach to the tolerable value, are sampled. The other components of the wave are aligned with the sampled components by using delay elements. Figure 4.1 shows the basic operation principle of WCSM. FF and DL represent flip-flops and delay elements, respectively. As it is seen in Figure 4.1, only some of the signal components of propagating waves, whose delay difference values reach to the tolerable value, are sampled by using flip-flops.
4.1 Principles of Wave Component Sampling Method

Consider the 2 input AND gate in Figure 4.2. A0, B0 and C0 are the initial values, and A1, B1 and C1 are the final values of the inputs and the output, respectively. Assume that, both of the inputs change their state at the same time instant. Then, the output C changes its state with a propagation delay, which depends on the inputs. The propagation delay has a mean value of $T_P$ seconds and a variation of $\Delta T_P$ seconds as shown in Figure 4.2. If one or both of the inputs have a transition variation of $\Delta T_{in}$ seconds, then the transition variation of the output becomes approximately $\Delta T_{in} + \Delta T_P$ seconds.

Figure 4.1 Proposed wave component sampling method (WCSM)

Figure 4.2 Input-output delay of AND gate
For describing WCSM in detail, an imaginary combinational circuit is constructed as in Figure 4.3. In this circuit, 3 combinational logic blocks, which are named as F1, F2, and F3, are used. Assume that the mean values of the propagation delays of F1, F2 and F3 are all 400ps and the transition variation of F1, F2 and F3 are 20ps, 40ps, and 80ps, respectively. Also, assume that the delay elements used in this combinational logic circuit do not have a transition variation at the output; i.e. they only give a time shift to the incoming signal. By using these delay elements, all the branches which enter to the same node are aligned with each other.

As seen from Figure 4.3, 4 inputs are applied to the circuit. It is assumed that they are all applied at the same time instant, i.e. there is no delay difference between them at the beginning. In Figure 4.3, total propagation times are not shown, only the minimum and maximum delay differences are displayed. While propagating on the circuit, the components of the wave experience different delays. At the beginning, all of the components have a delay difference of 0ps. After first logic stage, 4 components of the wave have delay differences of 0ps, 20ps, 40ps, and 80ps, respectively. After second logic stage, the delay differences become 0ps, 40ps, 80ps, and 160ps, respectively. Obviously, the delay differences are
increasing after each logic operation. After 4th logic stage, maximum delay difference becomes 320ps.

Figure 4.4 shows the data transition regions after the first logic stage. W1, W2, W3, and W4 are the components of the wave, and W shows the total propagating wave. The shaded regions are the transition regions, and the white regions are the stable regions, in which no transition occurs.

Total propagation time of this imaginary combinational logic circuit is 1600ps. Without pipelining, this logic circuit operates with a maximum operating frequency of 625MHz, ignoring the clocking overhead.

If wave-pipelining is used, then Equation (5) dictates the operating frequency. If total clocking overhead \( (T_S + T_H + 2\Delta u) \) is assumed to be 40ps, then \( T_{CLK} > (D_{MAX} - D_{MIN}) + 40\) ps
holds. If the target operating frequency is assumed to be 5GHz (i.e. clock period of 200ps), then, \((D_{\text{MAX}}-D_{\text{MIN}})\) should be smaller than 160ps. This is the critical value (i.e. the tolerable value) of the minimum and maximum delay difference. If classical wave-pipelining or mesochronous pipelining schemes are used, then all of the components of the wave are needed to be sampled after second logic stage, because the delay difference value of the 4\(^{th}\) path reaches to 160ps after 2\(^{nd}\) logic stage. Otherwise, the delay difference value of the 4th path reaches to 240ps after 3\(^{rd}\) logic stage, which exceeds the tolerable value of 160ps. Figure 4.5 shows the case in which mesochronous pipelining scheme is applied to this circuit.

As seen in Figure 4.3, only the 4\(^{th}\) branch reaches to the critical value of 160ps after 2\(^{nd}\) logic stage. Therefore, if WCSM is used, it is enough to sample only the 4\(^{th}\) component of the wave. Considering the 3\(^{rd}\) logic stage, sampling the 3\(^{rd}\) component after 2\(^{nd}\) logic stage is adequate. Otherwise, we will need to sample the 4\(^{th}\) component after 3\(^{rd}\) logic stage. Figure 4.6 shows the logic circuit when WCSM is applied. Instead of 8 flip-flops used in mesochronous or wave-pipelining schemes, only 3 flip-flops are enough for proper operation.

Figure 4.5 Flip-flops inserted using mesochronous pipelining scheme

![Figure 4.5](image-url)
Figure 4.6 Flip-flops inserted using proposed WCSM

Figure 4.7 shows the register insertion algorithm of WCSM. \( T_c \) is the critical value of the delay difference, where \( T_c = T_{CLK} - (T_s + T_H + 2\Delta U) \). Here, \((i,j)\) represents the location of a node, and \( \Delta T(i,j) \) shows the width of the transition region at node\((i,j)\). In calculating \( \Delta T(i,j) \), the transition variation of the logic block at node\((i,j)\) is added with the maximum transition variation of the inputs which enter to the logic block block at node\((i,j)\).
Start

\[ i=0, j=0 \]

Calculate \( \Delta T(i,j) \)

\( \Delta T(i,j) > \Delta T_c ? \)

YES

Insert a register on node \((i,j)\)

NO

\( i = i+1 \)

\( i = n? \)

YES

\( i = 0 \)

\( j = j+1 \)

NO

\( j = m? \)

YES

Finish

Figure 4.7 Flow chart of register insertion of WCSM
4.2 Advantages and Disadvantages of Wave Component Sampling Method

The advantages of the Wave Component Sampling Method can be listed as:

- **Number of synchronizing flip-flops or latches is decreased significantly.**

  Only the paths whose delay difference value reach to the critical level are sampled, the other components continue propagating without being sampled. Therefore, the total number of flip-flops or latches is decreased significantly.

- **Power consumption due to the flip-flops or latches is decreased.**

  Latches and flip-flops consume significant power, especially when a high performance latch or flip-flop is used at high operating frequency. The reduction in the total number of registers also reduces the power consumption due to these unnecessary registers. Delay elements, which are replaced with the registers for aligning the propagating waves, consume lower power than the registers, especially when the operating speed is high. Therefore, total power of the chip is reduced.

- **Clock distribution to the synchronizing flip-flops or latches becomes much easier.**

  Clock distribution is a challenging task, especially in complicated circuits. The distribution of a global clock signal to all of the flip-flops or latches with minimal skew is a very big problem. In WCSM, clock distribution is much easier because of two reasons: First, a global clock signal is not used and instead of it several clock signals which drive a small number of registers are used. Controlling skew between a small number of registers is much easier, therefore, routing of the clock signals with minimal skew becomes easy. Second, the reduction in the number of registers also decreases the total number of clock paths to be routed.

- **Power consumption due to the clock buffers is reduced significantly.**

  The power of the clocking network can be significant, which can be more than half of the total power of the entire chip. A lot of repeaters must be inserted for properly distributing the clock signal throughout the chip. In WCSM, the number of the registers is decreased and several...
clock signals are used to drive small number of registers, therefore the power consumption due to the clock distribution is decreased.

- **High speed operation**

  WCSM provides significant increase in the operating speed compared with the conventional or wave pipelining methods. The speed is also better than mesochronous pipelining method, because considering the layout, the reduction in the number of flip-flops and the clock signals makes placement and routing easier, which increases the operating speed of the chip.

The disadvantages of WCSM can be listed as:

- **The design complexity is increased compared with conventional pipelining.**
  In conventional pipelining, only the paths with worst case delay are considered. However, in WCSM, both the worst case and best case delay values of all the paths must be analyzed, which is similar to mesochronous or wave pipelining methods.

- **The operating frequency cannot be changed afterwards.**
  In WCSM, unnecessary registers are replaced with active delay elements and the components of the waves which are sampled with registers are aligned with the components of the waves, which propagate without being sampled. Therefore, the operating frequency must be initially set and it cannot be changed afterwards. Otherwise, the change in the frequency corrupts the alignment of the propagating waves.

- **WCSM is more susceptible to temperature and process parameter variations.**
  The absolute delay values of the active delay elements are strictly dependent on temperature and process variations. The registers are replaced with active delay elements; therefore the variation of the clock frequency and delay values of the delay elements with respect to temperature and process parameter changes must be handled carefully. Simulations using corner temperature and process parameters must be performed, and all the variations depending on temperature and process parameters must be analyzed.
CHAPTER 5

APPLICATION OF WCSM TO MULTIPLIER STRUCTURES

In order to demonstrate the effectiveness of the WCSM and compare it with the other pipelining methods, 8x8 bit multiplier is implemented using UMC-0.18µm CMOS technology. In [24], carry save adder multiplier structure was used to demonstrate the performance of the mesochronous pipelining scheme, therefore same structure is used for comparing WCSM with mesochronous pipelining method. For achieving high performance multipliers, several optimizations are performed in the implementation of the multiplier blocks.

The application of WCSM to other multiplier structures including booth encoding, Wallace trees and carry look-ahead adders is also investigated. 8x8 bit multiplier using these structures is implemented and the performance comparison with the other pipelining methods is performed.

5.1 Logic selection

The logic gates used in WCSM must have small delay variation at the output. The rise time and fall time of the logic must also be small, in order to have high clock frequency. At the same time, the power consumption of the logic and the latency must also be small.

Output delay of classical CMOS logic is strictly dependent on the input pattern. Figure 5.1 shows a classical CMOS 2 input NAND gate. When both of the inputs are HIGH, then PMOS transistors are OFF and the NMOS transistors pull down the output to LOW. When one of the inputs are LOW and the other input is HIGH, then the pull down path is closed and one PMOS transistor pulls up the output to HIGH. When both of the inputs are HIGH, then there will be two pull-up paths, therefore the output delay becomes much smaller. In
classical CMOS gates with 3 or higher inputs, the delay variation at the output becomes much higher. In [17], CMOS gates with current limiting transistors are proposed. Figure 5.2 shows a 2 input CMOS NAND gate with current limiting PMOS transistor at the top. Pull-up current is limited with PMOS transistor which is always ON, however in this case the rise time becomes longer, which is not suitable for very high speed operations.

A symmetrical circuit structure is important to achieve small delay variation at the output. Figure 5.3 shows the schematic diagram of symmetrical transmission gate logic, which has 4 inputs named as X, NX, Y, and Z, where NX is the complement of X input. The output function of this logic block is $Q = \overline{(X'Y + XZ)}$. This generic block is suitable for WCSM and it is used to implement several logic operations in the various sub blocks of the multiplier.
After the transmission gates, an inverter is used to provide the required drive strength needed for driving the cascading stages. It is also possible to use a cascade of 2 inverters at the output for further improving the drive strength, which is shown in Figure 5.4. The sizes of the transistors of transmission gate logic must be optimized for high speed, and low power operation with minimal delay variation at the output. The ratio between the inverters must also be optimized. Rise and fall time of not only the output but also the internal signals must be low.

The lengths of the transistors are used as minimum size, which is 180nm. The parameters to be optimized can be listed as:

- The width of the PMOS transistors of transmission gates (Wp).
- The width of the NMOS transistors of transmission gates (Wn).
- The width of the PMOS transistor of the first inverter stage (Wp1).
- The width of the NMOS transistor of the first inverter stage (Wn1).
- Assuming that the transistor sizes of the second inverter are a constant times that of the first inverter, the ratio between these inverters.
These parameters are analyzed considering all of the possible transitions at the input. The inputs and the outputs of the logic are connected to the logic gates with the same structure; therefore a simulation setup shown in Figure 5.5 is used. Cadence Analog Design Environment is used and post-layout simulations are performed.

Figure 5.5 Block diagram of the simulation setup

Figure 5.6 shows the simulation of this circuit, where the internal signals of transmission gate in the middle are displayed. In this figure, only a transition in “Y” input occurs. An input pulse with a period of 250ps is applied to the circuit. As it is seen from the figure, the width of the negative pulse is 243ps, and the rise time and fall time at the output of the transmission gates are 132ps and 86ps, respectively. Table 5.1 shows the parameters used in the simulation:
Figure 5.6 Simulation of transmission gate logic block

Table 5-1 Parameters of transistors

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wp</td>
<td>500n</td>
</tr>
<tr>
<td>Wn</td>
<td>240n</td>
</tr>
<tr>
<td>Wp1</td>
<td>600n</td>
</tr>
<tr>
<td>Wn1</td>
<td>240n</td>
</tr>
<tr>
<td>a</td>
<td>1.5</td>
</tr>
</tbody>
</table>

For optimizing the parameter values, parametric sweep analysis is performed. Figure 5.7 shows an example parametric analysis, in which “a” is kept as 1.5 and the sizes of the PMOS transistors are changed from 500n to 900n and the sizes of the NMOS transistors are changed from 180n to 350n. Every parameter takes 5 different values; therefore the total number of simulations is $5^4 = 625$. Figure 5.8 shows parametric sweep of the inverter ratio “a” between 1.5 and 2.3.
Figure 5.7 Parametric sweep analysis of transistor sizes

Figure 5.8 Parametric sweep analysis of inverter ratio
The average current drawn from 1.8V supply using 2 inverter cascades at the output is calculated as 59µA. When a single inverter is used, the average current becomes 19µA. The output delay variation and rise-fall time values are measured to be similar; therefore transmission gate logic with 1 inverter at the output is decided to be used in the multiplier design.

The sizes of the transistors of the inverter must also be optimized. A weak inverter with small transistors will not be able to drive the succeeding logic gates. Then, glitches occur between the transitions. In Figure 5.9, a small glitch in signal C1 is shown. If the sizes of the transistors become too large, than the pass transistors won’t be able to drive the inverter. Therefore the rise-time and fall-time will be high. Table 5.2 shows the sizes of the transistors optimized with parametric analysis.

![Glitch generation between transitions](image-url)
Table 5-2 Optimized transistor parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wp</td>
<td>450n</td>
</tr>
<tr>
<td>Wn</td>
<td>350n</td>
</tr>
<tr>
<td>Wp1</td>
<td>600n</td>
</tr>
<tr>
<td>Wn1</td>
<td>280n</td>
</tr>
</tbody>
</table>

Table 5.3 shows the output propagation times for all transitions in the input. Output delays change between 73ps and 100ps, giving an output delay variation of 27ps. Figure 5.10 shows graphical representation of output delay values.
<table>
<thead>
<tr>
<th>Transition no</th>
<th>ABCin_i</th>
<th>ABCin_f</th>
<th>Output Delay</th>
<th>Transition no</th>
<th>ABCin_i</th>
<th>ABCin_f</th>
<th>Output Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000</td>
<td>001</td>
<td>NC</td>
<td>29</td>
<td>100</td>
<td>000</td>
<td>NC</td>
</tr>
<tr>
<td>2</td>
<td>000</td>
<td>001</td>
<td>73ps</td>
<td>30</td>
<td>100</td>
<td>001</td>
<td>NC</td>
</tr>
<tr>
<td>3</td>
<td>000</td>
<td>011</td>
<td>78ps</td>
<td>31</td>
<td>100</td>
<td>010</td>
<td>83ps</td>
</tr>
<tr>
<td>4</td>
<td>000</td>
<td>100</td>
<td>NC</td>
<td>32</td>
<td>100</td>
<td>011</td>
<td>74ps</td>
</tr>
<tr>
<td>5</td>
<td>000</td>
<td>101</td>
<td>88ps</td>
<td>33</td>
<td>100</td>
<td>101</td>
<td>76ps</td>
</tr>
<tr>
<td>6</td>
<td>000</td>
<td>110</td>
<td>NC</td>
<td>34</td>
<td>100</td>
<td>110</td>
<td>NC</td>
</tr>
<tr>
<td>7</td>
<td>000</td>
<td>111</td>
<td>83ps</td>
<td>35</td>
<td>100</td>
<td>111</td>
<td>78ps</td>
</tr>
<tr>
<td>8</td>
<td>001</td>
<td>000</td>
<td>NC</td>
<td>36</td>
<td>101</td>
<td>000</td>
<td>85ps</td>
</tr>
<tr>
<td>9</td>
<td>001</td>
<td>010</td>
<td>74ps</td>
<td>37</td>
<td>101</td>
<td>001</td>
<td>85ps</td>
</tr>
<tr>
<td>10</td>
<td>001</td>
<td>011</td>
<td>78ps</td>
<td>38</td>
<td>101</td>
<td>010</td>
<td>NC</td>
</tr>
<tr>
<td>11</td>
<td>001</td>
<td>100</td>
<td>NC</td>
<td>39</td>
<td>101</td>
<td>011</td>
<td>NC</td>
</tr>
<tr>
<td>12</td>
<td>001</td>
<td>101</td>
<td>77ps</td>
<td>40</td>
<td>101</td>
<td>100</td>
<td>78ps</td>
</tr>
<tr>
<td>13</td>
<td>001</td>
<td>110</td>
<td>NC</td>
<td>41</td>
<td>101</td>
<td>110</td>
<td>96ps</td>
</tr>
<tr>
<td>14</td>
<td>001</td>
<td>111</td>
<td>75ps</td>
<td>42</td>
<td>101</td>
<td>111</td>
<td>NC</td>
</tr>
<tr>
<td>15</td>
<td>010</td>
<td>000</td>
<td>76ps</td>
<td>43</td>
<td>110</td>
<td>000</td>
<td>NC</td>
</tr>
<tr>
<td>16</td>
<td>010</td>
<td>001</td>
<td>96ps</td>
<td>44</td>
<td>110</td>
<td>001</td>
<td>NC</td>
</tr>
<tr>
<td>17</td>
<td>010</td>
<td>011</td>
<td>NC</td>
<td>45</td>
<td>110</td>
<td>010</td>
<td>73ps</td>
</tr>
<tr>
<td>18</td>
<td>010</td>
<td>100</td>
<td>77ps</td>
<td>46</td>
<td>110</td>
<td>011</td>
<td>79ps</td>
</tr>
<tr>
<td>19</td>
<td>010</td>
<td>101</td>
<td>NC</td>
<td>47</td>
<td>110</td>
<td>100</td>
<td>NC</td>
</tr>
<tr>
<td>20</td>
<td>010</td>
<td>110</td>
<td>83ps</td>
<td>48</td>
<td>110</td>
<td>101</td>
<td>75ps</td>
</tr>
<tr>
<td>21</td>
<td>010</td>
<td>111</td>
<td>NC</td>
<td>49</td>
<td>110</td>
<td>111</td>
<td>76ps</td>
</tr>
<tr>
<td>22</td>
<td>011</td>
<td>000</td>
<td>94ps</td>
<td>50</td>
<td>111</td>
<td>000</td>
<td>84ps</td>
</tr>
<tr>
<td>23</td>
<td>011</td>
<td>001</td>
<td>74ps</td>
<td>51</td>
<td>111</td>
<td>001</td>
<td>100ps</td>
</tr>
<tr>
<td>24</td>
<td>011</td>
<td>010</td>
<td>NC</td>
<td>52</td>
<td>111</td>
<td>010</td>
<td>NC</td>
</tr>
<tr>
<td>25</td>
<td>011</td>
<td>100</td>
<td>85ps</td>
<td>53</td>
<td>111</td>
<td>011</td>
<td>NC</td>
</tr>
<tr>
<td>26</td>
<td>011</td>
<td>101</td>
<td>NC</td>
<td>54</td>
<td>111</td>
<td>100</td>
<td>96ps</td>
</tr>
<tr>
<td>27</td>
<td>011</td>
<td>110</td>
<td>92ps</td>
<td>55</td>
<td>111</td>
<td>101</td>
<td>NC</td>
</tr>
<tr>
<td>28</td>
<td>011</td>
<td>111</td>
<td>NC</td>
<td>56</td>
<td>111</td>
<td>110</td>
<td>76ps</td>
</tr>
</tbody>
</table>
5.2 Delay Balancing

The delay differences between propagating waves must be minimized. A single inverter has a propagation delay of 32ps; therefore cascades of inverters are used for delay balancing between the propagating waves. Delay variation at the output of a single transmission gate logic is measured to be 27ps. When several transmission gates are cascades, the output delay variation increases further. Therefore delay adjustment with a resolution of 32ps is suitable.

An inverter used as a delay element must have small and equal rise and fall times. Otherwise, the width of the incoming pulse decreases or increases at the output of the inverter. Then the propagating pulses may vanish when several inverters are cascaded for providing high delay values. Figure 5.11 shows the transitions of a signal before and after a cascade of 8 inverters. As it is seen from the figure, the shape of the pulse is not distorted while propagating.
When a delay resolution smaller than 32ps is needed, than inverter cascades with different W/L ratios are used, which is shown in Figure 5.12. The ratio of the sizes of the cascading inverters must be carefully designed, in order to obtain the required delay value without any distortion in the propagating signal. Figure 5.13 shows the signals obtained by using 4 inverter cascades with normal inverters and tuned inverters. Delay between them is around 10ps.

Figure 5.12 Inverter cascade with different W/L ratios
Simultaneous generation of complementary outputs

When the complementary and normal signals are generated from different logic blocks, then a delay difference between them occurs. Consider a 2 input AND gate designed with transmission gate logic blocks in Figure 5.14, which produce Q and NQ signals by using separate logic blocks. Table 5.4 shows the corresponding input combinations of transmission gates configured to implement AND gate and NAND gate. Table 5.5 shows the output propagation times of Q and NQ signals, which are taken from the output delay values of generic transmission gate logic in Table 5.3. As it is seen from the table, there is a maximum delay difference of 17ps between Q and NQ signals.

Figure 5.14 Generation of complementary outputs by using separate logic
Table 5-4 Input combinations of generic trans. gate logic for NAND and AND gates

<table>
<thead>
<tr>
<th>ABC inputs of trans. gate implementing NAND</th>
<th>Corresponding ABC inputs of trans. gate implementing AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>011</td>
</tr>
<tr>
<td>010</td>
<td>001</td>
</tr>
<tr>
<td>100</td>
<td>111</td>
</tr>
<tr>
<td>110</td>
<td>101</td>
</tr>
</tbody>
</table>

Complementary and normal output signals are connected to succeeding logic blocks, including gates of the pass transistors. Both of Q and NQ will be at the same state for 17ps, which means that two different signals drive the same net for 17ps. This creates a conflict on the net, and some glitches may occur.

Simultaneous generation of complementary and normal signals with symmetrical transition is very important for decreasing the delay variation of propagating waves. For this purpose, the circuit shown in Figure 5.15 is designed. A transmission gate, which is always “ON” and has same delay with that of the inverter, is used to produce normal and complementary signals simultaneously. Since same logic is used to produce them, there will be no delay variation between Q and NQ due to input pattern. They always make symmetrical transition at the same time instant. Figure 5.16 shows the simulation of the complementary and normal output generation.
### Table 5.5 Output propagation times of Q and NQ signals

<table>
<thead>
<tr>
<th>Output delay of transitions of NAND gate</th>
<th>Output delay of corresponding transitions of AND gate</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 – 010</td>
<td>73ps</td>
</tr>
<tr>
<td>000 – 100</td>
<td>011 – 001</td>
</tr>
<tr>
<td>000 – 110</td>
<td>NC</td>
</tr>
<tr>
<td>010 – 000</td>
<td>011 – 101</td>
</tr>
<tr>
<td>010 – 100</td>
<td>76ps</td>
</tr>
<tr>
<td>010 – 110</td>
<td>001 – 101</td>
</tr>
<tr>
<td>010 – 100</td>
<td>77ps</td>
</tr>
<tr>
<td>010 – 110</td>
<td>001 – 101</td>
</tr>
<tr>
<td>100 – 000</td>
<td>NC</td>
</tr>
<tr>
<td>100 – 100</td>
<td>111 – 011</td>
</tr>
<tr>
<td>100 – 110</td>
<td>83ps</td>
</tr>
<tr>
<td>100 – 110</td>
<td>111 – 101</td>
</tr>
<tr>
<td>110 – 000</td>
<td>NC</td>
</tr>
<tr>
<td>110 – 010</td>
<td>101 – 011</td>
</tr>
<tr>
<td>110 – 100</td>
<td>73ps</td>
</tr>
<tr>
<td>110 – 100</td>
<td>101 – 001</td>
</tr>
</tbody>
</table>

Figure 5.15 Simultaneous generation of complementary and normal output signals
There is another important advantage of using this structure for producing complementary and normal outputs: Since a separate logic is not used, an input signal is connected to only one logic gate, instead of two. This makes the drive strength required to drive the logic to be half of using separate logic, which decreases the area and power consumption significantly.

### 5.4 Implementation of Multiplier Blocks

#### 5.4.1 Half adder design

Half adder block has 2 inputs, and produces Sum and Carry-out. Table 5.6 shows the truth table of half adder.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Sum = A \bmod B = A'B + AB'

Cout = A & B.
Generic transmission gate logic performs logical function of 
\[ Q = \overline{(AC + A'B)} = (A' + C')*(A + B') \]. For obtaining “Sum” using the generic transmission gate logic, the inputs of Half adder are connected to the inputs of transmission gate as:

\[ X = B, \ NX = B', \ Y = A', \ Z = A \]

Then, Sum output becomes:

\[ \text{Sum} = (B' + A')(B + A) = B'B + B'A + A'B + A'A = A'B + AB' \]

For Cout, inputs of half adder are connected to the inputs of transmission gate logic as:

\[ X = A, \ NX = A', \ Y = 1, \ Z = B' \]

Then, Cout becomes:

\[ \text{Cout} = (A' + B)(A + 0) = A'A + A'0 + AB + B0 = AB \]

Each input of half adder is connected to 2 transmission gate logic blocks. For increasing the drive strength of the inputs, they are passed through buffers. Figure 5.17 shows the block diagram of the half adder including the input buffers.
Figure 5.17 Schematic diagram of half adder block

Figure 5.18 shows an example simulation of half adder block. Table 5.7 shows the output propagation times for all input transitions. Figure 5.19 shows same values in a graphical representation.
Figure 5.18 A simulation example of half adder

Table 5-7 Output delay values of half adder for all input transitions

<table>
<thead>
<tr>
<th>AB_initial</th>
<th>AB_final</th>
<th>Sum</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
<td>194ps</td>
<td>NC</td>
</tr>
<tr>
<td>00</td>
<td>10</td>
<td>210ps</td>
<td>NC</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>NC</td>
<td>215ps</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>209ps</td>
<td>NC</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
<td>214ps</td>
<td>215ps</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>212ps</td>
<td>NC</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>221ps</td>
<td>216ps</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>NC</td>
<td>198ps</td>
</tr>
<tr>
<td>11</td>
<td>01</td>
<td>215ps</td>
<td>215ps</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>217ps</td>
<td>217ps</td>
</tr>
</tbody>
</table>
Figure 5.19 Graphical representation of the output delay values of half adder.

Figure 5.20 shows the layout of the half adder block.
5.4.2 Full adder design

Full adder has 3 inputs named as A, B, and Carry input (Cin). Table 5.8 shows the truth table of full adder.

Table 5-8 The truth table of full adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>Sum</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Sum=\(A \text{xor} B \text{xor} Cin = A'B'Cin + A'BCin' + AB'Cin + ABCin\)
Cout=\(AB + ACin + BCin + ABCin\)

Sum and Cout are generated using two stages in full adder implementation. An intermediate signal “P” (propagate) is generated in the first stage, where P=\(A \text{xor} B = A'B + AB'\), therefore connection of inputs to generic transmission gate block is similar to the half adder:

\[X = B, \; NX = B', \; Y = A', \; Z = A,\]

\[P = \neg(AB + A'B') = (A' + B')(A + B) = A'B + AB'\]

P signal is used in the generation of both the Sum and Cout, where

Sum=\(P \text{xor} Cin\)
Cout=\(\neg(PCin' + P'B')\).

Fan out of P and NP signals are 2, however they are connected to the gates of the transistors. Driving the gate of a transmission gate is easier than driving source or gate. Therefore,
driving capacities of P and NP signals are enough for driving the gates of double transmission gate logic. Figure 5.21 shows the transistor level diagram of full adder. “Cin” and “NCin” inputs are delayed using a cascade of 5 inverters. "P" and “NP” signals are used with the delayed version of "carry" input and “B” input to produce the outputs “Sum” and “Cout”.

Figure 5.21 Schematic diagram of full adder block

Table 5.9 shows the output delay values of full adder block. The minimum delay is 340ps and the maximum delay is 402ps. Figure 5.22 shows the graphical representation of the output delay values.
Table 5-9 Output delay values of full adder

<table>
<thead>
<tr>
<th>Trans. no</th>
<th>ABCin initial</th>
<th>ABCin final</th>
<th>Delay of Sum</th>
<th>Delay of Cout</th>
<th>Trans. no</th>
<th>ABCin initial</th>
<th>ABCin final</th>
<th>Delay of Sum</th>
<th>Delay of Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000</td>
<td>001</td>
<td>366ps</td>
<td>NC</td>
<td>29</td>
<td>100</td>
<td>000</td>
<td>390ps</td>
<td>NC</td>
</tr>
<tr>
<td>2</td>
<td>000</td>
<td>010</td>
<td>385ps</td>
<td>NC</td>
<td>30</td>
<td>100</td>
<td>001</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>3</td>
<td>000</td>
<td>011</td>
<td>NC</td>
<td>380ps</td>
<td>31</td>
<td>100</td>
<td>010</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>4</td>
<td>000</td>
<td>100</td>
<td>378ps</td>
<td>NC</td>
<td>32</td>
<td>100</td>
<td>011</td>
<td>349ps</td>
<td>364ps</td>
</tr>
<tr>
<td>5</td>
<td>000</td>
<td>101</td>
<td>NC</td>
<td>357ps</td>
<td>33</td>
<td>100</td>
<td>101</td>
<td>353ps</td>
<td>367ps</td>
</tr>
<tr>
<td>6</td>
<td>000</td>
<td>110</td>
<td>NC</td>
<td>353ps</td>
<td>34</td>
<td>100</td>
<td>110</td>
<td>376ps</td>
<td>370ps</td>
</tr>
<tr>
<td>7</td>
<td>000</td>
<td>111</td>
<td>367ps</td>
<td>364ps</td>
<td>35</td>
<td>100</td>
<td>111</td>
<td>NC</td>
<td>387ps</td>
</tr>
<tr>
<td>8</td>
<td>001</td>
<td>000</td>
<td>365ps</td>
<td>NC</td>
<td>36</td>
<td>101</td>
<td>000</td>
<td>NC</td>
<td>374ps</td>
</tr>
<tr>
<td>9</td>
<td>001</td>
<td>010</td>
<td>NC</td>
<td>NC</td>
<td>37</td>
<td>101</td>
<td>001</td>
<td>376ps</td>
<td>390ps</td>
</tr>
<tr>
<td>10</td>
<td>001</td>
<td>011</td>
<td>394ps</td>
<td>368ps</td>
<td>38</td>
<td>101</td>
<td>010</td>
<td>340ps</td>
<td>348ps</td>
</tr>
<tr>
<td>11</td>
<td>001</td>
<td>100</td>
<td>NC</td>
<td>NC</td>
<td>39</td>
<td>101</td>
<td>011</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>12</td>
<td>001</td>
<td>101</td>
<td>387ps</td>
<td>374ps</td>
<td>40</td>
<td>101</td>
<td>100</td>
<td>354ps</td>
<td>363ps</td>
</tr>
<tr>
<td>13</td>
<td>001</td>
<td>110</td>
<td>364ps</td>
<td>363ps</td>
<td>41</td>
<td>101</td>
<td>110</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>14</td>
<td>001</td>
<td>111</td>
<td>NC</td>
<td>354ps</td>
<td>42</td>
<td>101</td>
<td>111</td>
<td>367ps</td>
<td>NC</td>
</tr>
<tr>
<td>15</td>
<td>010</td>
<td>000</td>
<td>387ps</td>
<td>NC</td>
<td>43</td>
<td>110</td>
<td>000</td>
<td>NC</td>
<td>358ps</td>
</tr>
<tr>
<td>16</td>
<td>010</td>
<td>001</td>
<td>NC</td>
<td>NC</td>
<td>44</td>
<td>110</td>
<td>001</td>
<td>365ps</td>
<td>378ps</td>
</tr>
<tr>
<td>17</td>
<td>010</td>
<td>011</td>
<td>353ps</td>
<td>367ps</td>
<td>45</td>
<td>110</td>
<td>010</td>
<td>376ps</td>
<td>391ps</td>
</tr>
<tr>
<td>18</td>
<td>010</td>
<td>100</td>
<td>NC</td>
<td>NC</td>
<td>46</td>
<td>110</td>
<td>011</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>19</td>
<td>010</td>
<td>101</td>
<td>354ps</td>
<td>366ps</td>
<td>47</td>
<td>110</td>
<td>100</td>
<td>372ps</td>
<td>378ps</td>
</tr>
<tr>
<td>20</td>
<td>010</td>
<td>110</td>
<td>393ps</td>
<td>381ps</td>
<td>48</td>
<td>110</td>
<td>101</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>21</td>
<td>010</td>
<td>111</td>
<td>NC</td>
<td>367ps</td>
<td>49</td>
<td>110</td>
<td>111</td>
<td>366ps</td>
<td>NC</td>
</tr>
<tr>
<td>22</td>
<td>011</td>
<td>000</td>
<td>NC</td>
<td>378ps</td>
<td>50</td>
<td>111</td>
<td>000</td>
<td>363ps</td>
<td>374ps</td>
</tr>
<tr>
<td>23</td>
<td>011</td>
<td>001</td>
<td>370ps</td>
<td>390ps</td>
<td>51</td>
<td>111</td>
<td>001</td>
<td>NC</td>
<td>357ps</td>
</tr>
<tr>
<td>24</td>
<td>011</td>
<td>010</td>
<td>354ps</td>
<td>364ps</td>
<td>52</td>
<td>111</td>
<td>010</td>
<td>NC</td>
<td>402ps</td>
</tr>
<tr>
<td>25</td>
<td>011</td>
<td>100</td>
<td>353ps</td>
<td>363ps</td>
<td>53</td>
<td>111</td>
<td>011</td>
<td>393ps</td>
<td>NC</td>
</tr>
<tr>
<td>26</td>
<td>011</td>
<td>101</td>
<td>NC</td>
<td>NC</td>
<td>54</td>
<td>111</td>
<td>100</td>
<td>NC</td>
<td>401ps</td>
</tr>
<tr>
<td>27</td>
<td>011</td>
<td>110</td>
<td>NC</td>
<td>NC</td>
<td>55</td>
<td>111</td>
<td>101</td>
<td>389ps</td>
<td>NC</td>
</tr>
<tr>
<td>28</td>
<td>011</td>
<td>111</td>
<td>380ps</td>
<td>NC</td>
<td>56</td>
<td>111</td>
<td>110</td>
<td>365ps</td>
<td>NC</td>
</tr>
</tbody>
</table>
Figure 5.22 Graphical representation of output delay values of full adder

Figure 5.23 shows the layout diagram of full adder.

Figure 5.23 Layout diagram of the full adder block

### 5.4.3 Partial Product Generation

Partial products are generated by using AND operation of a single bit of the multiplier with all the bits of the multiplicand. AND gate is also designed by using generic transmission gate logic. The inputs of the AND block is connected to the inputs of transmission gate as:

\[ X = B; \ NX = B'; \ Y = 1; \ Z = A' \]

\[ Q = \neg(BA' + B'1) = (B' + A)(B + 0) = AB \]
Schematic diagram of AND gate is shown in Figure 5.24.

![Schematic diagram of AND gate](image)

**Figure 5.24 Schematic diagram of AND gate**

The inputs of X, NX and Z comes from the outputs of inverters. For balancing the drive strengths of Y input with those inputs, Y input is not connected directly to VDD. Rather, it is connected by using an inverter, whose input is tied to GND. Since NMOS transistor is always OFF, it is omitted and only PMOS transistor is used.

A partial product generator combines one bit of multiplier with all the bits of the multiplicand. In an 8x8 bit multiplier, a fan out of 8 is needed to produce partial products. For driving 8 inputs, the drive strength of the single bit of multiplier is increased by a buffer composed of a cascade of 3 inverters. Buffered signal is used to drive the gates of transistors in the transmission gate logic, since driving the gates require less drive capability than driving drain or source of the transistors.

8 bits of the multiplicand are delayed by using cascades of 4 inverters for equalizing the delay differences. Figure 5.25 and Figure 5.26 show the block diagram and simulation of the partial product generator, respectively. Layout diagram of the partial product generator is shown in Figure 5.27.
Figure 5.25 Block diagram of partial product generator
Figure 5.26 Simulation of partial product generator
5.4.4 Sampling of the signals

The width of the waves decreases while propagating through the logic, and they must be sampled before their aperture becomes too small which creates setup and hold time violations. Sampling resets the propagating waves, and they start propagating with equalized delays. It is important to use a register, which is capable of sampling narrow pulses.

The setup time is the minimum amount of time the data signal should be held steady before the clock event so that the data are reliably sampled by the clock. Hold time is the minimum amount of time the data signal should be held steady after the clock event so that the data are reliably sampled. Setup and hold time must be small, in order to sample narrow pulses. There are many flip-flop and latch structures. Mesochronous multiplier in [24] uses Sense Amplifier Based Flip-Flop (SAFF). The setup and hold time of SAFF is 10ps and 130ps, respectively. The clock High time of the SAFF is 160ps, and considering a clock signal with 50% duty cycle, the minimum clock period is 320ps. A margin of 30ps is used, so that operating frequency of SAFF becomes 2.86GHz.
In this implementation, C2MOS latches are used for sampling because of their high speed operation, which is seen in Figure 5.28. By using iterations and parametric analysis, the sizes of the transistors are optimized for sampling narrow pulses with minimal delay variation at the output. Table 5.10 shows the sizes of the transistors of the latch, which is capable of sampling a pulse with a width of 120ps. The setup time of the latch is 0. Latching occurs in the positive clock cycle; therefore the signals must be stable in this region. A clock signal with a frequency of 5GHz is used in the simulations of the C2MOS latch. Figure 5.29 shows the transitions of the internal signals of the latch. In Figure 5.30, several propagating waves with an aperture of 120ps and clock signal are shown before sampling. Figure 5.31 shows the same signals after sampling.

![Schematic diagram of C2MOS latch](image)

**Figure 5.28 Schematic diagram of C2MOS latch**

<table>
<thead>
<tr>
<th>Transistor</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wp1</td>
<td>1.6um</td>
</tr>
<tr>
<td>Wp2</td>
<td>1.6um</td>
</tr>
<tr>
<td>Wn1</td>
<td>480nm</td>
</tr>
<tr>
<td>Wn2</td>
<td>480nm</td>
</tr>
</tbody>
</table>

Table 5-10 Transistor sizes of C2MOS latch.
Figure 5.29 Simulations of internal signals of the latch

Figure 5.30 Propagating waves before sampling
Input to output delay of C2MOS depends slightly on the shape and position of the input. Therefore, a delay difference of 10ps occurs after sampling.

5.5 Implementation of 8x8 bit CSA multiplier

In order to demonstrate the effectiveness of WCSM and compare it with the mesochronous pipelining scheme, two 8x8-bit multipliers using both the mesochronous pipelining scheme and WCSM are implemented. First, the mesochronous multiplier using the same structure of the multiplier in [24] is implemented. Then WCSM is applied to this multiplier. Internal logic blocks described before are used in the multipliers.

5.5.1 Schematic design

Figure 5.32 shows the block diagram of the multiplier, where full-adders are shown with “F”, half adders are shown with “H”, and the registers are shown with “R”.

Full adder has two levels of transmission gate logic, while half adder and partial product generator have one level of logic. Every logic level increases the delay difference between signals; therefore at most 6 levels of logic stages are used between registers, which is seen in Figure 5.32.
The first register stage is used at the input. The second register stage is used after a cascade of a partial product generator, a half adder, and 2 full adders, which takes 6 levels of logic stages. The third register stage is used after a cascade of 3 full adders. The fourth register stage is used after a full adder plus a cascade of 3 half adders. The fifth register stage, which is also the output stage, is used after a cascade of 4 half adders plus an “OR” gate.

As it is seen from Figure 5.32, there are some paths which do not contain 6 levels of logic operation. The paths in the upper triangular region are composed of only buffer cascades. Also, the paths in the partial product generation region are composed of an “AND” gate and buffer cascades. Therefore, the delay differences of those paths do not reach to critical delay difference value. When WCSM is applied to this multiplier, the latches which are circled with dashed lines are eliminated and they are replaced with delay elements. Figure 5.33 shows 8x8 bit multiplier implemented using WCSM.
In mesochronous multiplier, 110 registers are used including input and output registers. However, in WCSM multiplier only 70 registers are used, without any performance loss in the operating frequency. The total number of the registers in the multiplier implemented with WCSM is 41% lower than the multiplier implemented with mesochronous pipelining method. The reduction of the registers also decreases the transistor sizes of the associated clock buffers, which significantly reduces the power consumption.

5.5.2 Operating Frequency

In the multiplier, at most 6 transmission gate logic cells are cascaded between two successive register stages. Each transmission gate logic has a delay difference of 25ps at the output, when the inputs are applied simultaneously. When they are cascaded, the delay differences of the logic cells are accumulated. Therefore, the total delay difference, which is also the width of the transition region, becomes 150ps after 6 logic levels. C2MOS latch samples the data at positive cycle of the clock signal; therefore the transition of the signals must occur at negative cycle. This condition limits the minimum width of the negative clock cycle to be equal to the width of the transition region, which is 150ps. If a clock signal with a duty cycle
of 50% is used, then the minimum clock period becomes 300ps. Considering the additional delay difference coming from the latch itself and to have some margin, it is proper to use a clock signal with a period of 330ps (an operating frequency of 3GHz).

5.5.3 Layout Implementation of the multipliers

The layouts of the multipliers are implemented using UMC 0.18um technology with Cadence design tools. All the logic blocks, clock buffers and the latches are drawn by using full custom design methodology. 1 poly and 3 metal layers are used in the design. The placement and routing is performed carefully to minimize the delay difference between signals.

The height of the standard logic cell is 15.8µm. Figure 5.34 shows the layout of the multiplier implemented using mesochronous pipelining method. The area of both of the multipliers is 0.175mm².
5.5.4 Simulations of multipliers

Random input pattern at a frequency of 3GHz is applied to the multipliers. Figure 5.35 shows the propagating waves after the first partial product generation layer. The width of the propagating waves decreases to 260ps after partial product generation. Figure 5.36 shows the propagating waves after the first half adder layer. The width of the propagating waves become 235ps. Figure 5.37 and Figure 5.38 shows the propagating waves after the first full adder layer and the second full adder layer, respectively. The widths of the propagating waves become 185ps after first full adder layer and 130ps after second full adder layer.
Figure 5.39 shows the propagating waves at the output of the registers. The width of the propagating waves increases to 270ps after sampling.

Figure 5.35 Propagating waves after first partial product generator

Figure 5.36 Propagating waves after the first half adder layer
Figure 5.37 Propagating waves after the first full adder layer

Figure 5.38 Propagating waves after the second full adder layer
5.5.5 Performance comparison of the multipliers

The replacement of unnecessary latches with the delay elements does not affect the remaining circuit; therefore both of the multipliers have the same operating frequency of 3 GHz.

The number of the latches is reduced by 41%, which decreases the power consumption due to the latches. Since the number of latches is decreased, the transistor sizes of the clock buffers driving the latches are reduced in accordance with the reduction of the latches. This significantly reduces the power consumption due to the clock buffers. The delay elements replaced with the latches consume additional power; however it is lower than the power consumed by the latches. Table 5.11 shows the comparison between the multiplier with mesochronous pipelining scheme and the multiplier with WCSM. Total power of the multiplier is decreased by 9.5%.
### Table 5-11 Comparison between mesochronous and WCSM multiplier

<table>
<thead>
<tr>
<th></th>
<th>Mesochronous multiplier</th>
<th>WCSM multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of registers</td>
<td>110</td>
<td>70</td>
</tr>
<tr>
<td>Power of half adders</td>
<td>6.43mW</td>
<td>6.29mW</td>
</tr>
<tr>
<td>Power of full adders</td>
<td>20.63mW</td>
<td>20.94mW</td>
</tr>
<tr>
<td>Power of partial product generators</td>
<td>15.95mW</td>
<td>15.72mW</td>
</tr>
<tr>
<td>Power of clock buffers and latches</td>
<td>25.29mW</td>
<td>14.83mW</td>
</tr>
<tr>
<td>Power of delay elements</td>
<td>25.87mW</td>
<td>27.42mW</td>
</tr>
<tr>
<td>Total power</td>
<td>94.17mW</td>
<td>85.20mW</td>
</tr>
<tr>
<td>Percentage of power of clocking circuits</td>
<td>21.5%</td>
<td>17.4%</td>
</tr>
<tr>
<td>Reduction in total power</td>
<td>-</td>
<td>9.5%</td>
</tr>
</tbody>
</table>

#### 5.6 A 5GHz WCSM-Multiplier

The power consumption of registers and clock buffers increases when the number of pipeline stages is increased. For demonstrating the effectiveness of WCSM in fully pipelined structures, an ultra high speed multiplier is designed using carry save structure. Figure 5.40 shows the block diagram of the full pipelined multiplier. After all logic operations, the propagating waves are sampled. Therefore, there are totally 17 clock signals, which all have a frequency of 5GHz but different phases, in the multiplier. Since each clock signal drives at most 16 gates, the sizes of the clock buffers are much smaller than that of a single clock buffer.
When WCSM is applied to this circuit all the unnecessary registers, which are used after delay elements, are omitted. The signals of these paths are aligned with the sampled signals by using cascades of inverters, which decreases the total number of latches by 45% compared with the mesochronous pipelined multiplier. The sizes of the corresponding clock buffers are also decreased.

For comparing 5GHz WCSM multiplier with the mesochronous pipelined multiplier, both of the circuits are implemented using UMC 0.18um CMOS technology. Full custom design methodology with Cadence design tools is used in the design of all the logic blocks and latches. Post layout simulations are performed using Analog Design Environment of Cadence. Figure 5.41 shows the propagating waves at a frequency of 5 GHz before sampling.
Maximum power consumptions of the multipliers are measured by alternatively applying 00x00 and FFxF to the inputs, in which maximum number of transitions occurs in the multiplier circuit. Table 5.12 shows the maximum power comparison of the proposed multiplier with the mesochronous pipelined multiplier. The total power consumption of mesochronous pipelined multiplier is 113.5mA, and 65mA of it comes from clock buffers and latches. In WCSM multiplier, the power consumption is slightly increased due to delay elements. However, the power consumption of clock buffers and latches decreases significantly, therefore the total power consumption is decreased by 13.7% without any performance loss in the operating speed of the circuit.
Table 5-12 Comparison between 5GHz mesochronous and WCSM multipliers

<table>
<thead>
<tr>
<th>Part</th>
<th>Wave-pipelined multiplier</th>
<th>Proposed multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half adders</td>
<td>6.12mW</td>
<td>6.59mW</td>
</tr>
<tr>
<td>Full adders</td>
<td>13.14mW</td>
<td>13.90mW</td>
</tr>
<tr>
<td>Partial product generators</td>
<td>19.44mW</td>
<td>27.18mW</td>
</tr>
<tr>
<td>Clock buffers and latches</td>
<td>117.0mW</td>
<td>69.12mW</td>
</tr>
<tr>
<td>Delay elements</td>
<td>48.60mW</td>
<td>59.70mW</td>
</tr>
<tr>
<td>Total power</td>
<td>204.3mW</td>
<td>176.5mW</td>
</tr>
<tr>
<td>Power percentage of clocking circuits</td>
<td>57.2%</td>
<td>39.2%</td>
</tr>
</tbody>
</table>

For measuring the average power with random inputs, same random input pattern is applied to both multipliers. Table 5.13 shows the comparison of the power consumption with random inputs. The rate of the transitions is lower when random input pattern is used; therefore the power of logic elements and latches decreases. However, the power of clock buffers remains almost the same, therefore the percentage of total power of clocking circuits in the multiplier is increased to 62.5%, and in WCSM multiplier it is only 42.1%. Overall power of the WCSM multiplier is also 18.4% lower than that of the mesochronous-pipelined multiplier.
Table 5-13 Comparison of the multipliers with random inputs

<table>
<thead>
<tr>
<th></th>
<th>Wave-pipelined multiplier</th>
<th>Proposed multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half adders</td>
<td>3.92mW</td>
<td>3.85mW</td>
</tr>
<tr>
<td>Full adders</td>
<td>9.09mW</td>
<td>9.02mW</td>
</tr>
<tr>
<td>Partial product generators</td>
<td>12.80mW</td>
<td>18.85mW</td>
</tr>
<tr>
<td>Clock buffers and latches</td>
<td>100.85mW</td>
<td>55.44mW</td>
</tr>
<tr>
<td>Delay elements</td>
<td>34.79mW</td>
<td>44.42mW</td>
</tr>
<tr>
<td>Total power</td>
<td>161.45mW</td>
<td>131.58mW</td>
</tr>
<tr>
<td>Power percentage of clocking circuits</td>
<td>62.4%</td>
<td>42.1%</td>
</tr>
</tbody>
</table>

5.7 IMPLEMENTATION OF THE OTHER MULTIPLIER STRUCTURES

5.7.1 Booth encoder design

Modified Booth-2 algorithm is implemented using UMC-0.18µm CMOS technology. Figure 5.42 shows the block diagram of the mux_6x1 logic which selects inputs from the set of {0, +M, +2M, -2M, -M, 1}. Both the normal and complementary outputs are generated. Figure 5.43 shows the MUX module containing 8 mux_6x1 logic blocks. Figure 5.44 shows the top level schematic of the booth-2 module. There are 5 MUX modules, each of them containing 8 internal MUX modules. There are also 16 buffers which drive the inputs of the MUX’es. 4 of the MUX modules select one of 5 inputs, and the last MUX selects one of 2 inputs, which is either 0 or M.
Figure 5.42 Design of Mux6x1 using generic transmission gate logic
Figure 5.43 Block diagram of 8xmux module
When mesochronous pipelining or conventional pipelining method is used, then there will be 10 latches in mux_6x1 block including latches of S1_d and S2_d2 signals. There are 8x5=40 mux_6x1 block in the booth-2 module, therefore total number of latches is 400. If WCSM is applied, then there is no need to sample S1 and S2 signals, and 7 latches are enough in mux_6x1 module. This decreases total number of latches by 30%, which is 280.

### 5.7.2 Wallace tree design

Table 5.14 shows the constructed Wallace tree, in which 3 levels of logic is used. In this table, following abbreviation is used:

- **X**: Partial product bit
- **H**: Half adder (X+X)
- **H’**: Modified half adder (X+X+1)
- **F**: Full adder (X+X+X)
- **G**: Gates (X+1)
- **D**: Direct transfer.
Table 5-14 Wallace tree construction of Modified Booth-2 with 3 levels

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>S0</th>
<th>S0</th>
<th>S0</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>S1</td>
<td>S0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>S0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>S2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>S1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>S2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>S3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>H</th>
<th>H'</th>
<th>F</th>
<th>F,D</th>
<th>F,H</th>
<th>F,H</th>
<th>F,D</th>
<th>F,H</th>
<th>F</th>
<th>F,D</th>
<th>H</th>
<th>F</th>
<th>D</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>F,D</td>
<td>F,D</td>
<td>F,D</td>
<td>F,D</td>
<td>F,D</td>
<td>F</td>
<td>F,H</td>
<td>H</td>
<td>F</td>
<td>D</td>
<td>H</td>
<td>D</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H'</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>H</td>
<td>F</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Figure 5.45 shows the block diagram of the Wallace tree implementation. Full adder blocks have 2 stages, and half adder blocks have 1 stage. If conventional pipelining or mesochronous pipelining with latches after all logic stages are used in Wallace tree, then every half adder and delay element must be sampled two times, for equalizing the path with the full adder. In this case, there will be 239 latches in the conventional pipelined implementation. If WCSM is applied, the total number of latches becomes 117, which constitutes a reduction of 51%. The comparison between conventional or mesochronous pipelining and WCSM in terms of number of registers is shown in Table 5.15.
Figure 5.45 Block diagram of wallace tree with modified Booth-2 algorithm

Table 5-15 Comparison of Wallace trees with mesochronous pipelining and WCSM

<table>
<thead>
<tr>
<th></th>
<th>Conventional or mesochronous pipelining methods</th>
<th>Wave Component Sampling Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latches used in half adders</td>
<td>24x2x2=96</td>
<td>24x2x1=48</td>
</tr>
<tr>
<td>Latches used in full adders</td>
<td>23x5=115</td>
<td>23x3=69</td>
</tr>
<tr>
<td>Latches used in delay elements</td>
<td>14x2=28</td>
<td>-</td>
</tr>
<tr>
<td>Total number of latches</td>
<td>239</td>
<td>117</td>
</tr>
<tr>
<td>Reduction in latches</td>
<td>-</td>
<td>51%</td>
</tr>
</tbody>
</table>
5.7.3 Carry Lookahead adder design

4 bit carry look-ahead adder circuit is designed using transmission gate logic with two inputs. Figure 5.46 shows the first carry look-ahead adder circuit design. It can be seen from the figure that, the generation of the group propagate (PG) and group generate (GG) signals are completed in 3 and 5 levels of logic stages, respectively. The generation of carry output (C4) is completed in 6 levels of logic stages, which is quite high.

![Figure 5.46 Schematic design of 4-bit carry look-ahead adder circuit](image)

The carry signal is produced using the formula $C_{k+1} = P_k C_k + G_k$. The logic circuit of this carry generation formula using 2 input logic gates can be seen in Figure 5.47. It can be seen from the figure that the carry output is generated 2 logic stages after the arrival of the carry input. In order to speed up this operation, carry generation formula is modified to be compatible with the transmission gate logic, such that:

$$C_{k+1} = P_k C_k + G_k(C_k + !C_k) = C_k(P_k + G_k) + !C_k G_k$$

![Figure 5.47 Carry generation logic](image)
Figure 5.48 shows the modified carry generation using this formula, from which it can be seen that the carry is generated using only one level of logic.

![Modified carry generation circuit](image)

Figure 5.48 Modified carry generation circuit

Figure 5.49 shows the carry generation at the first stage of 4 bits. Since there is no incoming carry signal in the first stage, no need to produce group propagate or group generate signals. Therefore, all the logic related to GG and PG is omitted. Also there is no need to produce P0 signal. Therefore, the carry signal is produced using 5 levels of logic in the first stage.

![Carry generation circuit of first stage with 5 levels of logic](image)

Figure 5.49 Carry generation circuit of first stage with 5 levels of logic

In the generation of P signal, both “EXOR” and OR operations can be used. “EXOR” operation is preferred, because “A XOR B” is needed while obtaining Sum output. However, when OR gate is used in the generation of P1, then P1+G1=A1+B1+A1B1=A1+B1=P1. Therefore, OR gate which produces P1+G1 can be neglected for decreasing the latency of carry generation of the first stage by 1. The carry output at the first stage can be completed using 4 levels of logic instead of 5, which is shown in Figure 5.50.
Figure 5.50 Carry generation circuit of first stage with 4 levels of logic

Figure 5.51 shows the carry generation of 16 bit adder circuit. In the first group of 4 bits, no carry look-ahead logic is used, while in the 2nd and the 3rd groups, carry outputs are generated 1 logic level after the arrival of the incoming carry signal.

In the design of 8x8 bit multiplier, the constructed Wallace tree produces 2 outputs of 13 bits and 16 bits, respectively. Therefore, 13 bit carry look-ahead adder is suitable for this particular application. Figure 5.52 shows the block diagram of 13 bit carry look-ahead adder, where carry look-ahead logic is only used in the 2nd group of 4 bits.
Figure 5.51 16 bit adder with carry look-ahead logic
Figure 5.52 13 bit adder with carry look-ahead logic.

For comparing the areas of the carry look-ahead adder and the carry save adder, 16 bit carry save adder tree is designed using Half adder blocks. Figure 5.53 shows the schematic design of the 16 bit carry save adder tree. It can be seen from the figure that, the addition takes 16 levels of logic stage, where each stage is composed of Half adders.
Figure 5.53 16 bit carry save adder tree.

Table 5.16, 5.17 and 5.18 show the number of sub blocks and total number of transistors used in the 16-bit carry save adder, 16 bit carry look-ahead adder and 13 bit carry look-ahead adder, respectively. Comparing Table 5.16 with Table 5.17, it can be seen that carry look-ahead adder implementation utilizes 35% less transistors than the 16 bit carry save adder implementation.

<table>
<thead>
<tr>
<th>Block name</th>
<th>Number of instantiation</th>
<th>Number of trans. gate logic in block</th>
<th>Total number of trans. gate logic</th>
<th>Number of transistors used</th>
</tr>
</thead>
<tbody>
<tr>
<td>HA</td>
<td>136</td>
<td>4</td>
<td>524</td>
<td>3144</td>
</tr>
<tr>
<td>Delay</td>
<td>120</td>
<td>1</td>
<td>120</td>
<td>720</td>
</tr>
<tr>
<td>Latch</td>
<td>80</td>
<td>3</td>
<td>240</td>
<td>1440</td>
</tr>
<tr>
<td>Total number of transistors without latches</td>
<td></td>
<td></td>
<td></td>
<td>3864</td>
</tr>
</tbody>
</table>

Table 5-16 Transistor count of 16-bit carry save adder implementation
Table 5-17 Transistor count of 16-bit carry lookahead adder implementation.

<table>
<thead>
<tr>
<th>Block name</th>
<th>Number of instantiation</th>
<th>Number of pass logic in block</th>
<th>Total number of pass logic</th>
<th>Number of transistors used</th>
</tr>
</thead>
<tbody>
<tr>
<td>PG</td>
<td>15</td>
<td>4</td>
<td>60</td>
<td>360</td>
</tr>
<tr>
<td>XOR</td>
<td>16</td>
<td>2</td>
<td>32</td>
<td>192</td>
</tr>
<tr>
<td>AND2</td>
<td>14</td>
<td>2</td>
<td>28</td>
<td>168</td>
</tr>
<tr>
<td>OR2</td>
<td>19</td>
<td>2</td>
<td>38</td>
<td>228</td>
</tr>
<tr>
<td>MUX</td>
<td>14</td>
<td>2</td>
<td>28</td>
<td>168</td>
</tr>
<tr>
<td>Delay</td>
<td>230</td>
<td>1</td>
<td>230</td>
<td>1380</td>
</tr>
<tr>
<td>Total number of transistors</td>
<td></td>
<td></td>
<td></td>
<td>2496</td>
</tr>
</tbody>
</table>

Table 5-18 Transistor count of 13-bit carry lookahead adder implementation.

<table>
<thead>
<tr>
<th>Block name</th>
<th>Number of instantiation</th>
<th>Number of trans. gate logic in block</th>
<th>Total number of trans. gate logic</th>
<th>Number of transistors used</th>
</tr>
</thead>
<tbody>
<tr>
<td>PG</td>
<td>12</td>
<td>4</td>
<td>48</td>
<td>288</td>
</tr>
<tr>
<td>XOR</td>
<td>13</td>
<td>2</td>
<td>26</td>
<td>156</td>
</tr>
<tr>
<td>AND2</td>
<td>7</td>
<td>2</td>
<td>14</td>
<td>84</td>
</tr>
<tr>
<td>OR2</td>
<td>15</td>
<td>2</td>
<td>30</td>
<td>180</td>
</tr>
<tr>
<td>MUX</td>
<td>11</td>
<td>2</td>
<td>22</td>
<td>132</td>
</tr>
<tr>
<td>Delay</td>
<td>180</td>
<td>1</td>
<td>180</td>
<td>1080</td>
</tr>
<tr>
<td>Total number of transistors</td>
<td></td>
<td></td>
<td></td>
<td>1920</td>
</tr>
</tbody>
</table>

Block diagram of 16-bit CLA adder with WCSM is shown in Figure 5.54. The delays of all the paths in the direction of propagation are balanced by using active delay elements. For simplicity, not all of the delay elements are shown in the figure. Rather, following abbreviation is used: “i_dj” means “i” signal is delayed by “j” stages, i.e. “P_d3” means P signal is delayed by 3 stages.

Summation is completed in 11 stages. Only 1 logic level is used in every stage. For having an ultra high speed adder, latches are inserted between every stage. Therefore, there are 11 register regions and 11 clock signals.
If mesochronous pipelining method or conventional pipelining methods are used, then the total number of registers in 16-bit CLA design becomes 251. When WCSM is applied, a register is only used after a logical operation occurs. Therefore, all the registers, which are used after a delay element, are omitted and replaced by delay elements. In this case, total number of latches becomes 96, which is 62% lower than the conventional or mesochronous pipelining methods.
Table 5.19 shows the comparison of the multipliers implemented using booth encoder, Wallace tree, and carry look-ahead adder blocks with conventional pipelining and WCSM, respectively. As it is seen from the table, total number of registers is decreased by 45%, when WCSM is used.

Figure 5.54 Block diagram of 16-bit CLA adder with balanced paths
Table 5-19 Overall comparison of the multipliers

<table>
<thead>
<tr>
<th></th>
<th>Conventional or mesochronous pipelining methods</th>
<th>Wave Component Sampling Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latches used in booth encoder</td>
<td>400</td>
<td>280</td>
</tr>
<tr>
<td>Latches used in Wallace tree</td>
<td>239</td>
<td>117</td>
</tr>
<tr>
<td>Latches used in carry look-ahead logic</td>
<td>251</td>
<td>96</td>
</tr>
<tr>
<td>Total number of latches</td>
<td>890</td>
<td>493</td>
</tr>
<tr>
<td>Percentage of reduction in number of latches</td>
<td>-</td>
<td>45%</td>
</tr>
</tbody>
</table>

5.7.4  CLA Adder Design with 4-input logic gates

Carry look-ahead adders are used to reduce the total latency of the outputs. However, when using 2-input logic gates, group propagate and group generate signals cannot be produced fast enough, which decreases the efficiency of carry look-ahead logic. Therefore, 16 bit carry look-ahead adder using 4 input logic blocks is implemented, which is shown in Figure 5.55. As it is seen from the figure, first carry output (C4) is produced with a latency of 3 logic levels. At the same time instant, GG1 & GG2 (group generate 1-2) signals are produced, therefore C8 and C12 are obtained with a latency of 4. Therefore, overall summation is completed with a latency of 5. The reduction of latency brings a lot of reduction in the number of delay elements in wave-pipelining. Therefore, the delay characteristics of 4-input logic elements are analyzed.
Figure 5.55 16-bit carry lookahead adder with 4-input logic gates
5.8 Delay analysis of logic blocks with 4 inputs

Delay difference value of Classical CMOS logic is much higher than that of the pass transistor logic. One of the main reasons is that the number of pull-up or pull-down paths depends on the input data pattern. Considering CMOS Nand gate with 2 inputs, if both of the inputs are LOW, then there will be two pull-up paths. For limiting the current, [17] proposed to use a serial transistor at the top. Figure 5.56 shows a 4 input CMOS AND gate with a serial current limiting pull up transistor at the top.

The fastest transition at the output occurs when all of the inputs switch from HIGH to LOW, which causes all of the pmos transistors to join pull-up current. The delay time at the output becomes 92ps in this case. The slowest transition occurs when all the inputs are HIGH and the last input (connected to the nmos transistor closest to ground) switches to LOW, i.e. ABCD input pattern is 1111 and switches to 1110. In that case, output delay time is 173ps. If the input pattern is 1111 and switches to 0111, then the delay time at the output becomes 116ps. Therefore, the place of the switching transistor at the serial pull-down path is another main reason of the delay difference at the output. Table 5.20 shows the delay values of the 4 input AND gate depending on the input patterns of concern. It can be seen that, delay difference value is 81ps. Table 5.21 shows the output delay values when there is no limiting transistor at the top. In that case, the delay difference value becomes 84ps, and the slowest path becomes pull-down path, rather than one transistor pull-up of the previous one.
Table 5-20 Delays of 4-input CMOS with a limiting pmos of W=900nm at the top

<table>
<thead>
<tr>
<th>ABCD1</th>
<th>ABCD2</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1111</td>
<td>139ps</td>
</tr>
<tr>
<td>1111</td>
<td>0000</td>
<td>92ps</td>
</tr>
<tr>
<td>0111</td>
<td>1111</td>
<td>107ps</td>
</tr>
<tr>
<td>1111</td>
<td>0111</td>
<td>116ps</td>
</tr>
<tr>
<td>1110</td>
<td>1111</td>
<td>143ps</td>
</tr>
<tr>
<td>1111</td>
<td>1110</td>
<td>173ps</td>
</tr>
</tbody>
</table>

Delay difference value: 81ps

Table 5-21 CMOS_AND4_v1_delays with no limiting transistor at the top

<table>
<thead>
<tr>
<th>ABCD1</th>
<th>ABCD2</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1111</td>
<td>131ps</td>
</tr>
<tr>
<td>1111</td>
<td>0000</td>
<td>51ps</td>
</tr>
<tr>
<td>0111</td>
<td>1111</td>
<td>107ps</td>
</tr>
<tr>
<td>1111</td>
<td>0111</td>
<td>89ps</td>
</tr>
<tr>
<td>1110</td>
<td>1111</td>
<td>135ps</td>
</tr>
<tr>
<td>1111</td>
<td>1110</td>
<td>121ps</td>
</tr>
</tbody>
</table>

Delay difference value: 84ps

It is possible to use a small pmos limiting transistor at the top, but in that case the rise time will be very high, which is not a recommended case in high speed wave-pipelined design. A 2 input AND gate is also designed which can be seen in Figure 5.57. The output delay values of this circuit can be seen in Table 5.22, and the delay difference value at the output becomes 27ps.
Using 2 input AND gates, another 4 input AND gate is designed, which can be seen in Figure 5.58. Table 5.23 shows the output delay values where the delay difference value is 58ps. Comparison with the CMOS_AND4_v1 gate is given in Table 5.24. Although the delay difference value decreases from 81ps to 58ps, the number of transistors is increased from 11 to 21 and the latency is also increased from 173ps to 234ps. Therefore, the choice will be a design trade-off.
Figure 5.58 4-input CMOS And gate constructed with 2-input gates

Table 5-23 Delay values of CMOS_AND4_v2

<table>
<thead>
<tr>
<th>AB1</th>
<th>AB2</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>0111</td>
<td>1111</td>
<td>234ps</td>
</tr>
<tr>
<td>1111</td>
<td>0111</td>
<td>198ps</td>
</tr>
<tr>
<td>0000</td>
<td>1111</td>
<td>209ps</td>
</tr>
<tr>
<td>1111</td>
<td>0000</td>
<td>176ps</td>
</tr>
</tbody>
</table>

Delay difference value

58ps
<table>
<thead>
<tr>
<th></th>
<th>CMOS_AND4_v1</th>
<th>CMOS_AND4_v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of trans.</td>
<td>11</td>
<td>21</td>
</tr>
<tr>
<td>Output Latency</td>
<td>176ps</td>
<td>234ps</td>
</tr>
<tr>
<td>Delay difference at the output</td>
<td>81ps</td>
<td>58ps</td>
</tr>
</tbody>
</table>

Table 5-24 Comparison of CMOS_AND4_v1 and CMOS_AND4_v2.
CHAPTER 6

CONCLUSION

In this thesis a novel wave-pipelining methodology named as Wave Component Sampling Method (WCSM) is developed and discussed. In all of the previous pipelining methods such as conventional pipelining, wave pipelining, and mesochronous pipelining, all of the components of propagating waves are sampled whenever they arrive to a synchronization stage. However, WCSM allows partial sampling of the signal components of the propagating waves. Only the components, whose minimum and maximum delay differences reach to the tolerable value, are sampled and the other signal components are delayed using active delay elements. Therefore, this methodology promises significant reduction in the number of the sampling flip-flops or latches.

To demonstrate the effectiveness of this method and to compare it with the mesochronous pipelining methodology, two 8x8 bit multipliers are implemented using mesochronous pipelining scheme and WCSM, respectively. Several optimizations are performed in the design of sub blocks for achieving high performance multipliers with low power consumption.

Minimizing the delay differences between propagating waves is very important for having high speed multiplication with a small number of pipeline stage. Therefore, a generic transmission gate logic block, which has minimum delay variation at the output depending on the input pattern, is designed. This transmission gate logic block has three inputs and performs the output function of $Q=\overline{(X*Z+NX*Y)}$. The minimum and maximum delay variation of this generic logic block is kept within 27ps.

In the implementation of multiplier, both the normal and complementary signals are required. When the normal and complementary signals are generated using different logic blocks, a delay difference between them occurs. And when they are connected to the gates of
the transistors of cascading transmission gate logic blocks, the asymmetry between the transitions of normal and complementary signals creates conflict on the succeeding output. Therefore, simultaneous generation of normal and complementary signals is important. A method for generating normal and complementary signals simultaneously with symmetric transitions is proposed. Instead of using separate logic blocks, both the normal and the complementary outputs are generated using same transmission gate logic. Another transmission gate, which is always “ON” and has same delay value with that of the inverter, is used to obtain both of the normal and the complementary signals. The reduction in the number of logic blocks also reduces the drive strength required at the output of the preceding logic block.

Half adder and full adder blocks are designed using generic transmission gate logic blocks. Half adder has single stage, where full adder has two stages of logic operation. The delay variation at the output of half adder and full adder blocks are 27ps, and 62ps, respectively.

In the multiplier, fan out is kept as at most two for having high speed operation. However, in the generation of partial products, a fan out of 8 is needed. For driving 8 inputs, the drive strength of the single bit of multiplier is increased by a buffer composed of a cascade of 3 inverters with increasing transistor sizes. Buffered signals are used to drive the gates of transistors in the transmission gate logic, since driving the gates requires less drive capability than driving drain or source of the transistors.

While propagating, the width of the waves decreases. They must be sampled before their aperture becomes too small which creates setup and hold time violations. C2MOS latch is designed and used to sample narrow waves with a minimum aperture of 120ps. Latches are operational at a frequency of 5GHz.

8x8-bit carry save adder (CSA) multipliers are implemented with mesochronous pipelining scheme and WCSM. For comparing the methods adequately, same structure of [24] is used, in which a layer of register is used after 3 layers of full adders. Full custom design methodology with Cadence design tools is used and UMC-0.18µm CMOS technology is employed in the implementation. The operating frequency of both of the multipliers is 3GHz. The number of the latches is decreased by 41% when WCSM is employed. The reduction in the number of latches also decreases the power consumption of the associated clocking
network. Post layout simulations show that total power of the chip is also decreased by 9.5%, without any performance loss.

For investigating the benefits of WCSM in higher level pipelined circuits, two CSA multiplier using mesochronous pipelining method and WCSM are implemented with a fully pipelined structured. A register layer is used after all of the logic layers. In this case, the operating frequency of the multipliers is increased to 5GHz, which is the fastest multiplier using 0.18µm CMOS technology. In the multiplier employing WCSM, the number of the registers is decreased by 45%. The power of the multiplier is also decreased by 18.4%. This demonstrates that, benefits of WCSM increase when the number of pipeline stages and operating frequency of the circuit increase.

WCSM is also applied to the other multiplier structures for observing its effects with different circuit structures. Booth encoder, Wallace tree and carry look-ahead blocks for 8x8 bit multiplication are designed with full pipelined structures, and then WCSM is applied to them. In the design of booth encoder, the number of registers decreases from 400 to 280, which constitutes a reduction of 30%. In the design of Wallace tree, the number of registers is decreased from 239 to 117, with a reduction of 51%. The number of the registers is also decreased from 251 to 96 in the implementation of 16 bit carry look-ahead adder, constituting a reduction of 62%. The overall reduction in the implementation of 8x8 bit multiplier employing booth encoder, Wallace tree and carry look-ahead adder is from 890 to 493, which constitutes a reduction of 45%.

WCSM is a novel pipelining methodology which provides a significant reduction in the number of registers, without a performance loss. For future research, the application of WCSM to different pipelined circuits could be investigated. Crypto processes, filter applications, multiplier and accumulators (MAC), memory structures, communications algorithms like Viterbi decoders etc are good candidates for the application of WCSM.

Besides the benefits of WCSM, its design complexity is high. Computer Aided Design (CAD) tools could be developed for automatically implementing circuits using WCSM.
BIBLIOGRAPHY


APPENDIX A

HDL CODES OF MULTIPLIER BLOCKS

A.1 Verilog HDL code of top module (booth3_bias_wallace_CLA16)

module booth3_bias_wallace_CLA16(
    input [7:0] a,//multiplicand
    input [7:0] b,//multiplier
    output [15:0] c,
    output [15:0] sum_wallace,
    output [15:0] sum_cla,
    output cla_overflow
);

wire [3:0] sel0, sel1;
wire [2:0] sel2;
wire [10:0] mux0_out,mux1_out,mux2_out;

wire [15:0] pp0,pp1,pp2,pp0_y,pp0_s,pp1_y,pp1_s,pp2_y;
wire s0,s1;

wire [15:0] comp;

wire [15:0] sum1,sum2,sum;

//wire [15:0] sum_wallace;
wire [15:0] sum_cla_compsuz;

KplusM
KplusM( .M(a[7:0]),
.K_M(K_M[10:0])
);

Kplus2M
Kplus2M
  .M(a[7:0]),
  .K_2M(K_2M[10:0])
):

Kplus3M
Kplus3M
  .M(a[7:0]),
  .K_3M(K_3M[10:0])
):

Kplus4M
Kplus4M
  .M(a[7:0]),
  .K_4M(K_4M[10:0])
):

booth3_mux
mux0(
  .sel(sel0[3:0]),
  .mux_in_0(K_M[10:0]), //K+M //ilk bit y0
  .mux_in_1(K_2M[10:0]), //K+2M //ilk bit y0
  .mux_in_2(K_3M[10:0]), //K+3M //ilk bit y0
  .mux_in_3(K_4M[10:0]), //K+4M //ilk bit y0
  .mux_out(mux0_out[10:0])
),

mux1(
  .sel(sel1[3:0]),
  .mux_in_0(K_M[10:0]), //K+M //ilk bit y0
  .mux_in_1(K_2M[10:0]), //K+2M //ilk bit y0
  .mux_in_2(K_3M[10:0]), //K+3M //ilk bit y0
  .mux_in_3(K_4M[10:0]), //K+4M //ilk bit y0
  .mux_out(mux1_out[10:0])
)

booth3_mux_sondaki
mux2(
  .sel(sel2[2:0]), //sonuncu muxta select 3 bit
  .mux_in_0(K_M[10:0]), //K+M //ilk bit y0
  .mux_in_1(K_2M[10:0]), //K+2M //ilk bit y0
  .mux_in_2(K_3M[10:0]), //K+3M //ilk bit y0
  .mux_in_3(K_4M[10:0]), //K+4M //ilk bit y0
  .mux_out(mux2_out[10:0])
);
A.2 Verilog-HDL code of KplusM

module KplusM(
    input [7:0] M,
    output [10:0] K_M //10.bit y0
); //output y0
    assign K_M[10]=M[5];
    assign K_M[9:0]={2:0,M[7:6],!M[5],M[4:0]};
endmodule
A.3 Verilog-HDL code of Kplus2M

module Kplus2M(
  input [7:0] M,
  output [10:0] K_2M //10.bit y0
); //output y0

//assign y0=M[4];
assign K_2M[10:0]={M[4],1'd0,M[7:5],!M[4],M[3:0],1'd0};

A.4 Verilog-HDL code of Kplus3M

module booth3_mux(
  input [3:0] sel,
  input [10:0] mux_in_0, //K+M //ilk bit y0
  input [10:0] mux_in_1, //K+2M //ilk bit y0
  input [10:0] mux_in_2, //K+3M //ilk bit y0
  input [10:0] mux_in_3, //K+4M //ilk bit y0
  output reg [10:0] mux_out //ilk bit y0 cikisi
); //ilk bit y0 cikisi
always @(sel or mux_in_0 or mux_in_1 or mux_in_2 or mux_in_3)
case (sel) //ilk bit y0 cikisi
  4'd0://K+0
    begin
      mux_out[10]<=0;//y0
      mux_out[9:6]<=4'd0;
      mux_out[5]<=1;
      mux_out[4:0]<=5'd0;
    end
  4'd1://K+M
    mux_out[10]<=mux_in_0[10:0];//y0
  4'd2://K+M
    mux_out[10]<=mux_in_0[10:0];//y0
  4'd3://K+2M
    mux_out[10]<=mux_in_1[10:0];
  4'd4://K+2M
    mux_out[10]<=mux_in_1[10:0];
  4'd5://K+3M
    mux_out[10]<=mux_in_2[10:0];
  4'd6://K+3M
    mux_out[10]<=mux_in_2[10:0];
  4'd7://K+4M
    mux_out[10]<=mux_in_3[10:0];
  4'd8://K-4M
)
A.5 Verilog-HDL Code of Kplus4M

module Kplus4M(
  input [7:0] M,
  output [10:0] K_4M //10.bit y0
  //output y0
);

//assign y0=M[3];
assign K_4M[10:0]=(M[3],M[7:4],!M[3],M[2:0],2'd0);

endmodule

A.6 Verilog-HDL Code of Booth3Mux

module booth3_mux(
input [3:0] sel,
input [10:0] mux_in_0, //ilk bit y0
input [10:0] mux_in_1, //ilk bit y0
input [10:0] mux_in_2, //ilk bit y0
input [10:0] mux_in_3; //ilk bit y0

output reg [10:0] mux_out; //ilk bit y0 cikisi
);
always @(sel or mux_in_0 or mux_in_1 or mux_in_2 or mux_in_3)
case (sel)
4’d0://K+0
begin
mux_out[10]<=0; //y0
mux_out[9:6]<=4’d0;
mux_out[5]<=1;
mux_out[4:0]<=5’d0;
end
4’d1://K+M
mux_out[10:0]<=mux_in_0[10:0]; //y0
4’d2://K+M
mux_out[10:0]<=mux_in_0[10:0]; //y0
4’d3://K+2M
mux_out[10:0]<=mux_in_1[10:0];
4’d4://K+2M
mux_out[10:0]<=mux_in_1[10:0];
4’d5://K+3M
mux_out[10:0]<=mux_in_2[10:0];
4’d6://K+3M
mux_out[10:0]<=mux_in_2[10:0];
4’d7://K+4M
mux_out[10:0]<=mux_in_3[10:0];
4’d8://K-4M
mux_out[10:0]<=-mux_in_3[10:0];
4’d9://K-3M
mux_out[10:0]<=-mux_in_2[10:0];
4’d10://K-3M
mux_out[10:0]<=-mux_in_2[10:0];
4’d11://K-2M
mux_out[10:0]<=-mux_in_1[10:0];
4’d12://K-2M
mux_out[10:0]<=-mux_in_1[10:0];
4’d13://K-M
mux_out[10:0]<=-mux_in_0[10:0];
4’d14://K-M
A.7 Verilog-HDL code of wallace_booth3_bias_8x8

module wallace_booth3_bias_8x8(
    input [15:0] pp0,
    input [15:0] pp1,
    input [15:0] pp2,
    input [15:0] pp0_y,
    input [15:0] pp1_y,
    input [15:0] pp2_y,
    input [15:0] pp0_s,
    input [15:0] pp1_s,
    output [15:0] sum1,
    output [15:0] sum2,
    output [15:0] sum
);

wire [15:0] S1,C1,S2,C2;

//Level 1
half_adder u11(.A(pp0[0]),.B(pp0_s[0]),.S(S1[0]),.C_out(C1[0]));
assign S1[1]=pp0[1]; assign C1[1]=1’d0; //dogrudan u12
half_adder u13(.A(pp0[2]),.B(pp1_s[2]),.S(S1[2]),.C_out(C1[2]));
full_adder u14(.A(pp0[3]),.B(pp1[3]),.C_in(pp1_s[3]),.S(S1[3]),.C_out(C1[3]));
full_adder u15(.A(pp0[4]),.B(pp1[4]),.C_in(pp1_s[4]),.S(S1[4]),.C_out(C1[4]));
half_adder u16(.A(pp0[5]),.B(pp1[5]),.S(S1[5]),.C_out(C1[5]));
full_adder u17(.A(pp0[6]),.B(pp1[6]),.C_in(pp2[6]),.S(S1[6]),.C_out(C1[6]));
full_adder u18(.A(pp0[7]),.B(pp1[7]),.C_in(pp2[7]),.S(S1[7]),.C_out(C1[7]));
full_adder u19(.A(pp0[8]),.B(pp1[8]),.C_in(pp2[8]),.S(S1[8]),.C_out(C1[8]));
full_adder u110(.A(pp0[9]),.B(pp1[9]),.C_in(pp2[9]),.S(S1[9]),.C_out(C1[9]));
module half_adder(
    input A,
    input B,
    output S,
    output C_out
);

assign S=A ^ B;
assign C_out= A && B;

A.8 Verilog-HDL code of half adder

module half_adder(
    input A,
    input B,
    output S,
    output C_out
);

assign S=A ^ B;
assign C_out= A && B;

A.9 Verilog-HDL code of full adder

module full_adder(
    input A,
    input B,
    input C_in,
    output S,
    output C_out
);

assign S=A ^ B ^ C_in;
assign C_out= (A&B) | (A&C_in) | (B&C_in);
endmodule

A.10 Verilog-HDL code of 16 bit Carry lookahead adder

module CLA_16(
    input [15:0] A,
    input [15:0] B,
    input Cin,
    output PP,
    output PG,
    output [15:0] Sum,
    output Cout
);

wire [3:0] P,G,C;
CLA_4bit
    u1(.A(A[3:0]),.B(B[3:0]),.Cin(C[0]),.Sum(Sum[3:0]),.PP(P[0]),.PG(G[0]))),
    u2(.A(A[7:4]),.B(B[7:4]),.Cin(C[1]),.Sum(Sum[7:4]),.PP(P[1]),.PG(G[1]))),
    u4(.A(A[15:12]),.B(B[15:12]),.Cin(C[3]),.Sum(Sum[15:12]),.PP(P[3]),.PG(G[3])));

assign C[0]=Cin;
assign C[1]=G[0] | P[0] & C[0];
A.11 Verilog-HDL code of 4-bit carry lookahead adder

module CLA_4bit(
    input [3:0] A,
    input [3:0] B,
    input Cin,
    output PP,
    output PG,
    output [3:0] Sum,
    output Cout
);

wire [3:0] P,G,C;

full_adder_CLA
u1(.A(A[0]),.B(B[0]),.Cin(C[0]),.Sum(Sum[0]),.P(P[0]),.G(G[0])),
    u2(.A(A[1]),.B(B[1]),.Cin(C[1]),.Sum(Sum[1]),.P(P[1]),.G(G[1])),
    u3(.A(A[2]),.B(B[2]),.Cin(C[2]),.Sum(Sum[2]),.P(P[2]),.G(G[2])),
    u4(.A(A[3]),.B(B[3]),.Cin(C[3]),.Sum(Sum[3]),.P(P[3]),.G(G[3]));

assign C[0]=Cin;
assign C[1]=G[0] | (P[0] & C[0]);
endmodule
A.13 Verilog-HDL code of testbench

module tb_booth3_bias_wallace_cla16;

    // Inputs
    reg [7:0] a;
    reg [7:0] b;

    // Outputs
    wire [15:0] c,sum_wallace,sum_cla;
    wire cla_overflow;
    reg [15:0] a_b;

    // Instantiate the Unit Under Test (UUT)
    booth3_bias_wallace_CLA16 uut (  
        .a(a),  
        .b(b),  
        .c(c),  
        .sum_wallace(sum_wallace),  
        .sum_cla(sum_cla),  
        .cla_overflow(cla_overflow)  
    );

always @(a or b)
    a_b[15:0]=a[7:0]*b[7:0];

    reg clk;

    initial
    begin clk=0;
    #1;  
    forever
    begin
        clk=!clk;
        #2;  
    end
end


reg hata;
initial hata=0;
reg hata_wallace;
initial hata_wallace=0;
reg hata_cla;
initial hata_cla=0;

always @(posedge clk)
if (a_b[15:0]!=c[15:0])
    hata<=1;
else
    hata<=0;

always @(posedge clk)
if (a_b[15:0]!=sum_wallace[15:0])
    hata_wallace<=1;
else
    hata_wallace<=0;

always @(posedge clk)
if (a_b[15:0]!=sum_cla[15:0])
    hata_cla<=1;
else
    hata_cla<=0;

initial begin
    // Initialize Inputs
    a = 0;
b = 0;

    // Wait 100 ns for global reset to finish
    #100;

    forever
    begin
        a=$random;
b=$random;
        #10;

        // Add stimulus here
    end

end

dmodule
CURRICULUM VITAE

PERSONAL INFORMATION
Surname, Name: Sever, Refik
Nationality: Turkish (TC)
Date and Place of Birth: 4th August 1979, Ankara
Phone: +90 242 310 63 89
email: refiksever@akdeniz.edu.tr

EDUCATION

<table>
<thead>
<tr>
<th>Degree</th>
<th>Institution</th>
<th>Year of Graduation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS</td>
<td>METU Electrical &amp; Electronics Engineering</td>
<td>2003</td>
</tr>
<tr>
<td>BS</td>
<td>METU Electrical &amp; Electronics Engineering</td>
<td>2001</td>
</tr>
<tr>
<td>High School</td>
<td>Ankara Atatürk Anatolian High School</td>
<td>1997</td>
</tr>
</tbody>
</table>

WORK EXPERIENCE

<table>
<thead>
<tr>
<th>Year</th>
<th>Place</th>
<th>Enrollment</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010 – Present</td>
<td>Akdeniz University</td>
<td>Instructor</td>
</tr>
<tr>
<td>2000-2009</td>
<td>TÜBİTAK UZAY</td>
<td>Senior Researcher, Project Manager, Design Engineer</td>
</tr>
</tbody>
</table>

FOREIGN LANGUAGES
English

RECENT PUBLICATIONS

Refik Sever, Murat Askar, “8x8-Bit Multiplier Designed With a New Wave-Pipelining Scheme,” in proceedings of International Symposium on Circuits and Systems (ISCAS-2010), May 31- June 3, Paris.


