Binary Arithmetic for Finite-Word-Length Linear Controllers : MEMS Applications

Abdelkrim Kamel Oudjida

SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UFC UNIVERSITÉ DE FRANCHE-COMTÉ

Binary Arithmetic for Finite-Word-Length Linear Controllers: MEMS Applications

ABDELKRIM KAMEL OUDJIDA

SPIM

Thèse de Doctorat

école doctorale sciences pour l'ingénieur et microtechniques

UFC UNIVERSITÉ DE FRANCHE-COMTÉ

THÈSE présentée par

Abdelkrim Kamel OUDJIDA

pour obtenir le

Grade de Docteur de

L'Université de Franche-Compté

Spécialité: **Automatique**

Binary Arithmetic for Finite-Word-Length Linear Controllers: MEMS Applications

Unité de Recherche:
FEMTO-ST, UMR CNRS 6174

Soutenue le 20 Janvier 2014 devant le jury:

Amara AMARA	Président	Professeur des Universités, ISEP, Paris
Adel BELOUCHRANI	Rapporteur	Professeur, ENP, Alger
Chouki AKTOUF	Rapporteur	Maître de Conférences HDR, INP de Grenoble
Thibault HILAIRE	Examinateur	Maître de Conférences, Université Paris 6
Brahim BOUZOUIA	Examinateur	Directeur de Recherche, CDTA, Alger
Nicolas CHAILLET	Directeur de Thèse	Professeur des Universités, Université de Besançon

Gödel's Incompleteness Theorems

Theorem 1 – Blind spot: *There is a sentence G such that if the formal system is consistent, G is not a theorem nor is it not a theorem.*

Theorem 2 – Consistency: *No consistent theory, with a certain amount of arithmetic, can prove its own consistency.*

Acknowledgments

I wish to thank my supervisor, Prof. Nicolas Chaillet, for the support and trust he so generously provided me in the course of the accomplishment of my task. His continuous enthusiasm and encouragement constituted the driving force of this work.

I am deeply grateful to Dr. Brahim Bouzouia, Director of CDTA, for his help and support during all the period of the thesis.

I am so indebted to Prof. Noureddine Zerhouni, Mr. Karim Henda, and Prof. Yassine Haddab for their precious help. Without their effective contribution, this thesis would not have been concretized.

My special thanks go to my two colleagues, Mohamed Lamine Berrandjia and Ahmed Liacha, for their technical help in a warm and joyful atmosphere.

I am much thankful to my ex-student, Khaled Tahraoui, for having provided me with a tremendous number of papers that much contributed to enhance the quality of the work.

I would like to acknowledge Dr. Joël Agnus, research engineer within AS2M department of Femto-St institute, for his valuable advices.

Finally, I would like to thank my wife and sons for their great patience and support during my seemingly never-ending nights preparing the thesis.

Abstract

This thesis addresses the problem of optimal hardware-realization of finite-word-length (FWL) linear controllers dedicated to MEMS applications. The biggest challenge is to ensure satisfactory control performances with a minimal hardware. To come up, two distinct but complementary optimizations can be undertaken: in control theory and in binary arithmetic. Only the latter is involved in this work.

Because MEMS applications are targeted, the binary arithmetic must be fast enough to cope with the rapid dynamic of MEMS; power-efficient for an embedded control; highly scalable for an easy adjustment of the control performances; and easily predictable to provide a precise idea on the required logic resources before the implementation.

The exploration of a number of binary arithmetics showed that radix-2^r is the best candidate that fits the aforementioned requirements. It has been fully exploited to designing efficient multiplier cores, which are the real engine of the linear systems.

The radix-2^r arithmetic was applied to the hardware integration of two FWL structures: a linear time variant PID controller and a linear time invariant LQG controller with a Kalman filter. Both controllers showed a clear superiority over their existing counterparts, or in comparison to their initial forms.

Glossary

Abstraction Simplification of details, approximation of complex problems

ADC Analog to Digital Converter

AFM Atomic Force Microscopy

ALU Arithmetic and Logic Unit

ASIC Application Specific Integration Circuit

Ath Adder Depth, the maximum number of serial adder-operations from input to output

Avg Average number of additions

BHM Bull Horrocks modified, an existing heuristic MCM algorithm

BIGE Bounded Inverse Graph Enumeration, an optimal SCM algorithm

BMA Booth Multiplication Algorithm, an existing algorithm for signed multiplication

CAD Computer-aided design, tools for design automation

CDE Common Digit Elimination

CLB Configurable Logic Bloc

CMOS Complementary Metal Oxide Semiconductor

COTS Commercial Off-The-Shelf

CSD Canonical Signed Digit, the SD form with no adjacent nonzero digits and the minimum number of nonzero digits

CSE Common Subexpression Elimination, a framework for solving SCM and MCM

DAC Digital to Analog Converter

DAG Directed Acyclic Graphs, a framework for solving SCM and MCM

DBNS Double Base Number System, an existing number representation system

DFS Dynamic Frequency Scaling

Digit Clashing The CSE problem of disappearing patterns due to colliding digits

DMAC Double Multiply-And-Accumulate

DSP Digital-Signal-Processor/Processing

FF Flip-Flop

FPGA Field Programmable Gate Array

FPR Fixed-Point Representation

FWL Finite Word Length

H(k) Heuristic with k extra nonzero digits, an existing heuristic SCM algorithm

H(k)+ODP The H(k) algorithm with ODPs, a proposed heuristic SCM algorithm

Hcub Cumulative Benefit Heuristic, an existing MCM algorithm

HDL Hardware Description Language

Heuristic An effective but potentially suboptimal method for solving a problem

HIS Host Side Interface

HPM High Performance Multiplication, an existing adder reduction technique

IOB Input Output Block

Logic resources An abstraction of the amount of silicon required to implement a logic function

LQG Linear Quadratic Gaussian

LTI Linear Time Invariant

LTV Linear Time Variant

MAC Multiply-And-Accumulate

MAG Minimized Adder Graph, an existing optimal SCM algorithm

MBMA Modified Booth Multiplication Algorithm, , an existing algorithm for signed multiplication

MCM Multiple Constant Multiplication, find a low-cost add-shift-subtract realization of multiplication by each of the given constants

MEMS Micro-Electro-Mechanical Structure

MM Matrix Multiplication

MPC Model Predictive Control

MPM Multi-Precision Multiplication

MSD Minimal Signed Digit, any SD representation with the minimum number of nonzero digits

MV Multiplication by a Variable

NEMS Nano-Electro-Mechanical Structure

NP-hard Non-deterministic Polynomial-time hard

ODMAC Optimized Double Multiply-And-Accumulate

ODP Overlapping Digit Pattern, a proposed technique for partially resolving the CSE digit clashing problem

OS Operating System

Pattern (CSE) A collection of signed digits that define how existing terms are added-operated together

PC Personal Computer

PID Proportional Integral Derivative, an existing control law

PLD Programmable Logic Device

PPG Partial Product Generator

RAG-*n* n-dimensional Reduced Adder Graph, an existing heuristic MCM algorithm

RMRMA Recursive-Multibit-Recoding Multiplication Algorithm, a new algorithm

RNS Residue Number System, an existing number representation system

RTL Register Transfer Level

RTOS Real Time Operating System

SCM Single Constant Multiplication, same as MCM but for a single constant

SD Signed Digit, a recoding used in the CSE framework

Search space The set of all solutions that can be found by an algorithm, this is smaller than the solution space for heuristic algorithms

SEM Scanning Electron Microscopy

SiGe Silicon Germanium

SM Sign and Magnitude

SOC System On Chip

Solution space The set of all feasible solutions

SRAM Static Random Access Memory

TC True and Complement

TSMC Taiwan Semiconductor Manufacturing Company

Upb Upper bound, maximum number of additions

VHDL Very high speed integrated circuit Hardware Description Language

VLIW Very Large Instruction Word

Contents

List of Figures

List of Tables

Personal Publications

Publications in Indexed Journals

- A.K. Oudjida, N. Chaillet, "Radix-2^r Arithmetic for the Multiplication by a Constant," Accepted for publication on January 20th 2014. IEEE Trans. on Circuits and Systems II, Expess Brief.
- A.K. Oudjida, N. Chaillet, A. Liacha, M.L. Berrandjia, and M. Hamerlain, "Design of High-Speed and Low-Power Finite-Word-Length PID Controllers," Journal of Control Theory and Applications (JCTA), vol. 12, N° 1, pp.68-83, 2014, ISSN:1672-6340, SPRINGER, Germany.
- A.K. Oudjida, N. Chaillet, M.L. Berrandjia, and A. Liacha, "A New High Radix-2^r ($r \geq 8$) Multibit Recoding Algorithm for Large Operand Size ($N \geq 32$) Multipliers,". Journal of Low Power Electronics (JOLPE), vol. 9, N° 1, pp. 50-62, April 2013, ISSN:1546-1998, American Scientific Publishers (ASP), USA.
- A.K. Oudjida, N. Chaillet, A. Liacha, and M.L. Berrandjia, "A New Recursive Multibit Recoding Algorithm for High-Speed and Low-Power Multiplier,". Journal of Low Power Electronics (JOLPE), vol. 8, N° 5, pp. 579-594, Dec.2012, ISSN:1546-1998, American Scientific Publishers (ASP), USA.

Publications in International Conferences

- A.K. Oudjida, M.L. Berrandjia, and N. Chaillet, "A New Low-Power Recoding Algorithm for Multiplierless Single/Multiple Constant Multiplication," Proceedings of the 12th edition of IEEE-FTFC Low-Voltage Low-Power Conference, ISBN:978-1-4673-6104-0/13, June 19-21 2013, Paris, France.
- A.K. Oudjida, N. Chaillet, A. Liacha, and M.L. Berrandjia "New High-Speed and Low-Power Radix-2^r Multiplication Algorithms," Proceedings of the 11th edition of IEEE-FTFC Low-Voltage Low-Power Conference, ISBN:978-1-4673-0821-2/12, June 06-08 2012, Paris, France.
- A.K. Oudjida, N. Chaillet, A. Liacha, M. Hamerlain, and M.L. Berrandjia, "High-Speed and Low-Power PID Structures for Embedded Applications," Proceedings of the 21th Edition of the International Workshop on Power and Timing Modeling, Optimization and Simulation PATMOS, LNCS 6951, pp. 257-266, SPRINGER-VERLAG Editor, September 26-29 2011, Madrid, Spain.

Chapter 1

General Introduction

Chapter 1

General Introduction

This chapter defines the problematic issue treated in this thesis and indicates the objective to achieve. It sets the main requirements and specifications that fixe the research scope and limitation, and informs on the essential contribution of the thesis. It also gives an idea on the organization of this manuscript.

1.1 Motivation and Problem Statement

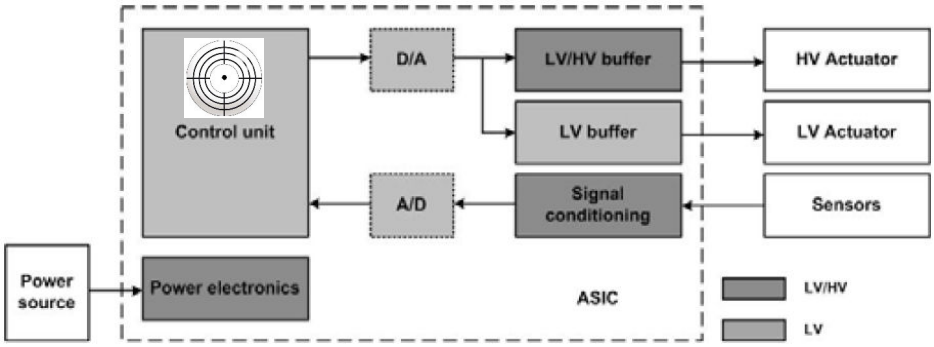
As a MEMS application, micromanipulation is a field which has the particularity of being very sensitive to noise. To reach the desired force/position accuracy in the gripping of micrometric objects, the development of advanced control methods is necessary. The latter often results in control laws, filters, and algorithms which are executed in real-time on platforms that are in total disproportion with the dimensions of the manipulator tool (micro-gripper) and the micro-objects manipulated. They are bulky and expensive, and have the huge drawback of preventing any embedded or mobile utilisation (*portability*). Therefore, the *hardware integration* of these control laws and all related problems is quite worthwhile, and constitutes a real challenge.

With a sound experience in micromanipulation, and being aware of the hardware integration as an ineluctable future step, the Automatic Control and Micro-Mechatronic Systems (AS2M) department of FEMTO-ST Institute, Besançon, initiated in 2010 a new research project called embedded electronics for MEMS. This project has been elaborated in conjunction with the Microelectronics and Nanotechnology Division (DMN) of CDTA, Algiers, joining their complementary skills together for the same objective. It is within the framework of this collaboration project CDTA/FEMTO-ST that the present thesis has evolved.

The most commonly used platform for micromanipulation setups is composed of a PC (with Matlab/Simulink or Labview software for instance) connected to a processor (dSPACE compatible for instance) for the real-time control, incorporating a sophisticated floating-point arithmetic unit with a high precision and large dynamic range. One intermediate step toward portability will be to *substitute* the calculation unit used for the real-time calculation by an FPGA board, wherein only the *control algorithm* is embedded [1]. The ideal solution would be to incorporate the whole electronics (control, power, signal conditioning, and conversion) into a low-power ASIC [2], directly connected to the micro-gripper (Fig. 1.1). Such a solution provides 100% autonomy, but its implementation requires varied and multidisciplinary skills.

1.2 Objective of the Thesis

The replacement of the dSPACE unit by an FPGA control board will primarily arise the problem of stability. The latter is due to the inaccuracies of calculations caused by the use of an approximate representation of real numbers (Fixed-point) with a reduced precision and dynamic range. The fixed-point representation is dictated in this specific case by implementation considerations. Not only it is much easier to implement, but also leads to more computational speed and less power consumption and logic resources, in comparison to floating-point representation.



Source : [2]

FIGURE 1.1 – ASIC solution for MEMS control providing 100% autonomy.

More precisely, the use of fixed-point instead of floating-point will exaggerate the quantification and rounding effects on the transfer function of the system (disturbance of the pole-zero position). This effect is assimilated to an additional source of noise (white noise) which is added to the already existing environmental noise. Consequently, the biggest challenge is to ensure satisfactory control performances with a minimum hardware (minimum word sizing of the controller structure). This problematic issue is well known under the name of optimal realizations of digital Finite-Word-Length (FWL) controllers. It involves optimizations in control theory and binary arithmetic.

The objective of the thesis is the development of a new binary arithmetic adapted to the FWL problem, enabling an easy generation of minimum word-size controller-structures with acceptable control performances.

1.3 Requirements and Specifications

The hardware integration of FWL controllers is a vast problematic issue. In order to define the research scope and limitation of the thesis, a number of requirements and specifications are established. They are summarized as follows:

- Only linear controllers are considered. This includes both the Linear Time-Invariant (LTI) and Time-Variant (LTV) controllers;
- Only binary arithmetic is used as a mean of optimization. Are not considered the optimization methods based on the utilization of sparse and insensitive matrices with minimization of transfer-function or pole-zero disturbance.
- A new fixed-point binary arithmetic is to be proposed. It must be:
  - Fast to cope with the high dynamics of MEMS;
  - Power efficient for an embedded control;
  - Highly scalable for an easy adjustment of the control performances;
  - Predictable to provide a more-or-less precise idea on the speed and logic resources before the implementation. This feature is very useful to the automatic synthesis of linear controllers.
- Use of the same word size and the same fixed-point position in the word for all the coefficients and variables involved. Are not addressed issues related to the order of sum-of-product operations, nor those pertaining to different rounding modes (truncation, to-the-nearest, etc.).

- To target a wide range of applications, the arithmetic optimization is undertaken at the algorithmic level, and not at the architectural level;
- Because only LTI and LTV controllers are concerned, the main arithmetic functions involved are:
  - Single and Multiple Coefficient Multiplication (SCM and MCM);
  - Variable operand multiplication;
  - Multi-precision multiplication.
- As the optimization effort is deployed at algorithmic level, there is a need to explore a large number of alternative algorithms in order to select the best one. Therefore, reconfigurable circuits (FPGA) stand in this case as the most appropriate option of integration. FPGA allows a fast prototyping;
- Although FPGA serves for the validation of the developed algorithms, HDL code must be fully compliant to the standard IP design-reuse methodology. This means also that the HDL-RTL code will be 100% technology independent, offering the possibility to be mapped both on FPGA and ASIC using a digital standard-cell-library of a given technology foundry. This measure guarantees a systematic reuse of the same code for any future ASIC integration.
- Arithmetic optimization results are applied to PID and LQG controllers with Kalman's filter. These two linear controllers were used in a previous work [3] to control the FTG-100 microgripper (www.femtools.com) on a dSPACE platform.

1.4 Contribution of the Thesis

We have investigated and compared four binary arithmetics: CSD, DBNS, RNS, and Radix-2^r. In spite of a serious limitation of Radix-2^r for high values of r due to the generation of an important number of odd-multiples $\{1,3,5,\cdots,2^{r-1}-1\}$, it seems relatively the most liable arithmetic to fulfil the above-set requirements. We have circumvented the limitation of Radix-2^r by a recursive construction of higher radices based on a combination of lower radices. This technique has enabled to develop:

- A new heuristic (RADIX-2^r) for SCM. It has the major advantage of being 100% predictable in maximum number of additions (upper-bound), in average number of additions, and in number of cascaded adders (adder-depth) forming the critical path. Furthermore, its computational complexity is linearly proportional to the constant bit size (N), which means that it has no limitation with regard to N. Despite the big number of existing heuristics, none of them is predictable, not even partially. Besides, most of them exhibit a polynomial runtime complexity $O(N^{\alpha})$ with $\alpha \geq 3$, and some of them show even an exponential $O(2^N)$ complexity. Only CSD is linear and predictable. But the latter is largely superseded by RADIX-2^r which requires 18% less additions for a 64-bit constant.
- A new fully predictable heuristic (RADIX-2^r) for MCM with a computational complexity $O(N \times M)$, where M is the number of constants. Compared to the standard CSD for (N,M)=(64,10), a saving of 53% is obtained. The saving increases as the product $N \times M$ increases.

- A series of new high-radix multiplication algorithms with variable operands. They have the merit of being fast, energy efficient, highly-scalable, and predictable. Contrary to existing Radix-2^r algorithms where the highest value of r is limited to 16, the proposed algorithms have no limitation with regard to r. Theoretically, the higher the radix, the shorter the critical path (faster).
- A new mathematical formalism which enables to derive the optimal high-radix multiplication algorithm from a given combination of low-radix multiplication algorithms. The idea has been physically validated on FPGA.
- A series of new multi-precision multiplication algorithms, offering the possibility to run simultaneously several small-size multiplications on the same $N \times N$-bit multiplication array. The proposed algorithms have the superiority over exiting ones to support a generic partitioning of the array. This results in a higher computational throughput with reduced power consumption.
- The currently best known analytically-proved bounds (Upper-bound, Adder-depth, Average) with the exact number of additions for SCM and MCM.

1.5 Organization of the Thesis

Except the introduction and conclusion, the remainder of the manuscript is structured in five chapters.

Because of the intended applications, corresponding informations are given in chapter 2. We first start with micromanipulation as a MEMS application. We describe its specificities and requirements. This is followed by the state-of-the-art of embedded control-electronics for MEMS. We review the basic digital solutions and discuss the pros and cons of each option. At the end, we present an overview of the different optimization techniques applied to FWL controllers.

In chapter 3 we deal with the binary arithmetic. First, the mostly used number formats, fixed-point and floating-point, are introduced and compared to one another with regard to the precision and dynamic range. We then introduce the most commonly used number representation systems. We insist more particularly on the two arithmetic operations (+, ×) required by linear systems.

Chapter 4 is devoted to the operation of multiplication by a constant. A range of the most frequently cited algorithms are presented, followed by the introduction of a new heuristic called RADIX-2^r. A detailed description of the latter is given, accompanied with a discussion on the experimental results.

The same is done in chapter 5 for the variable-operand multiplication. We first describe the most advanced high-radix multiplication algorithms, and then we introduce a series of new high-radix algorithms and show how to extract the optimal one. Next, we discuss the experimental results. The same presentation scheme is applied to multi-precision multiplication.

In chapter 6, we apply the results of research developed in the previous chapters on the PID and LQG controllers with Kalman's filter.

Finally, we provide some concluding remarks on the accomplished work. We comment its strengths and weaknesses, and propose a roadmap for the continuation of the project of embedded electronics for MEMS.

Bibliography

[1] E. Manmasson et *al*., “FPGA in Industrial Control Applications,” IEEE Trans. on Industrial Informatics Journal, vol. 7, N° 2, pp. 224-243, May 2011.

[2] R. Casanova et al, “Integration of the Control Electronics for a mm^3-sized Autonomous Microrobot into a Single Chip,” Proceedings of the IEEE International Conference on Robotics and Automation (ICRA’09), pp. 3007-3012, Kobe, Japan, May 12-17, 2009.

[3] M. Boudaoud, "Commande en Effort d'une Micropince en Actionnement Electrostatique," Master Thesis, AS2M Department, FEMTO-ST, Besançon, July 2009.

Chapter 2

Problem Background

Chapter 2

Problem Background

In this chapter, we first define the application context of the thesis and then give the state-of-the-art of each topic involved. We describe the main characteristics of the micromanipulation domain as a MEMS application, followed by a survey of hardware integration solutions adapted to the embedded control. We examine the solution space with regard to the MEMS requirements and select, wherever possible, the best option that leads to an optimized implementation of the controller. The selection is carried out depending on the control-bandwidth, the power consumption, and the precision of calculation.

2.1 Application Context

MEMS (Micro-Electro-Mechanical Systems) technology dates back to the discovery of the piezoresistive effect in silicon and germanium at Bell Labs in early 1950s [1]. It combines lithographically formed mechanical-structures with electrical elements to create physical systems that operate on the scale of microns. A MEMS is defined as a system of micrometric dimension (less than 100 µm) incorporating at least two of the following features: sensor, signal processing electronics, actuator, power or transmission. Over the last 60 years, an impressive variety of MEMS devices were developed, constantly pushing the boundaries of physics, mechanics, electronics, and control theory.

Microrobotics is one of the MEMS applications. It is the field of miniature robotics, in particular mobile robots with characteristic dimensions less than 1mm. Microrobots [2] and Micromanipulators [3] are two outstanding examples of MEMS devices, dedicated to work in the micro-world. At this scale, the systems should have accuracy and resolution that are better than one micron.

The AS2M department of FEMTO-ST is specialized in micromanipulation/microassembly applications. For their setups, researchers use bulky equipments (typically standard PC and additional electronic cards) to control the developed micromanipulators. This constraining environment makes any mobile utilisation difficult. One of the smallest components of the micromanipulator is the end-effector. In most cases, this is a micro-gripper. It becomes really interesting if the microgripper can be easily changed for the adaptation to a new task. Therefore, the solution is to integrate the control electronics within the MEMS part to obtain some sort of a "plug-and-play" micro-gripper. However, the implementation of this solution requires advanced skills in hardware integration. For such a purpose, a collaboration project between FEMTO-ST and CDTA has been established in 2010. It aims at exploring and proposing appropriate solutions for the hardware integration of the control part of the micro-gripper. The present thesis is a part of this collaboration project.

However, controlling structures at the scale of micron is not only challenging from the modeling and control-law point-of-view, but also *computationally* challenging [4], mainly for three reasons:

- These miniature devices are capable of extremely fast movements, requiring very high control bandwidth to ensure their stability. For instance, in [3] a micro-gripper with two degrees of freedom piezocantilevers is proposed (Fig. 2.1a). It is dedicated to applications where both

high performance and high dexterity are required. Such a micro-gripper presents a strong coupling between the two axes. In addition, it is very oscillating and strongly nonlinear (hysteresis and creep). All these problems are compensated using additional control/filtering techniques that require more computational time. Thus, the stability is guaranteed only if enough computational power is provided. To give an order of magnitude, the dynamic of the piezocantilever is typically controlled with a sampling time of 20µs (Fig. 2.1b).

- With small physical sizes and high resolution come low tolerance to error, and therefore a need for a high computational control precision. For instance, in micromanipulation and microassembly, fixing a micro-lens at the tip of an optical fiber with 1µm of relative positioning error or 0.4µrad of orientation error, may cause a loss of 50% of the light flux [3].
- The reality is that most of the MEMS devices are embedded in autonomous equipments which are bound by power and size constraints. This dictates the use of power-efficient solutions in control and in the implementation of the controller as well. A typical example is the mm^3 robots [2], called I-SWARM. It is a real autonomous mobile micro-robot; powered by solar cells, equipped with a locomotion unit for moving, an infra-red unit for communication, and a contact tip for detecting near objects. The whole is managed by a *low-power* ASIC, which is in fact a real system-on-chip (SoC) solution.

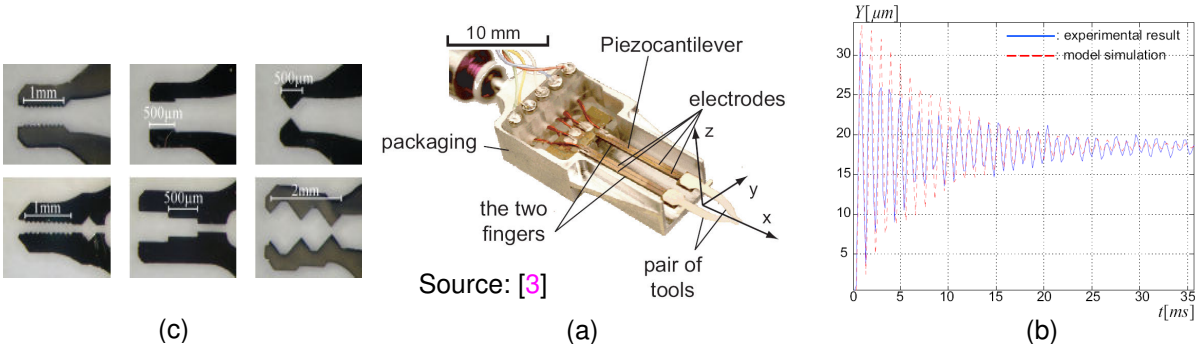


FIGURE 2.1 – (a) Microgripper with two degrees of freedom piezocantilevers.
(b) Step response of the piezocantillever.
(c) Terminal parts of the microgripper.

Consequently, in addition to an advanced modeling and control, *high-speed*, *low-power*, and *high precision of calculation* are mandatory for the MEMS control. In the case of an embedded control, which is the purpose of this thesis, the precision of calculation becomes even a more severe constraint, harder to satisfy without compromising speed and power. This is essentially due to the utilization of fixed-point numbers which are an *approximation* of real numbers. The fixed-point format is dictated in embedded applications for its high-speed and low-power features.

Obviously, this thesis does not have the pretention to address all issues related to embedded control for MEMS. Such a complex and substantial problem would require a series of complementary works. The objective is rather restricted only to the determination of a suitable binary arithmetic that conciliates into a good compromise speed, power, and precision, while preserving satisfactory control performances.

Because micromanipulation is the MEMS application domain to which the proposed arithmetic is applied, a brief summary of its characteristics is given hereafter for a better understanding of the real *control* challenges.

2.2 Micromanipulation as a MEMS application

Micromanipluation addresses the problem of gripping, handling, moving, and placing objects of micrometric sizes. During the last decade, the need for micromanipulation systems (Fig. 2.1) with micro/nanometers accuracy and fast dynamics has been growing rapidly [5]. Such systems occur in applications including:

- Manipulation of biological elements in medicine and biotechnology (micro-organisms, cells, DNA, etc.).;
- Assembly of micromechanical rigid parts (micro sprocket wheels, optical micro lens, hybrid circuits, etc.);
- Micro/ Nano-sensors for environnemental monitoring;
- Metrology and nanometer resolution imaging (AFM and SEM);
- Study of micro-world phenomena, such as adhesion forces.

Before exposing the major problems pertaining to the microworld, it is useful to have a precise idea on the order of magnitude in size and weight of the physical entities manipulated (Fig. 2.2 and 2.3).

Force and position measurements are very important to perform micromanipulation and micro-assembly tasks. Small components are often fragile (e.g. biological cells) and may be damaged or destroyed if they are grasped without force *control*.

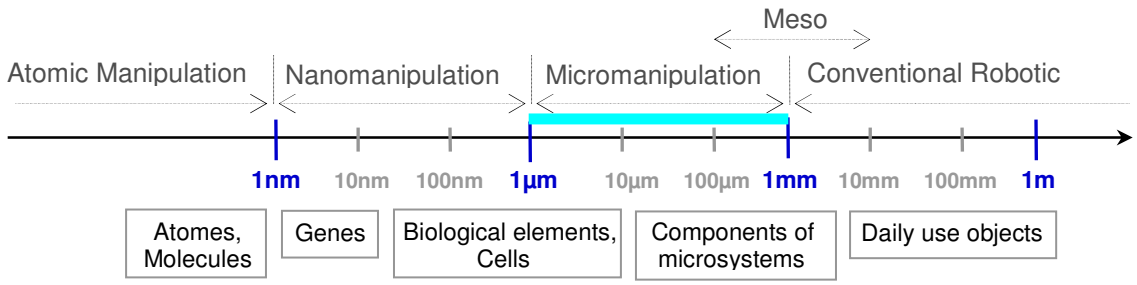


FIGURE 2.2 – Sizes and dimensions characterizing the microworld.

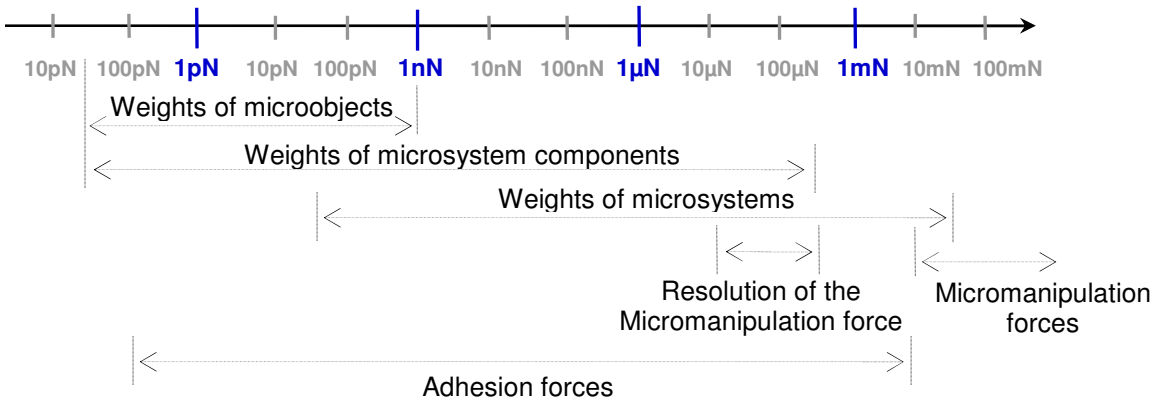


FIGURE 2.3 – Some force spans characterizing the microworld.

Working in the micro/nano-world involves displacements from nanometers to tens of microns [6]. Because of this precision requirement, environmental conditions could generate noise and disturbance that are in the same range as the desired displacements (very low signal/noise ratio). These environmental conditions include: thermal variation; vibration; air-flow; and humidity.

In fact the forces that come into play at the micro level are different from those at the macro level. This is due to what is commonly called the scale effect [7]. Indeed, when the size becomes small enough, the surface effects become dominant over the volume effects. Gravity, which is often relied on in control and assembly of macro systems, plays only a minimal role at the micro level. Instead, the dominant forces are (Fig. 2.4):

- Electrostatic forces (F_{elec}) generated by tribo-electrification and charge transfer during the contact;
- Surface-tension forces (F_{tens}) between the two contacting elements, related to the level of humidity;
- Van der Waal's forces (F_{Vdw} , atomic forces).

There is also another kind of problems related to uncertainties. It is interesting to note that because of adhesion forces, that still remain difficult to control, the equations of the dynamics and kinematics of the micro-objects are subject to uncertainties, so that the movements are unpredictable. The uncertainties are also due to the limited performances of the sensors. To handle a micro-object, the required accuracy is the tenth of the size of the micro-object in the worst case, and the resolution is equal to 1/n times of the precision (n>1). At present, propose force and position sensors with such a performance and a suitable size remains a challenge [8]. Precise position sensors already exist, such as interferometers, but their size is not suitable (bulky).

Another problem of the scale-effect is the inability to directly use the human sight sense in the microworld. A microscope-camera system is often used to monitor micromanipulation tasks. Fully automated or remotely operated, these systems are equipped with a micromanipulation screen, enabling the human operator to intervene [9].

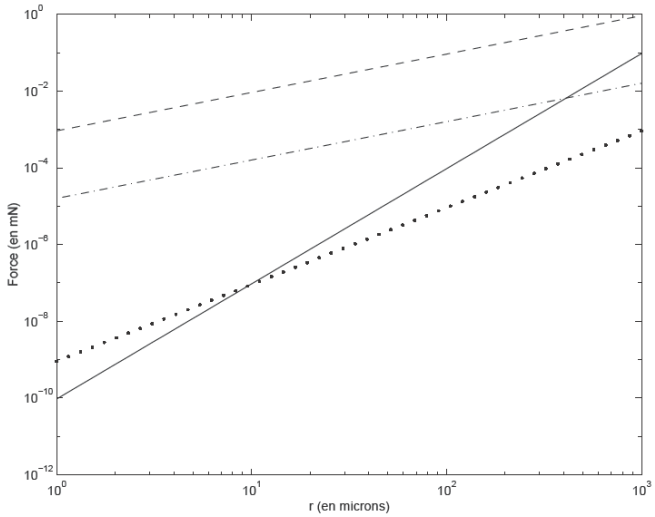


FIGURE 2.4 – Amplitude of the forces in the microworld for a microsphere of radius r (F_g: – ; F_{tens} : - - ; F_{Vdw} : -.- ; F_{elec} : ...).

The specific complexities of the microworld (adhesive forces) added to the lack of appropriate sensors is a real challenge that is far from being mastered [6]. The capture and release by a microgripper of "sticky" parts (Fig. 2.5) is a hard problematic issue [6][7][10][11][12].

Because of all these difficulties, the design of micromanipulation systems (Microgrippers) must offer the best performance in terms of accuracy and resolution. As to their use, it must reproduce as

accurately as possible the user commands. In order to respond to this double requirement, there is an increasing resort to use:

- Smart materials such as piezoceramics [5][6][13], magnetostrictive, shape memory, electroactive polymer, are used to develop microactuators, microrobots, micromanipulators and Microsystems;
- Robust control techniques (both in theory and implementation) based on closed-loop feedback [14][15] combined with advanced filtering techniques [8].

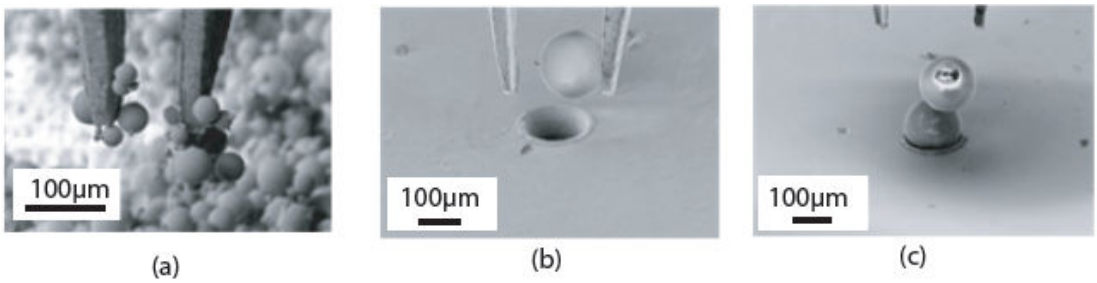


FIGURE 2.5 – Micromanipulation problem due to adhesion forces. (a): Difficulty to grip micro-objets; (b): Problems in placing a micro-object; (c): difficulty of positioning a micro-object due to adhesion forces created by surrounding micro-objects.

2.3 Embedded Control-Electronics for MEMS

The literature on the control issue of MEMS/NEMS systems is so extensive, but all proposed solutions fall into two distinct categories:

- Analog controllers: that is, implementing the controller as an active analog feedback circuit. Used to be the primary method of implementing controllers before digital systems of sufficient performance became widely and cheaply available. This type of controllers presents numerous challenges since analog systems are more difficult to design (requiring careful control of all active components), implement (especially in modern low-cost processes optimized for digital systems), and maintain (there is not possibility of "patching" the system). Furthermore, the physical complexity of modern mechanical devices may require modal control that cannot be implemented by a single analog controller [4].
- Digital controllers: due to the above shortcomings of analog controllers and aided by the fulgurous progress in semiconductor technology (transistor size shrinking) [16], tightly integrated solutions that combine high-performance power of digital processing with accurate sensors are used (*closed-loop control*).

To provide stable and robust control, a digital control system must be able to measure the process variables and set actuator output command within a fixed period of time (loop-cycle time). However, recent experience of MEMS/NEMS developers [4] has shown that the *state-space representation* [17] [18] is being extensively applied across a wide variety of MEMS devices for two reasons:

- Its relative ease in determining stable control equations;
- It forms the basis for many more complex control techniques;
- Its recursive behaviour is suitable for a digital implementation.

Hereafter, are the discrete state-space equations for LTI and LTV, respectively:

Explicit discrete time-invariant
$$\begin{aligned} x(k+1) &= A\cdot x(k)+B\cdot u(k) \\ y(k) &= C\cdot x(k)+D\cdot u(k) \end{aligned} \tag{2.1}$$

Explicit discrete time-variant
$$\begin{aligned} x(k+1) &= A(k)\cdot x(k)+B(k)\cdot u(k) \\ y(k) &= C(k)\cdot x(k)+D(k)\cdot u(k) \end{aligned} \tag{2.2}$$

where:

$x(\cdot)$ is called the "state vector", $x(t)\in \Re^n$;

$y(\cdot)$ is called the "output vector", $y(t)\in \Re^q$;

$u(\cdot)$ is called the "input (or control) vector", $u(t)\in \Re^p$;

$A(\cdot)$ is the "state (or system) matrix", $dim[A(\cdot)]=n\times n$;

$B(\cdot)$ is the "input matrix", $dim[B(\cdot)]=n\times p$;

$C(\cdot)$ is the "output matrix", $dim[C(\cdot)]=q\times n$;

$D(\cdot)$ is the "feedthrough (or feedforward) matrix", $dim[D(\cdot)]=q\times p$..

A depth in-sight into state-space equations reveals that the computation pattern is mainly based upon:

- *Matrix Multiplications* (*MMs*), which involve the use of the elementary but time-critical multiply-and-accumulate (*MAC*) operation [19][20] in the case of LTV systems, and SCM/MCM in the case of LTI systems;
- And a limited movement of data between temporary storage elements (load/store of data from/in memory and registers).

The computational overhead is maximal when digital controller is performing these operations, more especially when it is performing MMs. Hence the sampling-rate (*performance*) is limited by the rate at which the device performs these computations. However, the high potential of parallelism inherent in MM suggests that significant performance improvements can be achieved [21][22] by providing increased architectural parallelism.

The number of elementary scalar operations (MAC, SCM/MCM) involved depends on the *control complexity*, based on:

- The number of state variables that fixes the dimension of the matrices;
- The number of I/O required;
- And, the density of the matrices (the use of sparse and insensitive matrices wherever possible considerably reduces the computation complexity);

With time constants several orders of magnitude faster than their non-MEMS counterparts, many MEMS devices require *control-bandwidth* and accuracy exceeding the ability of conventional digital solutions [4]. On the other side, for autonomy purposes and size constraints [2], many MEMS devices dictate the use of *low-power* consumption solutions. There is a trade-off between performance and power consumption that is quite critical.

The important question that arises at this stage is: with a plethora of available solutions for digital embedded control [23][24][25][26][27][28], which one fits the best MEMS application requirements?

This question can be quickly answered for the most part by looking to the pros and cons of existing solutions with respect to the above-mentioned severe antagonistic constraints (high performance & low power consumption), but the ultimate solution must be tightly tailored to the application case according to the intended objectives.

2.4 Review of the Basic Digital Solutions for Embedded Control

The current state-of-the-art [24][16] of embedded solutions offers mainly four possibilities based on the use of:

- Commercial Off-The-Shelf (COTS) electronics components. This includes general purpose microprocessors and microcontrollers;
- Digital-Signal-Processors (DSPs) such as the TMS320C55x [29] and TMS320C64x [30], which are respectively the lowest power and the highest performance DSP available [15];
- Field-Programmable-Gate-Array (FPGAs) such as Xilinx's [31] and Altera's [32] FPGAs;
- Application-Specific-Integrated-Circuits (ASICs) which are chips designed from scratch according to a given technology foundry (e.g. TSMC 90 nm) .

Let us examine in detail the pros and cons of each solution, focusing more particularly on the MEMS control requirements.

2.4.1 Commercial Off-The-Shelf (COTS) Electronics Components

The common measure of control system performance and robustness is *jitter*, which is a measure of the variation of the actual loop cycle-time from the desired loop cycle-time. However, the *nondeterminism* in the execution of microprocessors makes static timing boundaries difficult to determine and limits reliable bandwidth. This is mainly due to three reasons:

- In a general purpose microprocessor, the processor resources are held up while it is busy performing the MAC operations and the speed or the sampling rate is decided by the latency of these instructions;
- Additionally, priority interrupts and bus contention contribute to the nondeterminism encountered in typical microprocessors or microcontrollers [24][27].
- Worse enough, in general-purpose operating systems (OS) such as Windows (Ex: PC based control platform) where the microprocessor is programmed to handle multiple asynchronous tasks, the jitter becomes exaggeratedly unbounded (*random*) so closed-loop control system stability cannot be guaranteed [33]. Processor-based control systems with real-time operating systems (RTOS) are commonly able to guarantee control-loop jitter of less than 100 microseconds.

2.4.2 Digital Signal Processors (DSPs)

DSP are mainly differentiated from general purpose microprocessors by additional parallelism [24][25]. For instance, the TMS320C64x is a VLIW architecture with eight execution units, including four multipliers and four ALUs [30]. Using its eight execution units, the processor can execute up to

eight 32-bit instructions in a single clock cycle (up to 1GHz), allowing it to achieve a high level of parallelism. The TMS320C64x is able to perform four 16-bit multiplications in parallel. All execution units in the TMS320C64x have a throughput of one cycle and latencies from one to several cycles depending on the instruction.

To date, for sound/image/video applications, conventional DSPs have provided more than adequate bandwidth, with consistent performance gains from feature scaling [16] and added architectural parallelism. However, for this new class of control applications (MEMS), the *latencies* that are inherent to many DSP architectures limit the achievable control bandwidth which refers to the throughput of the controller as each state-space time step is dependent on the last. The latency becomes then a critical aspect. Further, MEMS systems, by nature, are small in size and complexity, resulting in simple control computations that must be performed at very high rates [34]. For small control systems, the benefits of added parallelism in high performance DSPs fails to mitigate the complexity and performance overhead of the architecture. In other words, DSP architecture is oversized for MEMS applications.

Besides, as control complexity scales, the computational advantage is outweighed by the cost of moving data between local register files and main memory, which induces an exponential fall off in control bandwidth [25]. It is important to note that computation complexity grows quadratically with *control complexity* due to the n^2 multiplications in a matrix multiply. This has also a negative effect on power consumption which nearly reaches 1 amp in the case of TMS320C64x, making it impractical for use in small embedded systems.

For MEMS control requirements, a good DSP evaluation study based on control bandwidth / power consumption versus control complexity is given in [4].

2.4.3 Field Programmable Gate Arrays (FPGA)

FPGAs belong to the class of user programmable digital devices called Programmable Logic Devices (PLDs, in contrast to ASICs). A PLD is an integrated circuit that enables the user to configure it in many ways, enabling the implementation of various digital logic functions, of varying sizes and complexities. For instance, Xilinx's XC4VSX55 FPGA comprises an array of 128x48 Configurable Logic Blocs (CLBs), 512 18bit-MACs, 320 18Kb-Block-RAMs and 640 I/Os [31]. Some FPGAs, like XC4VFX family, include up to 2 PowerPC Processors, which are real System-on-Chip (SoC) platforms.

One of the fundamental differences between FPGAs and DSPs is the number of MACs included [25], which is respectively 512 for XC4VSX55 FPGA and only 4 for TMS320C64 DSP [28]. In the case of 512 tap filter computation, 512 MAC operations per simple are involved. So what takes 128 (512/4) clock cycles for DSP can be completed in a single clock cycle for FPGA. In fact the reality is lower than this, as data has to be pulled in and out of memories; a number of additional clock cycles are required either for FPGA or DSP. Nevertheless, this example shows the most important feature of FPGA parallelism.

Because of the inherent parallelism of FPGA architecture, many independent control loops can run at different deterministic rates without relying on shared resources that might slow down their responsiveness as in the case of COTS solutions. Hence, the jitter for FPGA-based control loops depends only on the accuracy of the FPGA clock source. It typically ranges in the order of

picoseconds. Furthermore, since there are enough resources for parallelization, the control loop rate is limited only by the sensors, actuators, and I/O modules [24]. This is an outstanding contrast to COTS and DSPs control systems, where the processing performance was typically the limiting factor.

As comparative example, the Proportional-Integral-Derivative (PID) control algorithm, commonly used for regulating analog processes such as temperature and pressure, executes in just 300 nanoseconds (3.33 MHz) on FPGA (according to 2006 benchmarks), whereas it executes at the rate of 2.75 KHz on typical COTS processor [24].

Recent trends in FPGA based control are:

- Dynamic reconfigurability: In the recent years, the specifications for control systems have grown to include a certain degree of *intelligence*. They vary from specifications requiring certain amount of fault tolerance to operating under varying operating conditions [35]. These systems must also be capable of intelligent sensor selection, remote monitoring and operation and must be capable of implementing sophisticated control algorithms that require adaptation. By systematically partitioning the system; functionality requiring large amounts of reconfiguration can be given such kind of resources on an FPGA; thereby ensuring that the above mentioned objectives are met. This is especially useful in certain kinds of fault tolerant systems. Suppose the system detects the occurrence of a fault; then a new configuration can be loaded (either partially or fully) so that the fault is taken care of (either remedied or bypassed) and the control system performance is not affected.
- Hardware/software Co-design: this is an evolving aspect of FPGA based control. An application of this particular approach in the area of Model Predictive Control (MPC) is illustrated in [36]. Hardware software co-design is a new paradigm in which a microprocessor/microcontroller is embedded in an FPGA. Control algorithms that require a large number of computationally involved operations like matrix manipulations cannot be effectively implemented on a single microprocessor based set up, as the microprocessor gets bogged down while performing these operations. It is in this regard that the parallel architecture of the FPGA can be exploited to develop a matrix coprocessor for performing these computations; while the general purpose microprocessor that was embedded in the chip can be used to perform other operations. This is more efficient and still retains its system on chip nature due to the fact that the processor and the FPGA come together, bundled on single chip.

2.4.4 Application Specific Integrated Circuits (ASIC)

Today, the reality is that control systems for MEMS are bound by three antagonistic constraints: power, size and performance. Contrary to DSPs and FPGAs, ASIC is the unique solution capable to cope with such a difficulty.

By rethinking the architectural choices (efficiently separated and optimized decision part and data-path, both tightly tailored to the application case), it is possible to create control systems with reasonable power margins, negligible area overhead and adequate bandwidth. This can be well illustrated by two recent practical ASIC cases:

- A general purpose solution in [4];
- And a specific purpose solution in [2].

In [4], a general purpose scalable closed-loop-feedback control architecture was developed, based on 90 nm CMOS TSMC technology. This ASIC solution provides higher bandwidth and lower power than respectively state-of-the-art DSP TMS320C64x and TMS320C55x. The maximum clock frequency of the architecture is 1.2 GHz and is limited by a latency of 6-stage data-path. It shows potential for a wide range of applications, supporting control bandwidths as high as 40 MHz in small systems and well over 200 KHz in large control systems whereas none of TMS320C64x family is capable of 100 KHz bandwidth. With 16 MACs included, the chip size does not exceed 1mm^2 and requires less power than TMS320C64x. And with 8 MACs included, it requires less than more than 50% power than TMS320C55x.

In [2], an ASIC specifically dedicated for a real autonomous microrobot is described. The ASIC is a real SoC as it incorporates on the same silicon all necessary electronics required by an autonomous device: power electronics, buffers, ADCs, DACs, control and data-path units, analog transducers and an oscillator. As the microrobot is powered by solar cells delivering a limited energy of 1mW@1.4V and 500µW@3.6V, special care has been devoted to power consumption, such as:

- Use of 0.13 µm ultra *low leakage* SiGe CMOS technology;
- Use of ultra low leakage SRAM memories;
- Incorporate wherever possible low power *design techniques* such as:
  - clock gating: disable those circuits that are not operative;
  - power gating: turn off those modules that are not operative;
  - dynamic frequency scaling (DFS) which adjusts the clock frequency of the processor as a function of the workload;
- Use of *event driven* mode to woke up the processor once one of the peripherals finishes its task.

The whole ASIC electronic is mapped onto monolithic silicon surface of 3x3 mm^2, which can be rated at a maximum clock frequency of 12 MHz. The total leakage power consumption can be managed between 300 to 700 µW approximately.

Through these two recent illustrative examples, it's made clear that ASIC solution is the approach that by far offers better results in terms of speed, area and power than non-ASICs solutions (Table 2.1). However, the main drawback is that the chip has to be designed from *scratch* with its inherent risks, developing time and higher costs. As an *intermediate* solution, FPGA can be effectively used for

TABLE 2.1 – Main features of the basic digital solutions for embedded control.

Technology	Jitter[1]	Time Parallelism	speed	Power Consumption	Developing Cost	Developing time	Required Skills	Portability
COTS	Yes	C[2]	D	D	A	A	SP	D
DSP	No	B	C	C	B	B	ASP	C
FPGA	No	A	B	B	C	C	HD	B
ASIC	No	A	A	A	D	D	AHD	A

A–D: A, the best; D, the worst. 1: Jitter is a measure of variation between the actual loop cycle-time and the desired loop cycle-time. It is caused by the non-determinism of the execution time (Section 2.4.1). 2: Pseudo-parallelism (time sharing). SP: Software Programming. ASP: Advanced Software Programming. HD: Hardware Design. AHD: Advanced Hardware Design.

its shorter *design-cycle time* as fast *prototyping* device in order to get the control algorithm fine tuned and validated even if the ultimate objective is the creation of an ASIC.

So far, we have shown that ASIC and FPGA are the most appropriate mediums for MEMS control, independently of the effective implementation that requires important optimizations involving both control theory and binary arithmetic. This point is treated in the next section.

2.5 Overview of Finite-Word-Length (FWL) Controller Optimizations

The objective is twofold: we need to achieve an optimal ASIC/FPGA implementation of the controller without degrading control performances. To reach such a goal, a double expertise is required in hardware design and control system. But usually, hardware designers do not master control system design, and control system experts do not have the required skills to implement and evaluate the controllers using ASIC/FPGAs [37][38]. The only promising solution is a complete automation of the whole FWL-design-flow, starting from the performance specifications of the controller up to the generation of a synthesizable HDL code (VHDL or Verilog). Such a consistent work is being undertaken within a French ANR project called DEFIS (Design of Fixed-Point Embedded Systems) [39].

2.5.1 Definition of the FWL Effect

To satisfy the constraints (area, energy consumption, execution time) inherent to embedded systems, fixed-point arithmetic is widely used and preferred. However, Fixed-point arithmetic is employed as an approximation of real numbers (floating-point), with a fixed bit-length of the word used to represent data. This limitation leads to performance degradation (FWL effect) mainly due to:

- Quantization of coefficients (parametric errors) which is achieved by first rounding each coefficient to the nearest value based on the available word length before implementation of the system.
- And roundoff errors (signal quantization) which consists of rounding every internal signal at each time instant k to the nearest available quantization level. Besides, addition operations and especially multiplication operations produce results that require a longer word length for accurate representation. Each result is rounded in some manner at the least-significant bit and truncated or limited at the most-significant bit. The subsequently cumulated error during the computation process is assimilated to a numeric noise.

If any internal signal exceeds the available dynamic range, then arithmetic overflow occurs. The usual practice is then to employ saturation arithmetic whereby the particular offending variable is set to the maximum allowable magnitude. Overflow can cause significant distortion and/or instability such as signal clipping and limit-cycles. Thus, the internal signals must be scaled so as to appropriately restrict the occurrence of overflow.

In fact, the FWL effect is more-or-less exaggerated depending on:

- The structure of the realization used (I/O relationship, levels of parallelism, etc)
- As well as on the way the computations are performed (number of bits, different/unique fixed point position, rounding/truncation, etc).

The floating-point to fixed-point conversion is split into two steps. Firstly, the number of bits for the integer part is determined after the dynamic range evaluation. This number of bits must be minimized while ensuring that no overflow or a limited number of overflows occurs. Secondly, the number of bits for the fractional part is gradually increased till acceptable control performances are obtained. As a result, the limited number of bits used to code the data generates an error (FWL effect) between the infinite precision and finite precision values. This error can be estimated and controlled using some stability measures [40][41][42], such as:

- Deviation of the Transfer function $\left\| H - H^{\dagger} \right\|$;
- Deviation of the pole-zero position $\sum_k w_k (|\lambda_k| - |\lambda_k^{\dagger}|)^2$;
- Deviation with regard to the mould, etc.

In hardware implementation, the word-length defines the size (w) of the data-path in the architecture. Thus, reducing the word-length leads to a more compact structure of the controller. However, more substantial optimizations can be achieved using state-space models based on sparse and insensitive matrices, as illustrated hereafter.

2.5.2 Control-Theory Based Optimization

It is well-known that there exists an infinite set of state-space representations to represent a given LTI controller. These representations are equivalent in infinite precision since they yield the same input-output relationship. However, when a controller is implemented in fixed point arithmetic, it has to be represented with a finite word length (FWL) which leads to a deterioration of the numerical properties of the realization. Hence, the equivalent realizations are no more equivalent in finite precision. One realization may be better suited for implementation than another.

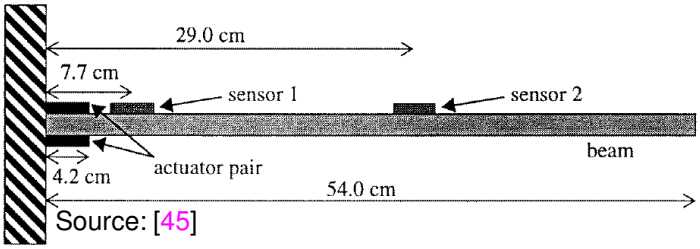
One state-space form of particular interest to hardware implementation is the sparse and insensitive form. The latter has the major advantage of:

- Containing many trivial elements of 0, 1 or -1 which reduces the number of elementary scalar operations (MAC, SCM/MCM). This form is particularly important for high-order controllers (large matrix sizes) as it requires much less hardware and power than the original state-space representation.
- Being least sensitive to quantization effects which allows it to be more coarsely quantized with smaller word lengths than a form that is more sensitive to these effects. Thus, of all possible forms, the insensitive form can be implemented with minimal word length for decreasing hardware requirements.

It is known that canonical controller realisations have sparse structures but may not have the required FWL stability robustness [40]. This posed a complex problem in the past of finding sparse controller realisations with good FWL closed-loop stability characteristics. But because this issue is of utmost importance to embedded systems, extensive studies have been undertaken and many solutions have been proposed [43][44]. The theory behind the determination of sparse and insensitive realisations with stability measure goes beyond the scope of this thesis. Nevertheless, the hardware benefit of using sparse and insensitive matrices is illustrated by the following example given in [45].

Illustrative Example

In [45], the motion of a cantilevered aluminium beam is controlled in a closed loop using an FPGA board. The beam is 54.0 cm long, 1.27 cm wide, and 0.318 cm thick; two PZT elements symmetrically mounted at the beam's base provide bending-moment type actuation 4.2 cm from the clamped end, while PZT elements centered at 7.7 cm and 29.0 cm from the clamped end sense strain. Actuator placement approached the point of maximum strain, while the placement of two sensors resulted in an observable system. Fig. 2.6 summarizes the mechanical configuration of the cantilever beam with one input and two outputs.



Source: [45]

FIGURE 2.6 – Simple cantilever beam.

The discrete time state-space representation of the controller in floating-point notation is given by Eq. 2.3.

$$A_c = \begin{bmatrix} 1.1501 & -0.0778 & 0.1187 & -0.3422 \\ 3.190 & 0.3589 & 0.9999 & -0.9604 \\ 0.0431 & -0.0571 & 1.0283 & 0.2892 \\ 0.4952 & -0.0544 & 0.073 & 0.5605 \end{bmatrix} \quad B_c = \begin{bmatrix} -0.3027 & 0.5503 \\ -0.8866 & 0.6291 \\ -0.0771 & -0.0040 \\ -0.3272 & 0.1025 \end{bmatrix} \quad C_c^T = \begin{bmatrix} -0.0073 \\ -0.0027 \\ -0.5824 \\ -0.5564 \end{bmatrix} \quad D_c = [0 \quad 0] \qquad (2.3)$$

This realization is certainly not unique, nor guaranteed to be problem free or minimal in a real hardware implementation with limited word length. Therefore the scaled balanced real Schur/Givens transform was applied resulting in a sparse and insensitive system. The result is given by Eq. 2.4.

$$\tilde{A}_c = \begin{bmatrix} 0.9373 & 0.1008 & 0.0123 & -0.0089 \\ -0.1263 & 0.7763 & -0.0462 & 0.0308 \\ 0 & 0 & 0.8727 & -0.0001 \\ 0 & 0 & 0 & 0.8628 \end{bmatrix} \quad \tilde{B}_c = \begin{bmatrix} -0.4535 & 0.7103 \\ 0 & 0.5674 \\ 0.0217 & 0.0922 \\ -0.0151 & -0.0628 \end{bmatrix} \quad \tilde{C}_c^T = \begin{bmatrix} -0.5010 \\ 0.5143 \\ 0.0255 \\ -0.0192 \end{bmatrix} \quad \tilde{D}_c = [0 \quad 0] \qquad (2.4)$$

Simulation of the new controller was performed to obtain the minimal word length required while still maintaining closed loop performance. Quantization to eight bits yielded acceptable results. Eq. 2.5 shows the final quantized implementable form using one sign bit and seven fractional bits.

$$[\tilde{A}_c]_w = \begin{bmatrix} 0.9375 & 0.1016 & 0.0156 & -0.0078 \\ -0.1250 & 0.7734 & -0.0469 & 0.0312 \\ 0 & 0 & 0.8750 & 0 \\ 0 & 0 & 0 & 0.8594 \end{bmatrix} \quad [\tilde{B}_c]_w = \begin{bmatrix} -0.4531 & 0.7109 \\ 0 & 0.5703 \\ 0.0234 & 0.0938 \\ -0.0156 & -0.0625 \end{bmatrix} \quad \tilde{C}_c^T = \begin{bmatrix} -0.5010 \\ 0.5143 \\ 0.0255 \\ -0.0192 \end{bmatrix} \qquad (2.5)$$

This simulation accounts for finite-word-length effects of the digital implementation including coefficient quantization, operator noise, and overflow. The resulting implementation after quantization, $\{[\tilde{A}_c]_w, [\tilde{B}_c]_w, [\tilde{C}_c]_w\}$, yielded a sparsity of 7 out of $n^2+np+nq$ =16+8+4=28 for a fully populated form. This yielded a 25% sparsity level.

A 100% full state space realization comprises $n^2+np+nq$ multiplications and $n^2+np+nq-(2n+q)$ additions (based on Eq. 2.1 and 2.2). In digital hardware, adder area scales linearly $O(w)$ with word length (w), and multiplier area scales quadratically $O(w^2)$. Zeros in the state space matrices remove multiply operations altogether, while a shorter word length affects the whole system area. We might be tempted to think that sparsity reduces area more than word length. This is not true. In [45], there is an example which shows the opposite. In any case, the determination of a sparse and insensitive form is the first step in the hardware optimization process of an LTI controller. The second step consists in optimizing the remaining array of multipliers based on the binary arithmetic.

Up to now, we have been dealing with LTI controllers only. As for LTV controllers, the determination of the minimum word length is the only possible optimization, given that all matrices are variable ($A(k)$, $B(k)$, $C(k)$, $D(k)$). But since the array of $n^2+np+nq$ multipliers is constructed using the same multiplier instance, the hardware optimization of the latter leads to a global optimization of the whole array. By optimization, we mean the minimization of the computational delay, logic resources, and power consumption. Because the computational model of LTV state-space involves matrix/vector multiplications mainly, further improvements of the computation latency can be achieved using efficient matrix multiplication architectures [21][22].

2.5.3 Binary-Arithmetic Based Optimization

2.5.3.1 Multiplication by a Constant

In LTI controllers (Eq. 2.1), multiplication by a constant (e.g. $a_{ij}\times x_i$) is the most important scalar operation involved in the computational pattern. To be efficiently handled the implementation must be multiplierless, that is, using exclusively additions, subtractions, and shifts [46]. This is illustrated as follows. In Eq. 2.5, the a_{12} and a_{22} elements of matrix $[\tilde{A}_c]_w$ are equal to 0.1016 and 0.7734, respectively. Their respective representations in 8-bit two's complement notation are:

$$a_{12} = 0.0001101 = 13\times 2^{-7} = (2^3+2^2+1)\times 2^{-7} \quad ; \quad a_{22} = 0.1100011 = 99\times 2^{-7} = (2^6+2^5+2+1)\times 2^{-7} .$$

Hence, according to Eq. 2.1 the multiplication of a_{12} and a_{22} by their common variable x_2 gives:

$$M_1=a_{12}\times x_2=(x_2\times 2^3+x_2\times 2^2+x_2)\times 2^{-7} \quad ; \quad M_2=a_{22}\times x_2=(x_2\times 2^6+x_2\times 2^5+x_2\times 2+x_2)\times 2^{-7} .$$

M_1 and M_2 require 2 and 3 additions, respectively. Thus, a total of 5 additions is needed. The number of additions can be minimized using for instance the exhaustive algorithm MAG [47]. Using MAG, a_{12} and a_{22} are written as follows:

$$a_{12} = [2^4-(2^2-1)]\times 2^{-7} \quad ; \quad a_{22} = (u\times 2^5+u)\times 2^{-7} \text{ with } u=2^2-1.$$

Hence, M_1 and M_2 become:

$$M_1=a_{12}\times x_2=[x_2\times 2^4-(x_2\times 2^2-x_2)]\times 2^{-7} \quad ; \quad M_2=a_{22}\times x_2=(u\times 2^5+u)\times 2^{-7} \text{ with } u=x_2\times 2^2-x_2.$$

The computation order of M_1 and M_2 is well illustrated in Fig. 2.7. Thus, the optimal numbers of additions for M_1 and M_2 are 2 and 2 additions, respectively. With a total of 4 additions, we saved only 1 addition. We assume that addition and subtraction have the same area/speed cost, and that shift is costless since it can be realized without any gates using hard wiring.

No further optimization is possible for a_{12} and a_{22} unless they are considered together, exploiting the fact that they are sharing the same variable (x_2). RAG-n [48] is one of the exhaustive algorithms capable of performing a multiple optimization. Applied simultaneously to a_{12} and a_{22}, it gives:

$$a_{12} = [2^4 - u] \times 2^{-7} \quad \text{and} \quad a_{22} = (u \times 2^5 + u) \times 2^{-7} \quad \text{with} \quad u = 2^2 - 1$$

The combined optimization is illustrated in Fig. 2.8. In this case the total number of additions is 3, achieving a saving of (5-3)/5=40% over the first naive approach.

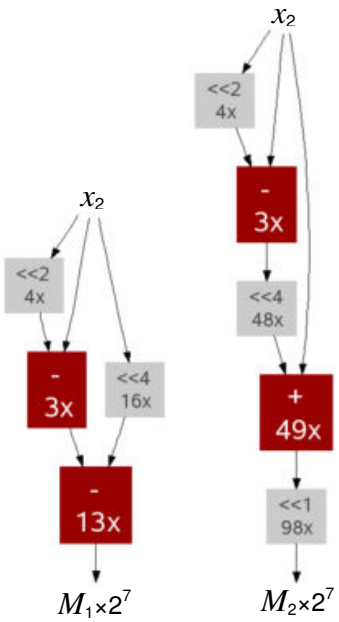


FIGURE2.7 – Minimum number of additions of M_1 and M_2 using a separate optimization of a_{12} and a_{22}.

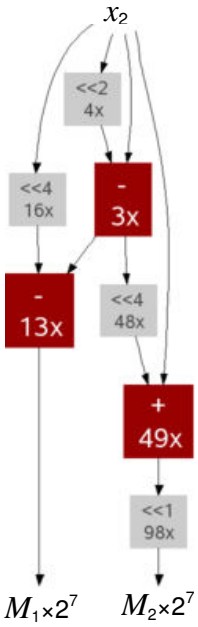


FIGURE 2.8 – Minimum number of additions of M_1 and M_2 using a combined optimization of a_{12} and a_{22}.

Single/Multiple Constant Multiplication (SCM/MCM) is the name given in literature to the problem of performing separate and combined multiplications, respectively. The computational complexity of SCM/MCM is conjectured to be NP-hard. Therefore, since the solution space to explore is so huge, one has to use heuristics. The exhaustive algorithms such as MAG and RAG-n are limited to small constants due to their exponential runtime and huge memory storage needed. Table 2.2 summarizes the state-of-the-art in SCM.

TABLE 2.2 – Main features of the most known SCM algorithms in the literature.

Algorithm				Runtime	Compression Performance	Predictability
Name	Author	Year	Type			
BIGE	Thong [49]	2011	Exhaustive	$O(2^N)$	Optimal	No
H(k)	Dempster [50]	2004	Heuristic	$O(2^N)$	A	No
MAG	Gustafsson [47]	2002	Exhaustive	$O(2^N)$	Optimal	No
–	Bernstein [51]	1986	Heuristic	$O(2^N)$	E	No
Hcub	Voronenko [52]	2007	Heuristic	$O(N^6)$	B	No
BHM	Dempster [53]	1995	Heuristic	$O(N^4)$	C	No
–	Lefèvre [54]	2001	Heuristic	$O(N^3)$	D	No
DBNS	Dimitrov [55]	2007	Heuristic	$O(N)$	F	No
CSD	Avizienis [56]	1961	Heuristic	$O(N)$	G	Yes

A–G: A, the highest ; G, the lowest.

The classification of the heuristics with regard to the compression performance is based on the published results, and remains valid up to 32 bits. Beyond that limit, we have no idea about the behavior of the proposed heuristics since they are not predictable (except CSD).

2.5.3.2 Multiplication by a Variable

As for LTV controllers (Eq. 2.2), variable-operand multiplication is the most important building block. Its optimization ineluctably leads to the optimization of the whole controller.

Although multiplication has been the focus of considerable optimization efforts over the last decades, it still remains a critical problematic issue because of its relatively:

- High signal propagation delay;
- High power dissipation;
- And large area requirement.

The continuous refinement of the mostly-used design paradigm based on modified Booth algorithm [57] combined to a reduction tree (Carry-Save-Adder array, Dadda, HPM, etc) has reached saturation. In [58] and [59] for instance, only slight improvements are achieved. Both proposals reduce the partial product number from $n/2+1$ to $n/2$ using different circuit optimization techniques of the critical path (n is the operand size).

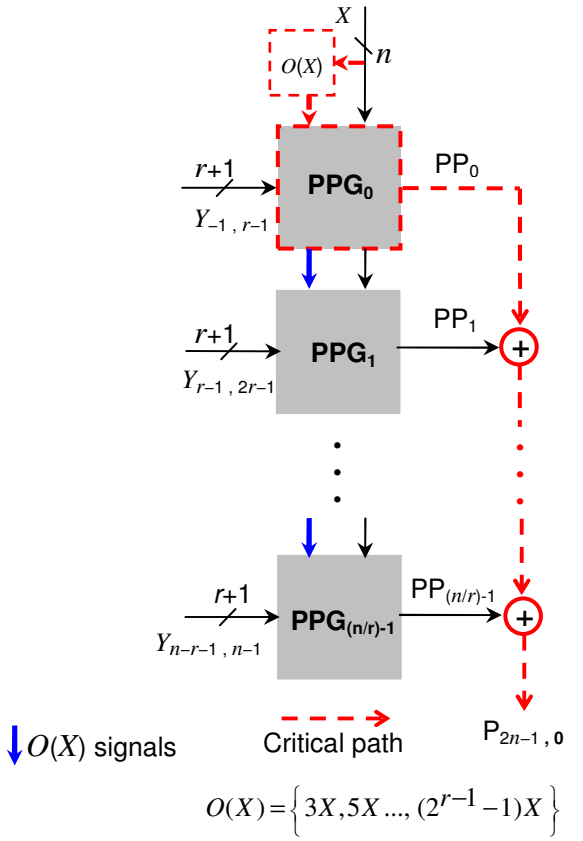


FIGURE 2.9 – Generalized $N \times N$ bit radix-2^r parallel multiplier.
$O(X)$ is the necessary set of odd-multiples corresponding to radix-2^r recoding.

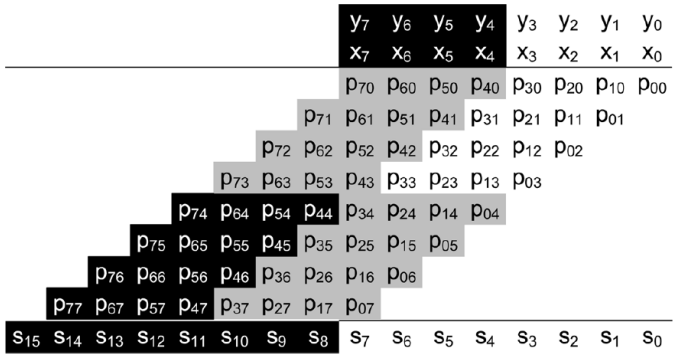
Theoretically, only the signed multibit recoding multiplication algorithm (Fig. 2.9) [60] is capable of a drastic reduction (n/r) of the partial product number, given that r+1 is the number of bits of the multiplier that are simultaneously treated ($1 \leq r \leq N$). Unfortunately, this algorithm requires the pre-computation of a number of odd multiples of the multiplicand (until $(2^{r-1}-1).X$) that scales linearly with r. The large number of odd multiples not only requires a considerable amount of multiplexers to perform the necessary complex recoding into PPG, but dramatically increases the routing density as well. Therefore, a reverse effect occurs that offsets speed and power benefits of the compression factor (n/r). This is the main reason why the multibit recoding algorithm was abandoned. In practice, most of industry commercial designs do not exceed r=3 (radix 8). Only the most recent Intel processors such as Itanium-Poulson [61] use radix-16.

In research, the highest radix algorithms are proposed in the works of Seidel [62] and Dimitrov [63]. Both works rely upon advanced arithmetic to determine minimal number bases that are representatives of the digits resulting from larger multibit recoding. The objective is to eliminate information redundancy inside r+1 bit-length slices for a more compact PPG. This is achievable as long as no, or just very few odd multiples are required.

In [62], Seidel has introduced a secondary recoding of digits issued from an initial multibit recoding for $5 \leq r \leq 16$. The recoding scheme is based on balanced complete residue system. Though it significantly reduces the number of partial products (n/r for $5 \leq r \leq 16$), it requires some odd multiples for $r \geq 8$. While in [63] Dimitrov has proposed a new recoding scheme based on a double base number system for $6 \leq r \leq 11$. The algorithm is limited to unsigned multiplication and requires a larger number of odd multiples though.

2.5.3.3 Multi-Precision Multiplication

When choosing a multiplier for a digital system, the bitwidth of the multiplier is required to be at least as wide as the largest operand of the applications that are to be executed on that digital system. The bitwidth of the multiplier is, therefore, often much larger than the data represented inside the operands, which leads to unnecessarily high power dissipation and unnecessary long delay. This resource waste could partially be remedied by having several multipliers, each with a specific bitwidth, and use the particular multiplier with the smallest bitwidth that is large enough to accommodate the current multiplication. Such a scheme (Fig. 2.10) would assure that a multiplication would be computed on a multiplier that has been optimized in terms of power and delay for that specific bitwidth.



Source: [64]

FIGURE 2.10 – Illustration of an unsigned 8-bit multiplication, where a 4-bit multiplication, shown in white, is computed in parallel with a second 4-bit multiplication, shown in black.

An array of multiplication that can be portioned in several sub-multiplications to be executed in parallel is referred to in literature as a multi-precision multiplication (MPM) array. Such an array is useful in the case of integration of a number of controllers with different bitwidths of the input signals.

A big number of multi-precision arrays have been proposed. They are summarized in [64][65]. Unfortunately, the proposed solutions are either restricted to unsigned multiplication, or they do not take power consumption into consideration, or they are not flexible enough.

2.6 Conclusion

From control implementation issues in microrobotics we have defined the problems related to the integration of controllers for MEMS applications. This would not have been possible without a good understanding of MEMS requirements, namely, a severe control precision, a high control bandwidth, and low power consumption. These requirements have been identified through the study of a typical MEMS application, that is, the micromanipulation. Afterwards, we have explored the different digital technology-solutions suitable to MEMS control. We have shown that ASIC/FPGA stand as the most appropriate options.

Next, a special attention has been devoted to the implementation and especially to the optimization of digital controllers, as it constitutes the core of this thesis. We have shown that the hardware optimization approach depends on the type of the controller, LTI or LTV. In the case of LTI controller, two complementary optimization steps are necessary. Firstly, the determination of the sparse and insensitive form of the state-space model leads to a significant reduction of the logic resources. Secondly, the resulting state-space form is once more optimized using SCM/MCM heuristics, which leads to another substantial reduction in hardware resources. As for LTV controller, the only possible hardware optimization involves the optimization of the multiplier module (MAC) as it is the main building-block of LTV controllers. In addition, a special type of multiplier, called multi-precision multiplier (MPM), has been introduced. It has the merit to considerably reduce power consumption in the case of integration of several controllers.

Among all discussed issues, we have more particularly insisted on the binary arithmetic which is the focal point of this thesis. We have established the state-of-the-art of SCM/MCM, MAC, and MPM. These three items are the key foundations behind any effective contribution to the hardware optimization of linear controllers.

Bibliography

[1] J. Bryzek and E. Abbott, "Control Issues for MEMS," Proceedings of the 42nd edition of the IEEE Conference on Decision and Control, vol. WeES3-1, pp. 3039-3047, Hawai, USA, Dec. 2003.

[2] R. Casanova et al, “Integartion of the Control Electronics for a mm^3-sized Autonomous Microrobot into a Single Chip,” IEEE International Conference on Robotics and Automation, ICRA, pp. 3007-3012, Kobe, Japan, May 12-17, 2009.

[3] M. Rakotondrabe et al., "Robust Feedforward-Feedback Control of a Nonlinear and Oscillating 2-DOF Piezocantilever," IEEE Trans. on Automation, Science, and Engineering, vol. 8, issue 3, pp. 506-519, July 2011.

[4] G. Hoover et al, “Towards Understanding Architectural Tradeoffs in MEMS Closed-Loop Feedback Control,” CASES’07, pp. 95-102, Sep. 30-Oct. 3, 2007, Salzburg, Austria

[5] M. Rakotondrabe, J.T. Wen & P. Lutz, “Control Issues in the Micro/Nano-World,” Introduction of the full day workshop (17 may) of the IEEE International Conference on Robotics and Automation, ICRA, pp. 1-1, Kobe, Japan, May 12-17, 2009.

[6] M. Rakotondrabe, & P. Lutz, “Main Aspects of the Control Issues in the Micro/Nano-World,” IEEE International Conference on Robotics and Automation, ICRA, pp. 3-7, Kobe, Japan, May 12-17, 2009.

[7] B. Borovic & al, “Control Issues for Microelectromechanical Systems,” IEEE Control Systems Magazine, pp. 18-21, April 2006.

[8] Y. Haddab, Q. Chen and P. Lutz, "Improvement of strain gauges micro-forces measurement using Kalman optimal filtering",International Journal of IFAC Mechatronics, vol. 19; N° 4, pp. 457-462, doi:10.1016/j.mechatronics.2008.11.012, 2009.

[9] Micky Rakotondrabe, “Développement et Commande Modulaire d’Une Station de Microassemblage,“ Thèse de Doctorat, Ecole Doctorale Sciences Physiques pour l’Ingénieur et Microtechnique, Nov. 2006.

[10] D.O. Popa & A.N. Das, “Presicion-Driven Hybrid Control for 3D assembly”, IEEE International Conference on Robotics and Automation, ICRA, pp. 17-23, Kobe, Japan, May 12-17, 2009.

[11] J. Wason & J. Wen, “Robust Vision-Guided Multi-Probe Microassembly,” IEEE International Conference on Robotics and Automation, ICRA, pp. 23-25, Kobe, Japan, May 12-17, 2009.

[12] R.A. Maclachlan & &l, “Control of an Active Handheld Instrument for Microsurgery and Micromanipulation,” IEEE International Conference on Robotics and Automation, ICRA, pp. 29-31, Kobe, Japan, May 12-17, 2009.

[13] M. Grossard et al, “Towards the mechanical and control-oriented optimization of micromechatronic systems for robust control”, IEEE International Conference on Robotics and Automation, ICRA, pp. 18-22, Kobe, Japan, May 12-17, 2009.

[14] H. Numasato and M. Tomizuka, “Settling Control and Performance of Dual-Actuator System for Hard Disk Drives,” Amercian Control Conference, pp.2779-2785, 2001.

[15] M. Rakotondrabe, Y. Haddab and P. Lutz, “Quadrilateral modelling and robust control of a nonlinear piezoelectruc cantilever”,IEEE Transaction on Control Systems Technology,Vol.17, to appear 2009.

[16] Reports of the International Technology Roadmap for Semiconductors (ITRS), 2007 & 2008.

[17] K. Ogata, “State-Space Analysis of Control Systems,” Prentice-Hall, 1967.

[18] B. Friedland, “Control System Design: an Introduction to State-Space Methods,” Mc Graw-Hill, 1986.

[19] A.K. Oudjida et al, "Design and Analysis of a High Performance Multiplier and its Generator," Proceedings of The International Conference on Microelectronics ICM'90, pp 2-32-1 : 2-32-14, October 13-17, 1990, Damascus, Syria.

[20] A.K. Oudjida et al, "Design of a High Performance CMOS Adder for Both High Performance Array and Accumulator,". Microelectronics journal, vol. 22 Nos. 5-6, pp 65-73, Elsevier Science Publishers Ltd., 1991, England.

[21] A.K. Oudjida et al,"Mapping Full-Systolic Arrays For Matrix Product On Xilinx’s XC4000(E,EX) FPGAs," The International Journal for Computation and Mathematics in Electrical & Electronics Engineering “COMPEL”, Vol. 21, Issue 1, pp. 69-81, 2002, ISSN 0332-1649, UK.

[22] A.K. Oudjida et al, “N Latency 2N I/O-Dandwidth 2D Array Matrix Multiplication Algorithm," The International Journal for Computation and Mathematics in Electrical & Electronics Engineering “COMPEL”, Vol. 21, Issue 3, pp. 377-392, 2002, ISSN 0332-1649, UK.

[23] W. Marx & V. Aggarwal, “FPGAs are Everywhere: in Design, Test & Control,” The Magazine of Record for the Embedded Computing Industry (RTC Magazine), April 2008.

[24] S. Gretlein et al, “DSPs, Microprocessors and FPGAs in Control,” The Magazine of Record for the Embedded Computing Industry (RTC Magazine), March 2006.

[25] N. Lall, “FPGAs and DSPs: What Makes Sense for your Design?,” The Magazine of Record for the Embedded Computing Industry (RTC Magazine), September 2005.

[26] D. Pellerin et al, “Mixed FPGA/Processor Platforms Accelerate Software Algorithms,” The Magazine of Record for the Embedded Computing Industry (RTC Magazine), March 2004.

[27] K. Parnell & R. Bryner, “Comparing & Contrasting FPGA and Microprocessor System Design and Development,” Xilinx’s White Paper, WP213, V1.1, July 24, 2004.

[28] D. Naylor, “Embedded System Design Considerations,” Xilinx’s White Paper, WP127, V1.0, March 6, 2002.

[29] TMS320C55x DSP Reference Guide. Technical Report, Texas Instruments, Feb 2004.

[30] TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide. Technical Report, Texas Instruments, Feb 2004.

[31] Xilinx Inc., www.Xilinx.com

[32] Altera Inc., www.altera.com

[33] National Instruments Inc., “FPGA Based Control: Millions of Transistors at your Command, 2004. www.ni.com

[34] K. Turner, “Robust Feedback Control Design of an Ultra-Sensitive, High Bandwidth Tunneling Accelerometer,” Proceedings of the American Control Conference, Vol. 6, pp. 4176-4180, June 2005.

[35] R. Muthuraman et al, “Intelligence in Embedded Control: a case study,” Technical and Leadership Workshop, pp. 125-130, April 2004

[36] P. Vouzis et al, “Towards a Co-design Implementation of a System for Model Predictive Control,” Proceeding of the Annual Meeting, American Institute of Chemical Engineerings, Cincinnati Convention Center, OH, November 2005

[37] M. Petko and G. Karpiel, “Semi-automatic implementation of control algorithms in ASIC/FPGA,” Proceedings of Emerging Technologies and Factory Automation Conference (ETFA '03), vol. 1, pp. 427- 433. Sept. 2003.

[38] J. Lima et al, “A Methodology to Design FPGA-based PID Controllers,” Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pp. 2577-2583, Taipei, Taiwan, October 2006.

[39] D. Menard et al., “Design of Fixed-Point Embedded Systems (DEFIS), French ANR Project,” Proceedings of the International Conference on Design and Architecture for Signal and Image Processing (DASIP), Karlsruhe, Germany, Oct. 2012.

[40] M. Gevers and G. Li, “Parametrizations in Control, Estimation and Filtering Probems,” Springer-Verlag, 1993.

[41] T. Hilaire and P. Chevrel, “Sensitivity-based pole and input-output errors of linear filters as indicators of the implementation deterioration in fixed-point context,” EURASIP Journal on Advances in Signal Processing, vol. special issue on Quantization of VLSI Digital Signal Processing Systems, January 2011.

[42] T. Hilairel, “On the transfer Function Error of State-Space Filters in Fixed-Point Context,” IEEE Trans on Circuit & Systems II: Express Briefs, vol. 56, N° 12, pp. 936-940, December 2009.

[43] Y. Feng, P. Chevrel, and T. Hilaire, “A Practical Strategy of an Efficient and Sparse FWL Implementation of LTI Filters,” Proceedings of the European Control Conference (ECC’09), Budapest, Hungary, August 2009.

[44] J. Wu, et al., “Constructing Sparse Realisations of Finite-Precision Digital Controllers Based on Closed-Loop Stability Related Measure,” Proceeding of the IEE Control Theory and Application , vol. 150, N° 1, pp. 61-68, January 2003.

[45] J.S. Kelly et al, “Design and Implementation of Digital Controllers for Smart Structures Using Field Programmable Gate Arrays,” Smart Material Structure Journal, PII: S0964-1726 (97) 87085-1, pp. 559-572, Printed in the UK, 1997.

[46] R. Kastner, A. Hosangadi, and F. Fallah, “Arithmetic Optimization Techniques for Hardware and Software Design,” Cambridge University Press, ISBN-13 978-0-521-88099-2, © 2010.

[47] O. Gustafsson, A.G. Dempster, and L. Wanhammar, “Extended Results for Minimum-Adder Constant Integer Multipliers,” Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), vol. 1, pp. I-73 I-76, Scottsdale Arizona, USA, May 2002.

[48] A.G. Dempster and M.D. Macleod, “Use of Minimum Adder Multiplier Blocks in FIR Digital Filters,” IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing 42, 9, pp. 569-577, 1995.

[49] J. Thong and N. Nicolici, “An optimal and practical approach to single constant multiplication,” IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 9, pp. 1373–1386, September 2011.

[50] A. Dempster and M. Macleod, “Using Signed-Digit Representations to Design Single Integer Multipliers Using Subexpression Elimination,” Proceedings of the IEEE International Symp. on Circuits and Systems (ISCAS), vol. 3, pp. III-165–168, Vancouver, Canada, May 2004.

[51] R.L. Bernstein, “Multiplication by Integer Constant,” Software – Practice and Experience 16, 7, pp. 641-652, 1986.

[52] Y. Voronenko and M. Püschel, “Multiplierless Multiple Constant Multiplication,” ACM Trans. on Algorithms (TALG), vol. 3, No. 2, Artcle 11, pp. 1-38, May 2007.

[53] A.G. Dempster and M.D. Macleod, “Use of Minimum Adder Multiplier Blocks in FIR Digital Filters,” IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing 42, 9, pp. 569-567, 1995.

[54] V. Lefèvre, “Multiplication by an Integer Constant,” INRIA Research Report, No. 4192, Lyon, France, May 2001.

[55] V.S. Dimitrov, L. Imbert, and A. Zakaluzny, “Multiplication by a Constant is Sublinear,” Proceedings of the 18th IEEE Symposium on Computer Arithmetic (ARITH’18), pp. 261-268, June 2007.

[56] A. Avizienis, “Signed-digit number representation for fast parallel arithmetic,”. IRE Trans. on Electronic Computers, vol. EC-10, No. 3, pp. 389–400, September 1961.

[57] O.L. MacSorley, “High-Speed Arithmetic in Binary Computers,” Proceedings of the IRE, Vol. 49(1), pp. 67-91, January 1961.

[58] F. Lamberti, “Reducing the Computation Time in (Short Bit-Width) Two’s Complement Multiplier,” IEEE Trans. on Computers, vol. 60, N° 2, pp. 148-156, February 2011.

[59] S.R. Kuang, J.P. Wang, and C.Y. Guo, “Modified Booth Multipliers with a Regular Partial Product Array,” IEEE Trans. on Circuit and Systems II, Express Brief, vol. 56, N° 5, May 2009.

[60] H. Sam, and A. Gupta, “A Generalized Multibit Recoding of Two’s Complement Binary Numbers and its Proof with Application in Multiplier Implementation,” IEEE Trans. on Computers, vol. 39, N° 8, August 1990.

[61] R.J. Rieldlinger, “A 32 nm 3.1 Billion Transistor 12-Wide-Issue Itanium Processor for Mission-Critical Servers,” Proceedings of IEEE International Solid-State Circuits Conference (ISSCC), pp. 84-86, San Francisco, CA ,USA, February 20-24, 2011.

[62] P.M. Seidel, L. D. McFearin, and D.W. Matula, “Secondary Radix Recodings for Higher Radix Multipliers,” IEEE Trans. on Computers, vol. 54, N°2, February 2005.

[63] V.S. Dimitrov, K.U. Järvinen, and J. adikari, “Area Efficient Multipliers Based on Multiple-Radix Representations,” IEEE Trans. on Computers, vol. 60, N° 2, pp 189-201, February 2011.

[64] M. Själander and P. Larsson-Edefors, “Multiplication Acceleration Through Twin Precision,” IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 17, N° 9, September 2009.

[65] S.R. Kuang, J.P. Wang, “Design of Power-Efficient Configurable Booth Multiplier,” IEEE Trans. on Circuit and Systems I, vol. 57, N° 3, March 2010.

Chapter 3

The Binary Arithmetic

Chapter 3

The Binary Arithmetic

This chapter covers the fundamentals of the binary arithmetic. We first introduce the two mostly used number formats in computer arithmetic: fixed-point and floating-point formats. They are confronted to one another with regard to the provided precision and dynamic range. Afterwards, we review the main number representation systems based upon fixed-point format. These include the canonical-signed-digit representation, the double-base number system, the residue number system, and the radix-2^r number system. In each number system, addition and multiplication functions are carefully investigated as they are the most important arithmetic operations involved in the hardware implementation of linear systems.

3.1 Introduction to the Binary Arithmetic

In binary arithmetic, numbers are represented using two symbols: typically 0 and 1. Thus, binary numbers are seen as a string of 0's and 1's, where the use of a 0/1 symbol corresponds to the dual On/Off position of the basic electronic component (*Transistor*) used to build up complex components. The transistor acts as a switch that either allows a presence of an electrical current (1), or an absence of an electrical current (0). We only use binary because we currently do not have the technology to create "switches" that can reliably hold more than two possible states. Such switches are theoretically possible at a quantum level, but quantum computers are not on sale for the time being.

In mathematical numeral systems, the symbols are called digits, and the number of symbols is called the *base* or *radix*. Binary numbers are expressed in radix-2, while for example decimal numbers are represented in radix-10 since ten digits are used (0,1,...,9). The notation commonly used to represent numbers is: $(x)_y$, where x is a number expressed in the base y. For example, $(10)_{10}$ represents the number ten in the decimal system; $(10)_2$ represents the number two in the binary system. In hardware design, the numeral systems such as octal (radix-8), hexadecimal (radix-16), etc., serve only to facilitate the manipulation of long chains of 0's and 1's. The ultimate effective implementation is realized in binary (radix-2).

Binary arithmetic is a mathematical field mainly concerned with:

- The study of number representation systems in order to eliminate logic-redundancy in the recoding. In other words, the determination of smaller numeric bases that represent binary numbers with a minimal number of digits;
- The search for algorithms that efficiently performs arithmetic operations (–, +,×, /, $\sqrt{\ }$, a^n, etc) based on a given number representation system;
- The exploration of the best implementation techniques when the target computational device is a DSP, a FPGA, or an ASIC.

There exist different types of binary arithmetic. The characteristics of each arithmetic are fundamentally conditioned by the format used to represent numbers. The mostly used arithmetics are: floating-point arithmetic and fixed-point arithmetic. The implementations and optimizations undertaken within each arithmetic are drastically different [1]. The choice of the arithmetic to be used is mainly dictated by the desired precision and the dynamic range of the application.

Furthermore, the complexity level of the arithmetic is determined by the required operations. The latter basically depend on the kind of the system treated: linear system or nonlinear system. A linear system is a mathematical model based on linear operations (–, +, ×, /). Linear systems typically exhibit features and properties that are much simpler and easier to understand and manipulate than the more general nonlinear case which require complex operations, such as polynomial or trigonometric functions.

3.2 Number Representation Formats

Representing an infinite, continuous set of *real numbers* with a finite set of *machine numbers* is not a straightforward task. Clever compromises must be found between some non-compatible requirements. Among the diverse requirements, the most desirable ones are [2]:

- Speed: In computation-intensive applications, as in digital control or DSP, computation-time is a critical factor that limits the whole system performance;
- Precision: Even if speed is important, getting inaccurate results faster may be worse than getting the correct results later;
- Range: We may need to represent large as well as tiny numbers;
- Portability: A program written on a given machine must run with no modifications on different machines;
- Ease of use and implementation: If a given arithmetic is too arcane, almost nobody will use it.

Various number formats exist [3]. The most commonly used formats are summarized as follows:

- Fixed-point number format: Offers a limited range and/or precision, but easy to implement. Very convenient for high-speed and low-power applications. It handles integer numbers $x \in I = \{-N,...,N\}$ as well as rational numbers of the form $x = a/2^f$ ("binary" rational), $a \in I$, and f is a positive integer.
- Floating-point number format: This is the most common approach. It offers a wide dynamic range and a high precision to accommodate extremely large and small numbers, respectively. However, it is relatively difficult to implement. It handles numbers of the form $x \times b^E$, where x is a rational number, b is an integer base, and E is an integer exponent.
- Logarithmic number format: Represents numbers by their signs and logarithms. Attractive for applications needing low precision and wide dynamic range.
- Rational number format: Approximates a real value by the ratio of two integers. Leads to difficult arithmetic operations.

We limit ourselves to the fixed-point and floating-point representations.

3.2.1 Fixed-Point Format

Numbers in the fixed-point format are represented by an ordered n-tuple. Each of the elements of the n-tuple is called a digit, and the n-tuple is called a digit-vector [1]. We begin with the representation of nonnegative integers, followed by the representation of signed integers.

3.2.1.1 Representation of Nonnegative Integers

The digit-vector that represents the integer x is denoted by

$$x = (x_{n-1}, x_{n-2}, \ldots, x_1, x_0). \tag{3.1}$$

Note that we use a zero-origin, leftward-increasing indexing. The number system to represent x consists of the following elements:

- The number of digits n.
- A set of numerical values for the digits. We call D_i the set of values of x_i. The cardinality of the set D_i is denoted by $|D_i|$. For example, $\{0,1,2,...,9\}$ is the digit set for the conventional decimal number system with cardinality 10.
- A rule of interpretation. This rule corresponds to a mapping between the set of digit-vector values and the set of integers.

The set of integers, each represented by a digit-vector with n digits, is a finite set with at most $K = \prod_{i=0}^{n-1} |D_i|$ different elements since this is the maximum number of different digit-vectors. For example, in a conventional decimal system a digit-vector of six digits can represent a million values. Sets that have been found generally useful to perform basic arithmetic operations include, for example, all integers from 0 to K–1.

The number systems most frequently used are *weighted systems*. Their representation mapping is:

$$x = \sum_{i=0}^{n-1} x_i \times w_i \,, \tag{3.2}$$

where $w = (w_{n-1}, w_{n-2}, ..., w_1, w_0)$ is the *weight-vector*.

A *radix number system* is a weighted number system in which the weight-vector is related to the *radix-vector* $r = (r_{n-1}, r_{n-2}, \ldots, r_1, r_0)$ as follows:

$$w_0 = 1; \quad w_i = w_{i-1} \times r_{i-1} \quad (1 \le i \le n-1). \tag{3.3}$$

This is equivalent to:

$$w_0 = 1; \quad w_i = \prod_{j=0}^{i-1} r_j \,. \tag{3.4}$$

Radix number systems are classified according to the radix-vector into fixed-radix and mixed-radix systems. In a *fixed-radix* system all elements of the radix-vector have the same value r (the radix). Consequently, the weight-vector is

$$w = \left(r^{n-1}, r^{n-2}, \ldots, r^2, r\ \ 1\right), \tag{3.5}$$

and the digit-sets are

$$D_i = D \quad \left(1 \le i \le n-1\right), \tag{3.6}$$

and

$$x = \sum_{i=0}^{n-1} x_i \times r^i \ . \tag{3.7}$$

The most frequently used radices are powers-of-two, such as 2 (binary), 4 (quaternary), 8 (octal), 16 (hexadecimal), and so on. The corresponding range of x represented with n radix-r digits is:

$$0 \le x \le r^n - 1 . \tag{3.8}$$

According to the set of digit values (D_i), the radix number systems are classified into redundant and nonredundant systems. A number system is *nonredundant* if each digit-vector represents a different integer; that is, if the representation mapping is one to one. It is *redundant* if there are integers that are represented by more than one digit-vector. More precisely, in a nonredundant system the set of values for D_i is $\{0,1,\ldots,r_{i-1}\}$ with $|D_i| = r_i$. For example, the nonredundant digit sets in the binary, quaternary, octal, and hexadecimal number systems are $\{0,1\}$, $\{0,1,2,3\}$, $\{0,1,2,\ldots,7\}$, $\{0,1,2,\ldots,15\}$, respectively.

A digit set D_i such that $|D_i| > r_i$ produces a redundant system allowing more than one representation of a value; for example, in the $\{-1,0,1\}$ binary system the vectors (0,0,1,1,1,1,0) and (0,1,0,0,0,–1,0) both represent the integer "thirty" (Table 3.1). An exception to this rule is the *canonical* systems that yield *minimum number* of digits for each value of x varying from 0 to $r^n - 1$. Canonical systems are *nonredundant* even if $|D_i| > r_i$. An example of such systems is the well-known Canonical Signed Digit (CSD) representation, used in designing the vast majority of LTI systems [4].

A system with fixed positive radix r and nonredundant set of digit values is called a radix-r conventional number system. These are by far the most commonly used number systems.

TABLE 3.1 – Representation of the integer "thirty" in different number systems.

Number system	Digit vector
Conventional radix-2 system (binary)	$(0011110)_2$
Conventional radix-3 system	$(0001010)_3$
Conventional radix-4 system	$(0000132)_4$
Conventional radix-10 system	$(0000030)_{10}$
Redundant Radix-2 system with digit set $\{-1=\bar{1},0,1\}$	$(0011110)_2$ $(0100\bar{1}0)_2$

For the implementation of arithmetic algorithm in (binary) digital systems, it is necessary to represent the digit-vectors by bit-vectors. This is done by defining a *code* for a digit and mapping the digit-vector by mapping each digit according to this code. In the binary (conventional) number system, the code is direct: the binary-digit values 0 and 1 are represented by the binary-variable values 0 and 1, respectively. For higher power-of-two radices, the most common code is the binary code, in which a digit d is represented by a bit vector $(d_{k-1}d_{k-2}\ldots d_1d_0)$ of $k=\log_2(r)$ bits, such that:

$$d=\sum_{i=0}^{k-1} d_i\times 2^i. \tag{3.9}$$

The use of this code for each digit results in a bit vector for x that is the same for any power-of-two radix, the only difference being the way the bits are grouped to form a digit. In the binary case, each bit corresponds to a digit, while in the radix-r case, groups of $\log_2(r)$ bits form a digit. Therefore, conversion from a bit-vector in a radix-2 representation to a radix-r representation and vice-versa is trivial. For example, the bit-vector

$$\begin{aligned} x &= (110001011101)_2 \\ &= ([110]\ [001]\ [011]\ [101])_8 \\ &= ([1100]\ [0101]\ [1101])_H \end{aligned} \tag{3.10}$$

corresponds to the octal digit-vector $(6135)_8$ and the hexadecimal digit-vector $(C5D)_H$. The fact that bit-vectors are identical permits the use of some binary algorithms to perform operations on integers represented in these higher radices.

3.2.1.2 Representation of Signed Integers

In the previous section we presented the representation of nonnegative integers. We now extend the discussion to the representation of signed integers (positive and negative). Two representations are by far the most common:

- The sign-and-magnitude (SM) representation: In SM, a signed integer x is represented by a pair (x_s,x_m), where x_s is the sign and x_m is the magnitude (positive integer). The two values of the sign (+,–) are represented by a binary variable, where traditionally "0" corresponds to + and "1" to –. The magnitude can be represented by any system for the representation of positive integers. If a conventional radix-r system is used, the range of signed integers, for n digits in the representation of the magnitude, is $0\le x_m\le r^n-1$. Note that zero has two representations: x_s=0, x_m=0 (positive zero) and x_s=1, x_m=0 (negative zero).
- The true-and-complement (TC) representation: In this system, there is no separation between the representation of the sign and the representation of the magnitude, but the whole signed integer is represented by a positive integer. The representations of positive integers are called *true forms*, and those of negative integers, *complement forms*. While TC is expressed in the general case by radix-r, we consider only the special case of radix-2, called the *two's complement representation*. The latter is described hereafter in details.

In the two's complement representation, a signed integer (bit-vector) x with n digits is represented as follows:

$$x = -x_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} x_i \times 2^i \,. \tag{3.11}$$

In converting a bit-vector to a value, we use the fact that the most significant bit (x_{n-1}) of x has a negative weight, while the remaining bits have positive weights. For example,

$$x = (11011) \rightarrow -16+8+2+1 = -5 \,.$$

The two's complement representation (Eq. 3.11) has the following properties:

- The representation of zero is unique. Zero is obtained when all digits are set to zero.
- The range of numbers is not symmetrical since $x = -2^{n-1}$ is representable but $x = 2^{n-1}$ is not. That is, the range is $\left[-2^{n-1}, 2^{n-1}-1\right]$.

3.2.1.3 Fixed-Point Arithmetic of Two's Complement Numbers

Before describing the arithmetic operations, we need first to formalize the representation of numbers in fixed-point format [5]. Let β be the number of digits (bits) of a signed number (bit-vector) x, and γ the number of digits of the fractional part of x (Fig. 3.1). Let α be the number of digits of the integer part (α=β–γ). Finally, FPR_x denotes the tuple (β, α, γ) defining the fixed-point representation of x. Hence, $x = x_{Int} + x_{fr}$ is written as $x = \left(x_{\alpha-1} \ldots x_1 x_0 \bullet x_{-1} \ldots x_{-\gamma+1}\, x_{-\gamma}\right)$, such that

$$x = -x_{\alpha-1} \times 2^{\alpha-1} + \sum_{i=-\gamma}^{\alpha-2} x_i \times 2^i \,. \tag{3.12}$$

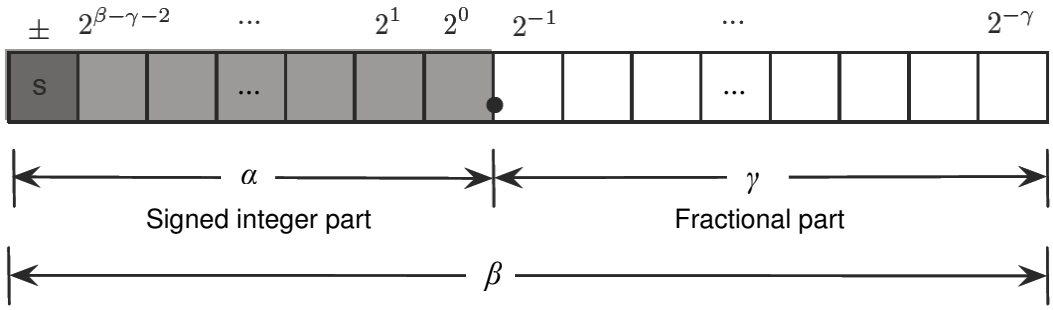


FIGURE 3.1 – Fixed-point representation of a signed real number in two's complement system.

To convert a real number x to FPR, we proceed as follows. β is a given value since it must be equal to the bit-width of the data-path. The number of bits of the integer part is given by:

$$\alpha_x = \left\lfloor \log_2 |x| \right\rfloor + 2 \,, \tag{3.13}$$

with $\lfloor a \rfloor$ is the floor function that rounds a to the nearest integer lower than or equal to x. Hence, we determine the fractional part as $\gamma_x = \beta_x - \alpha_x$, and the FPR of x is given by: $FPR_x = (\beta_x, \alpha_x, \gamma_x)$.

Conversely, note that in fixed-point representation, x is represented by the integer N_x such as:

$$N_x = \lfloor x \times 2^{\gamma_x} \rceil, \tag{3.14}$$

where $\lfloor a \rceil$ is the function that rounds a to the nearest integer. Thus, x is approximated by

$$x^{\dagger} = N_x \times 2^{-\gamma_x}. \tag{3.15}$$

Multiplication and addition are the two mostly used operations in linear systems. In fixed-point format, they are handled as follows.

3.2.1.4 Multiplication

Let us consider the operation $z = x \times y$, with $(\beta_x, \alpha_x, \gamma_x)$ and $(\beta_y, \alpha_y, \gamma_y)$. FPR_z is given by:

$$FPR_z = (\beta_x + \beta_y, \alpha_x + \alpha_y, \gamma_x + \gamma_y), \tag{3.16}$$

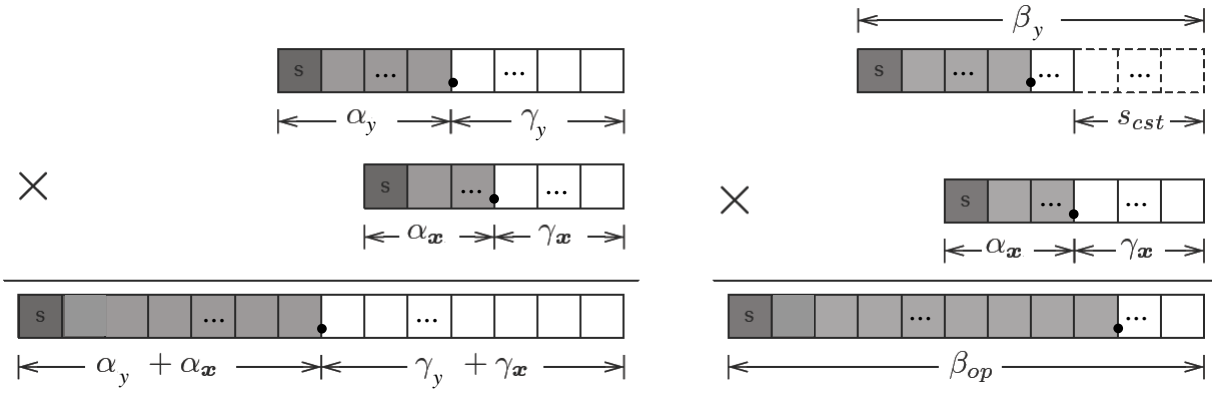
and the multiplication operation is realized by $N_z \leftarrow N_x \times N_y$. In Eq. 3.16 the multiplication is performed in *double-precision* since $\beta_z = \beta_x + \beta_y$, but generally the bit-width (W_{dp}) of the data-path is smaller than $(\beta_x + \beta_y)$. In this case, let us denote $\beta_{op} = W_{dp}$, and give the general expression of FPR_z:

$$FPR_z = (\beta_{op}, \alpha_x + \alpha_y, \beta_{op} - \lfloor \alpha_x + \alpha_y \rceil), \tag{3.17}$$

and the operation is realized by

$$N_z \leftarrow (N_x >> s_x) \times (N_y >> s_y), \tag{3.18}$$

where s_x and s_y and are right bit shifts applied on N_x and N_y such that $s_x + s_y = (\beta_x + \beta_y) - \beta_{op}$. But if $\beta_{op} = (\beta_x + \beta_y)$, then $s_x = s_y = 0$. The special and general cases given by Eq. 3.16 and 3.17, respectively, are illustrated by Fig. 3.2. A number of S_{cst} bits are truncated from the operand Y such that $\beta_{op} = W_{dp}$.



(a) Double precision multiplication (b) Simple precision multiplication

FIGURE 3.2 – Double (a) and Simple (b) precision.

3.2.1.5 Addition

Adding two fixed-point numbers requires the same point position. Let us consider the operation $z = x \times y$, with $(\beta_x, \alpha_x, \gamma_x)$ and $(\beta_y, \alpha_y, \gamma_y)$. The full-precision FPR_z is given by:

$$\begin{cases} \alpha_z = \max(\alpha_x + \alpha_y) + 1 \\ \gamma_z = \max(\gamma_x + \gamma_y) \\ \beta_z = \alpha_z + \gamma_z \end{cases} . \quad (3.19)$$

The general case (limited precision) where $\beta_{op} = W_{dp}$ is given by:

$$\begin{cases} \alpha_z = \max(\alpha_x + \alpha_y) + 1 \\ \gamma_z = \beta_{op} - \max(\alpha_x + \alpha_y) - 1 \\ \beta_z = \beta_{op} \end{cases} , \quad (3.20)$$

and the addition operation is realized by

$$N_z \leftarrow (N_x >> s_x) \times (N_y >> s_y), \quad (3.21)$$

with $s_x = \gamma_x - \gamma_z$ and $s_y = \gamma_y - \gamma_z$.

3.2.1.6 Overflow Detection

In the two's complement representation, an overflow occurs when the operands are of the same sign and the result of the addition represents an integer of opposite sign. Since the sign is determined by the most significant bit (x_{n-1}), the overflow detection is specified by the following switching expression:

$$AVF = (\bar{x}_{n-1} \times \bar{y}_{n-1} \times z_{n-1}) + (x_{n-1} \times y_{n-1} \times \bar{z}_{n-1}). \quad (3.22)$$

3.2.2 Floating-Point Format

As indicated earlier, a floating-point representation is used to represent real numbers. Since, as in fixed-point representation, the floating-point representation is encoded in a finite number of bits, it is possible to represent only a finite subset of the infinite set of real numbers. For a specific floating-point system, a real number that is *exactly* represented in the system is called a *floating-point number*. The rest of the real numbers either fall outside the range of the representation (overflow and underflow) or are represented by floating-point numbers that have a value that approximates the real number. The process of approximation is called *roundoff* (or rounding) and produces a roundoff error.

A floating-point number x is represented by a triple (S_x, M_x, E_x), such that:

$$x = (-1)^{S_x} \times M_x \times b^{E_x}, \quad (3.23)$$

where b is a constant called the base; E_x is a signed integer exponent; M_x is the significand (also called the mantissa); and $S_x \in \{0,1\}$ is the sign of the significand.

3.2.2.1 Dynamic Range

The objective of using floating-point representation is to increase the dynamic range, with respect to a fixed-point representation. This *dynamic range* is defined as the ratio between the largest and the smallest nonzero and positive number that can be represented [1]. For a fixed-point representation using n radix-r digits for the magnitude, the dynamic range is:

$$DR_{fxpt} = r^n - 1. \tag{3.24}$$

In contrast, for the floating-point representation:

$$DR_{ftpt} = \frac{M_{\max} \times b^{E_{\max}}}{M_{\min} \times b^{E_{\min}}}\ . \tag{3.25}$$

For instance, if the n digits are partitioned so that m digits are used for the significand and $n–m$ digits for the exponent, and $b=r,$ we get:

$$DR_{ftpt} = \left(r^m - 1\right) \times r^{\left(r^{n-m}-1\right)}. \tag{3.26}$$

As an example, if n=32, m=24, and r=2, the corresponding dynamic ranges are:

$$DR_{fxpt} = 2^{32} - 1 \approx 4.3 \times 10^9;$$

$$DR_{ftpt} = \left(2^{24} - 1\right) \times 2^{2^8 - 1} \approx 9.7 \times 10^{83}.$$

A large dynamic range is required in many applications to avoid overflows and underflows. If the dynamic range of the fixed-point representation is not sufficient, complicated operations have to be included in the program. Thus, a floating-point system is preferable in such applications.

3.2.2.2 Precision

In numerical analysis, errors are very often expressed in terms of relative errors. And yet, when we want to express infinitesimal errors, it is more adequate and frequently more accurate to express errors in terms of what we would intuitively define as the weight of the last bit of the significand (Fig. 3.3). To make that notion clearer, the term ulp (acronym for unit in the last place) is used. Several slightly different definitions of ulp exist in the literature. We cite hereafter the two most frequent ones:

> "*$ulp(x)$ is the gap between the two floating-point numbers nearest to x, even if x is one of them.*"
>
> *"ulp(x) is the distance between the closest straddling floating-point numbers a and b (i.e., those with $a \leq x \leq b$ and $a \neq b$), assuming that the exponent range is not upper-bounded."*

All ulp definitions coincide as long as x is not extremely close to a power of the radix. They have complex properties that differ to a small extent. However, a deep understanding of these complex properties is necessary for anyone who wants to prove exact tight bounds on the errors of infinitesimal computations. This issue goes beyond the scope of this thesis. Readers interested to study the precision of floating-point arithmetic are referred to [2].

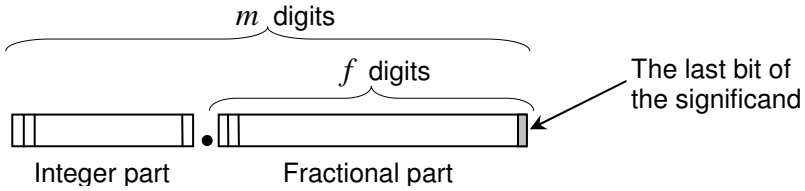


FIGURE 3.3 – Significand in radix-r fixed-point representation.

Devoting more digits to the exponent part widens the number representation range but reduces the precision. There is a trade-off between the range and precision that is quite critical. Generally, to improve the precision without much altering the range, the integer part in the significand is eliminated to the benefit of the fractional part, that is, we take f=m.

Contrary to the fixed-point representation where the gap between consecutive numbers is constant (2^{-f}), in floating-point representation the number distribution within the dynamic range is not uniform. Smaller numbers are denser, and larger numbers are sparser. Fig. 3.4 shows the number distribution pattern and the various subranges in floating-point representations. In particular, it includes the three special or singular values –∞, 0, and +∞ and depicts the meanings of overflow and underflow. Overflow occurs when a result is less than –*max* or greater than *max*. Underflow, on the other hand, occurs for results in the range (–*min*, 0) or (0, *min*).

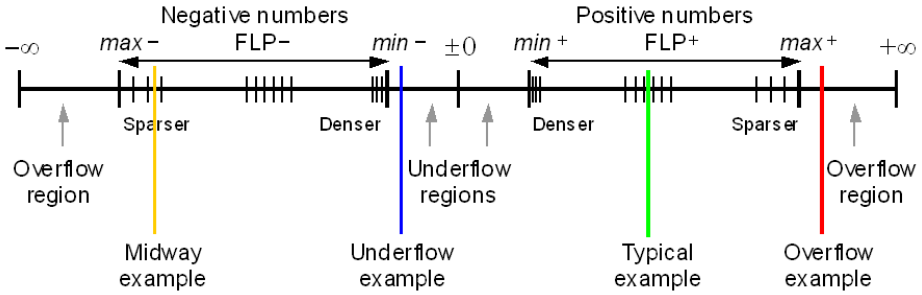


FIGURE 3.4 – Distribution of floating-point numbers within the dynamic range.
Source: [3]

In floating-point representation, the code assignment patterns are different, leading to different ranges and error characteristics. For the same range of representable values, floating-point tends to be better than fixed-point in terms of average relative-representation-error, even though the absolute representation error increases as the values get larger [3].

Though floating point representation provides greater dynamic range and better precision than fixed point, it is far more expensive to implement. Most embedded system applications tolerate a certain degree of inaccuracy and use the much simpler fixed-point notation to increase throughput and decrease area, delay, and energy.

Illustrative Example

The two representation formats in IEEE standard for radix-2 (binary) floating-point numbers, formally known as "ANSI/IEEE Std 754-1988," are depicted in Fig. 3.5. The short, or single-precision, format is 32 bits wide, whereas the long, or double-precision, version requires 64 bits. The two formats have 8-bit and 11-bit exponent fields and use exponent biases of 127 and 1023, respectively. The

significand is in the range [1, 2), with its single whole bit, which is always 1, removed and only the fractional part shown. The notation "23+1" or "52+1" for the width of the significand is meant to explain the role of the hidden bit, which does contribute to the precision without taking space. Table 3.2 summarizes the most important features of the IEEE standard floating-point representation formats.

	Sign	Biased exponent	Significand $M=1.f$ (the 1 is hidden)
	±	E+bias	f
32-bit:	1 bit	8 bits, bias = 127	23 + 1 bits, single-precision or short format
64-bit:	1 bit	11 bits, bias = 1023	52 + 1 bits, double-precision or long format

FIGURE 3.5 – The ANSI/IEEE standard 754-1988 floating-point representation.

In the 32-bit format, the largest and smallest numbers are $\pm 3.4\times10^{38}$ and $\pm 1.2\times10^{-38}$, respectively. The represented values are unequally spaced between these two extremes, such that the gap between any two numbers is about 10^{-7} times smaller than the value of the numbers. This is important because it places large gaps between large numbers, but small gaps between small numbers (ditto for 64-bit format).

TABLE 3.2 – Some features of the ANSI/IEEE 754-1988 standard floating-point number representation formats.

Feature	Single/Short	Double/Long
Word width, bits	32	64
Significand bits	23 + 1 hidden	52 + 1 hidden
Significand range	$[1, 2-2^{-23}]$	$[1, 2-2^{-52}]$
Exponent bits	8	11
Exponent bias	127	1023
Zero (±0)	E + *bias* = 0, f=0	E + *bias* = 0, f=0
Denormal	E + *bias* = 0, $f\neq0$ represents $\pm 0.f\times2^{-126}$	E + *bias* = 0, $f\neq0$ represents $\pm 0.f\times2^{-1022}$
Infinity (±∞)	E + *bias* = 255, f=0	E + bias = 2047, f=0
Not-a-number (NaN)	E + *bias* = 255, $f\neq0$	E + *bias* = 2047, $f\neq0$
Ordinary number	E + *bias* ∈ [1, 254] $E \in [-126,127]$ represents $1.f\times2^{E}$	E + *bias* ∈ [1, 2046] $E \in [-1022,1023]$ represents $1.f\times2^{E}$
min (±)	$2^{-126}\approx 1.2\times10^{-38}$	$2^{-1022}\approx 2.2\times10^{-308}$
max (±)	$\approx 2^{128}\approx 3.4\times10^{38}$	$\approx 2^{1024}\approx 1.8\times10^{308}$

3.3 Number Representation Systems

We have deliberately separated number representation formats from number representation systems. The latter rely on the former to create sophisticated arithmetic algorithms.

Number systems are developed in order to enable a reduction of the complexity of the arithmetic operations. The reason is that in most applications, the computational complexity of algorithms crucially depends upon *the number of zeros* of the input data in the corresponding number system. Because each number system exhibits specific numerical properties, the arithmetic operations are handled differently.

Only number systems based on fixed-point format are concerned in this thesis. In the previous section, we developed through Eq. 3.1 to 3.10 the general case of radix theory. This includes: fixed/mixed radix system; redundant/nonredundant number representation; and canonical/noncanonical number representation. These notions are the fundamentals of number systems.

3.3.1 Canonical Signed Digit (CSD)

CSD is the canonical form of the signed digit (SD) representation developed by Avizienis [6] in 1961. SD is a redundant fixed-radix (r=2) representation defined as follows:

$$x = \sum_{i=0}^{n-1} x_i \times 2^i \quad \text{with} \quad x_i \in D = \{\bar{1}, 0, 1\}. \tag{3.27}$$

The digit x_i is a ternary digit, sometimes called trit. It requires two bits to represent it, but the major benefit of SD is that addition/subtraction can be made without carry-propagation, accelerating therefore the operations, especially for large operand addition/subtraction.

In SD, the integer "seven", for example, has several representations:

$$(0111)_2 = 4+2+1 = 7\,;\ (10\bar{1}1)_2 = 8-2+1 = 7\,;\ (1\bar{1}11)_2 = 8-4+2+1 = 7\,;\ (100\bar{1})_2 = 8-1 = 7\,.$$

The rational number "5/8", for exemple; can be written differently:

$$\frac{5}{8} = 0.625 = (0.101)_2 = (1.\bar{1}01)_2 = (1.\bar{1}1\bar{1})_2 = (1.0\bar{1}\,\bar{1})_2\,.$$

One conversion method from two's complement notation to SD representation is to use Booth encoding (Table 3.3). For example, the two's complement value 7=$(0111)_2$, is converted to SD through the steps described in Fig. 3.6.

TABLE 3.3 – Booth encoding.

Y_j	Y_{j-1}	Digit
0	0	0
0	1	1
1	0	$\bar{1}$
1	1	0

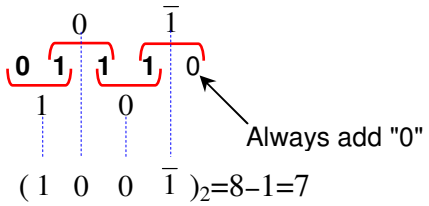


FIGURE 3.6 – Conversion process from Two's complement notation to SD representation of the positive integer "7".

A signed-digit representation with minimum number of non-zero digits is called a minimum signed-digit (MSD) representation. In general, there are several MSD representations. One particular MSD representation can be obtained by constraining two adjacent digits to not be both non-zero, that is, $x_{i+1} \times x_i = 0$. This particular MSD representation is called canonical signed digit (CSD)

representation. The CSD representation is unique and therefore not redundant. Reconsidering the example given above, the CSD representation for the numbers "7" and "5/8" are $(100\bar{1})_2$ and $(0.101)_2$, respectively.

The conversion from SD to CSD is done as follows:

- Convert long string of 1's: $011...11 \Rightarrow 100...0\bar{1}$;
- Convert long string of $\bar{1}$'s: $0\bar{1}\bar{1}...\bar{1}\bar{1} \Rightarrow \bar{1}00...01$;
- Merge adjacent digits of opposite signs: $1\bar{1} \Rightarrow 01$; $\bar{1}1 \Rightarrow 0\bar{1}$.

Using the CSD format on a n-bit value, the number of *non-zero* digits is bounded by (n+1)/2, and it tends asymptotically to an average value of (n/3)+(1/9). Compared to the traditional binary representation that requires n/2 digits on the average, CSD allows a *saving* of 33% in non-zero digits. This means that in constant multiplication $(C \times X)$, 33% less additions/subtractions are required, which leads to much compact implementations of LTI systems [4]. This is the reason why CSD is so popular.

Despite the fact that CSD minimizes the number of nonzero digits for the constant representation, it is far from being optimal. It is possible to decompose the n-bit value to further reduce the number of operations. This can be achieved using more complex number systems.

3.3.2 Double Base Number System (DBNS)

DBNS arithmetic was developed by Dimitrov in 1999 [7]. In DBNS, an integer x is expressed using bases 2 and 3, as follows:

$$x = \sum_{i,j} d_{i,j} \times 2^i \times 3^j \quad \text{with} \quad d_{i,j} \in D = \{0,1\}. \tag{3.28}$$

For example, the integer x=(10599)$_{10}$ is written in DBNS as follows:

$$x = (3^2 \times 2^8)+(3 \times 2^5)+(3^0 \times 2^{13})+(3^0 \times 2^3) -(3^0 \times 2^0) = (10599)_{10}.$$

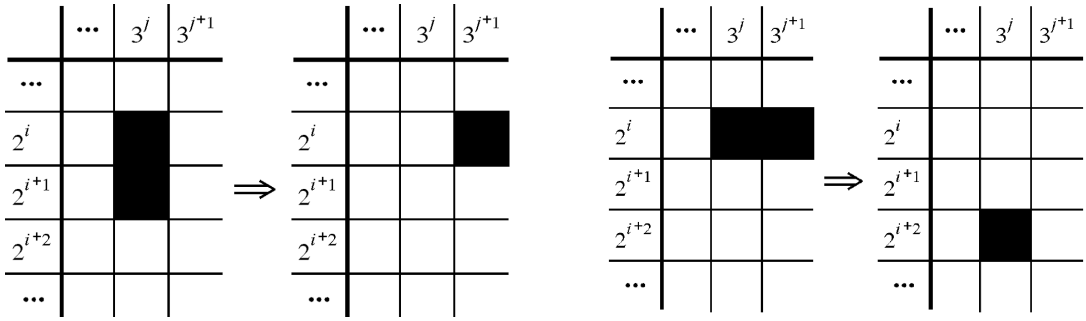
According to Eq. 3.2 and 3.3, DBNS is a weighted system, but not a radix system. DBNS representation is highly redundant; it is set clear from Eq. 3.28 that the traditional binary system is a special case (j=0) of DBNS. The canonical DBN representation (CDBNR) that expresses a given integer as a sum of minimal number of 2-integers $(2^i \times 3^j)$ is very complex to determine (NP-complete problem). Thus, arithmetic operations in this number system do not guarantee that the results are obtained in the minimal form. The author proposed a minimization heuristic called the greedy algorithm with the input as a positive integer x; and an output of 2-integers, a_i, such that $\sum_i a_i = x$. The algorithm finds the largest 2-integer, w, smaller than or equal to x, and recursively applies the same for x–w until reaching zero. The greedy algorithm terminates after

$O(\log(x)/\log(\log(x)))$ steps. The representation obtained by the greedy algorithm is called near-canonical DBNR (NCDBNR).

The mechanism of finding the NCDBNR plays a crucial role in performing basic arithmetic operations. NCDBNR form is further minimized by reducing adjacent non-zero digits based on a number of reduction rules given hereafter. The resulting form is called addition-ready DBNR (ARDBNR).

3.3.2.1 Basic ARDBNR reduction rules

One can use a geometrical interpretation, with orthogonal dimensions for each of the bases, to represent numbers in the DBNR. Nonzero DBNR digits are shown as black squares (active cells). This interpretation allows demonstrating simple identities on special combinations of active cells that provide a transformation of a DBNR to an ARDBNR. For example, Fig. 3.7(a) shows the representation of the identity $2^i \times 3^j + 2^{i+1} \times 3^j = 2^i \times 3^{j+1}$ to remove consecutive active cells lying in one column. Fig. 3.7(b) demonstrates the application of the identity, $2^i \times 3^j + 2^i \times 3^{j+1} = 2^{i+2} \times 3^j$, to remove consecutive active cells lying in one row.



(a) Column reduction Source: [7] (b) Row Column

FIGURE 3.7 – Reduction of consecutive active cells lying in one column (a) and in one row (b).

3.3.2.2 Advanced ARDBNR reduction rules

A generalized solution for the reduction can be applied using the purely exponential Diophantine equation:

$$2^{i_1} \times 3^{j_1} + 2^{i_2} \times 3^{j_2} + \ldots + 2^{i_k} \times 3^{j_k} = 2^{m_1} \times 3^{n_1} + 2^{m_2} \times 3^{n_2} + \ldots + 2^{m_l} \times 3^{n_l} \quad \text{with} \quad l<k. \qquad (3.28)$$

The problem of solving such Diophantine equations has been a subject of investigation over the last two decades, although some interesting results were obtained in the 30s and 40s. The reality is that only some special cases for *k* and *l* are considered. For example, using *k* =2 and *l*=1, following theorem can be proved:

Theorem 3.1 – *The Diophantine equation $x+y=z$, where GCD(x,y,z)=1 and x, y, and z are 6-integers, that is, x, y, z have the form $2^{n_1} \times 3^{n_2} \times 5^{n_3} \times 7^{n_4} \times 11^{n_5} \times 13^{n_6}$, with $n_i \geq 0$ and $i \in \{1,2,3,4,5,6\}$, has exactly 545 solutions.*

Refer to [7] for the proof. For DBNS, $n_3=n_4=n_5=n_6=0$, and the only solutions of $x+y=z$ are (1, 2, 3), (1, 3, 4), and (1, 8, 9). Therefore, these represent the only three cases where we can replace two active

cells with one. For k =3 and l=1, an interesting possibility for reducing the active cells follows from the solution of the Pillai's equation [7]:

$$2^a \pm 2^b = 3^c \pm 3^d . \tag{3.29}$$

Pillai was able to solve all of the above four equations, excluding the equation $2^a - 2^b = 3^c - 3^d$ on which he conjectured that the only solutions are (3, 1, 2, 1), (5, 3, 3, 1), and (8, 4, 5, 1). For DBNS, only the two following equations are relevant:

$$2^a - 2^b = 3^c + 3^d ; \tag{3.30}$$

$$2^a + 2^b = 3^c - 3^d . \tag{3.31}$$

The solutions of Eq. 3.30 are (2, 1, 0, 0), (3, 1, 1, 1), (5, 1, 1, 3), (3, 2, 1, 0), (5, 2, 0, 3), (4, 2, 1, 2), and (8, 2, 5, 3), while Eq. 3.31 has only one solution (0, 0, 1, 0). The total number of solutions of the equation $x+y+z=t$, $GCD(x,y,z,t)$=1, in 2-integers is 27 [7].

3.3.2.3 Addition

Let x and y be two integers in the CDBNR. We note that if x and y contain the element $2^i \times 3^j$, then the element $2^{i+1} \times 3^j$ does not exist. Therefore, addition can be computed by simply overlaying the corresponding DBNS maps; there will be no overlapping active cells. In order to prepare for another addition, we ideally perform a reduction into minimal form. In practice, we only require an ARDBNR and the symbolic substitution methods described in the previous section can be effectively used.

Let us define $I_x(i, j)$ as the DBNS map of the integer x, represented in the ARDBNR. The image $I_z(i, j)$ of the DBNS map of the number $z=x+y$ can be obtained using:

$$I_z(i+1, j) = I_x(i, j)\ AND\ I_y(i, j) ; \tag{3.32}$$

$$I_z(i, j) = I_x(i, j)\ XOR\ I_y(i, j) . \tag{3.33}$$

Note, using the ARDBNR, if $I_x(i, j) = I_x(i, j) = 1$, then $I_x(i+1, j) = I_x(i+1, j) = 0$ and, therefore, addition can be accomplished using a symbolic substitution technique. To reduce this result, it is sufficient to use the following rules (see Fig. 3.7):

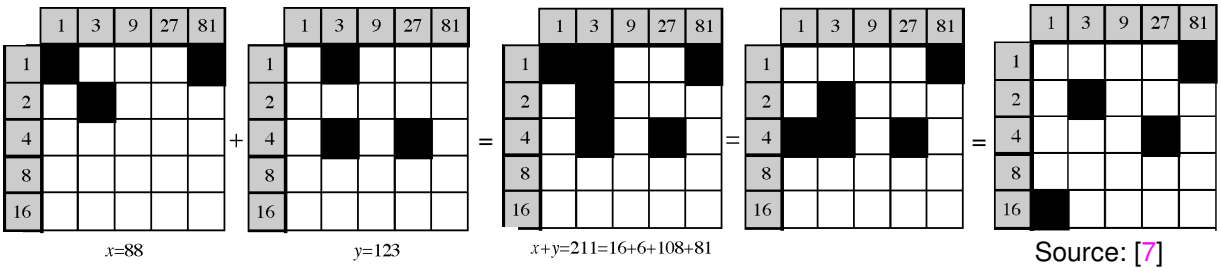
$$I_z(i, j+1) = I_z(i, j)\ AND\ I_z(i+1, j) ; \tag{3.34}$$

$$I_z(i+2, j) = I_z(i, j)\ AND\ I_z(i, j+1) . \tag{3.35}$$

As an example of performing addition, let us consider the addition of the numbers 88 and 123 using the proposed technique. The representation of the numbers is NCDBNR, obtained via the greedy algorithm. For the number, 88, the NCDBNR gives 88 = 81 + 6 + 1 (three active cells), while the CDBNR consists of only two active cells (88 = 72 + 16 = 64 +24). The addition operation is presented in Fig. 3.8. The example is selected so that the application of the reduction rules, based on the solution of Eq. 3.32, gives a result which is not optimal. In fact, 211 requires 3 ones in the CDBNR; one of them could be found using more sophisticated substitution rules:

$$I_z(i+3, j)\ AND\ I_z(i+1, j) = I_z(i, j)\ AND\ I_z(i, j+1)\ AND\ I_z(i+1, j+1); \tag{3.36}$$

$$I_z(i+5, j) = I_z(i+1, j)\ AND\ I_z(i, j+1)\ AND\ I_z(i+1, j+3) . \tag{3.37}$$



Source: [7]

FIGURE 3.8 – Example of the DBNS addition process.

3.3.2.4 Multiplication

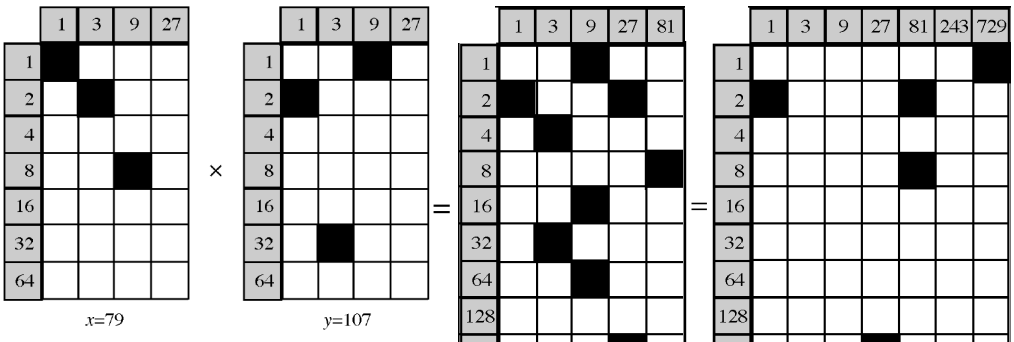
Let x and y be integers, represented by DBNS maps in the CDBNR. The CDBNR of their product, z, is an n-tuple of the elements $\left\{2^{i_z}\times 3^{j_z} = 2^{i_x+i_y}\times 3^{j_x+j_y}\right\}$, where the $\{i_x, j_x\}$ and $\{i_y, j_y\}$ are the 2-integer index locations of the active cells in the CDBNRs of x and y, respectively.

It is clear that the multiplication process simply corresponds to 2D shifts and DBNS additions in an equivalent way to that performed using binary arithmetic. The promise here, however, is that the number of operations is considerably reduced based on the sparseness of the representation. Let us consider the multiplication of the numbers 79 and 107, represented via their DBNS maps as shown in Fig. 3.9. The final reduced forms of the product can be found by using two specific solutions of the Pillai's equation 3.29:

$$I_z(i, j+4) = I_z(i, j)\ AND\ I_z(i+4, j)\ AND\ I_z(i+6, j); \tag{3.38}$$

$$I_z(i, j+3) = I_z(i+1, j)\ AND\ I_z(i, j+2)\ AND\ I_z(i+4, j)\ . \tag{3.39}$$

The representation of the multiplication result is shown in Fig. 3.9(a), and the ARDBNR reduction, using rules 3.38 and 3.39, in Fig. 3.9(b).



Source: [7]

FIGURE 3.9 – Example of the DBNS Multiplication Process.
(a) Multiplication result without reduction.
(b) Multiplication result ARDBNR reduction.

3.3.3 Residue Number System (RNS)

The concept of RNS dates back 1500 years ago in China [3]. In RNS, a number x is represented by the vector of its residues with respect to k prime moduli $m_{k-1} > m_{k-2} > \cdots > m_1 > m_0$. The residue

x_i of x with respect to the ith modulus m_i is similar to a digit, and the entire k-residue representation of x can be viewed as a k-digit number. Notationally, we write

$$x = (x_{k-1} \mid x_{k-2} \mid \cdots \mid x_1 \mid x_0)_{RNS(m_{k-1}|m_{k-2}|\cdots|m_1|m_0)} \; ;$$

$$x_i = x \bmod m_i = \langle x \rangle_{m_i} \quad \text{with} \quad x_i \in \{0,1,2,...,m-1\}. \tag{3.40}$$

The vector of moduli, $RNS(m_{k-1}|m_{k-2}|\cdots|m_1|m_0)$, can be deleted from the subscript when we have agreed on a default set. The product M of the k prime moduli is the number of different representable values in the RNS and is known as its *dynamic range*:

$$M = m_{k-1} \times m_{k-2} \times \cdots \times m_1 \times m_0 . \tag{3.41}$$

For example, $M = 8 \times 7 \times 5 \times 3 = 840$ is the total number of distinct values that are representable in $RNS = (8 \mid 7 \mid 5 \mid 3)$. Because of the equality

$$\langle -x \rangle_{m_i} = \langle M - x \rangle_{m_i} , \tag{3.42}$$

the 840 available values can be used to represent numbers from 0 up to 839, or from –420 up to +419, or any other interval of 840 consecutive integers. In effect, negative numbers are represented using a complement system with the complementation constant M. Here are some example numbers in RNS(8 | 7 | 5 | 3):

$(0 / 0 / 0 / 0)_{RNS}$ Represents 0 or 840 or . . .
$(1 / 1 / 1 / 1)_{RNS}$ Represents 1 or 84 1 or . . .
$(2 / 2 / 2 / 2)_{RNS}$ Represents 2 or 842 or . . .
$(0 / 1 / 3 / 2)_{RNS}$ Represents 8 or 848 or . . .
$(5 / 0 / 1 / 0)_{RNS}$ Represents 21 or 861 or . . .
$(0 / 1 / 4 / 1)_{RNS}$ Represents 64 or 904 or . . .
$(2 / 0 / 0 / 2)_{RNS}$ Represents –70 or 770 or . . .
$(7 / 6 / 4 / 2)_{RNS}$ Represents –1 or 839 or . . .

RNS representation is not redundant within the interval chosen of 840 consecutive integers. Given the RNS representation of x, the representation of $-x$ can be found by complementing each of the digits x_i with respect to its modulus m_i (0 digits are left unchanged). Thus, given that 21=(5 | 0 | 1 | 0)$_{RNS}$, we find: –21=(8–5 | 0 | 5–1 | 0)$_{RNS}$=(3|0|4|0)$_{RNS}$.

In practice, each residue must be represented or encoded in binary. For our RNS example, such a representation would require 11 bits (Fig. 3.10). To determine the number representation efficiency of our 4-modulus RNS, we note that 840 different values are being represented using 11 bits, compared to 2048 values possible with binary representation. Thus, the representational efficiency is 840/2048=41%.

3.3.3.1 Addition and Multiplication

As noted earlier, the sign of an RNS number can be changed by independently complementing each of its digits with respect to its modulus. Similarly, addition, subtraction, and multiplication can be performed by independently operating on each digit. The following examples for RNS(8 | 7 | 5 | 3)

illustrate the addition, subtraction, and multiplication processes, respectively:

$(5 / 5 / 0 / 2)_{RNS}$ Represents x=+5

$(7 / 6 / 4 / 2)_{RNS}$ Represents y=–1

$(4 / 4 / 4 / 1)_{RNS}$ $x+y$: $\langle 5+7 \rangle_8 = 4$, $\langle 5+6 \rangle_7 = 4$, etc.

$(6 / 6 / 1 / 0)_{RNS}$ $x–y$: $\langle 5-7 \rangle_8 = 6$, $\langle 5-6 \rangle_7 = 6$, etc.

$(3 / 2 / 0 / 1)_{RNS}$ $x \times y$: $\langle 5 \times 7 \rangle_8 = 3$, $\langle 5 \times 6 \rangle_7 = 2$, etc.

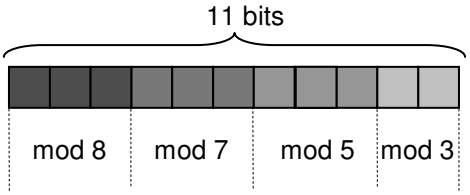


FIGURE 3.10 – Binary-coded number format for RNS(8 | 7 | 5 | 3).

The speed and simplicity are the primary advantages of RNS arithmetic. In the case of addition, for example, carry propagation is limited to within a single residue (a few bits). Thus, RNS representation pretty much solves the carry propagation problem. As for multiplication, a 4 x 4 multiplier (e.g.), is considerably more than four times simpler than a 16 x 16 multiplier, besides being much faster. In fact, since the residues are small (say, 6 bits wide), it is quite feasible to implement addition, subtraction, and multiplication by direct table lookup. With 6-bit residues, say, each operation requires a 4K x 6 table. Thus, excluding division, a complete arithmetic unit module for one 6-bit residue can be implemented with 9 KB of memory.

Unfortunately, however, what we gain in terms of the speed and simplicity of addition, subtraction, and multiplication can be more than nullified by the complexity of division and the difficulty of certain auxiliary operations such as sign test, magnitude comparison, and overflow detection.

3.3.3.2 Choosing the RNS Moduli

The set of the moduli chosen for RNS affects both the representational efficiency and the complexity of arithmetic algorithms. In general, we try to make the moduli as small as possible, since it is the magnitude of the largest modulus m_{k-1} that dictates the speed of arithmetic operations (*carry-propagation*). We also often try to make all the moduli comparable in magnitude to the largest one, since with the computation speed already dictated by m_{k-1}, there is usually no advantage in fragmenting the design through the use of very small moduli.

We illustrate the process of selecting the RNS moduli through two examples. Let us assume that we want to represent unsigned integers in the range 0 to $(100000)_{10}$ requiring 17 bits with standard binary representation. A simple strategy is to select appropriate prime numbers in sequence until the dynamic range M becomes adequate. For $(100000)_{10}$, it gives:

$$(17 / 13 / 11 / 7 / 3 / 2)_{RNS} ; \; M=(102102)_{10} .$$

With binary encoding of the six residues, the number of bits needed for encoding each number is:

$$5+4+4+3+2+1=19 \text{ bits.}$$

However, speed and cost do not just depend on the widths of the residues but also on the moduli chosen. Note that power-of-2 moduli simplify the required arithmetic operations, so that, for example, the modulus 16 might be better than the smaller modulus 13. Moduli of the form 2^a–1 are also desirable and referred to as low-cost moduli [3]. Hence, we are motivated to restrict the moduli to a power of 2 and odd numbers of the form 2^a–1. It is proved that the numbers 2^a–1 and 2^b–1 are relatively prime if and only if a and b are relatively prime. Thus, any list of relatively prime numbers $a_{k-2} > \cdots > a_1 > a_0$ can be the basis of the following k-modulus RNS

$$RNS\left(2^{a_{k-2}} \mid 2^{a_{k-2}} - 1 \mid \cdots \mid 2^{a_1} - 1 \mid 2^{a_0} - 1\right)$$

for which the widest residues are a_{k-2}-bit numbers. Note that to maximize the dynamic range with a given residue width, the even modulus is chosen to be as large as possible. Applying this strategy to our desired RNS with the target range [0, 100000], leads to the following result:

$$RNS\left(2^5 \mid 2^5 - 1 \mid 2^4 - 1 \mid 2^3 - 1\right) = RNS\left(32 \mid 31 \mid 15 \mid 7\right) ; M=104160.$$

The derived RNS requires 5+5+4+3=17 bits for representing each number, with the largest residues being 5 bits wide. In this case, the representational efficiency is close to 100% and no bit is wasted.

3.3.4 Radix-2^r Number System

The radix-2^r representation was developed by Sam in 1990 [8]. In radix-2^r, a n-bit two's complement number, x, is written as follows:

$$x = \sum_{j=0}^{(n/r)-1} \left(x_{rj-1} + 2^0 x_{rj} + 2^1 x_{rj+1} + 2^2 x_{rj+2} + \cdots + 2^{r-2} x_{rj+r-2} - 2^{r-1} x_{rj+r-1}\right) \times 2^{rj}$$

$$= \sum_{j=0}^{(n/r)-1} Q_j \times 2^{rj} \quad \text{with} \quad Q_j \in D = \{-2^{r-1}, -2^{r-1}+1, \cdots, -1, 0, 1, \cdots, 2^{r-1}-1, -2^{r-1}\} , \qquad (3.43)$$

where $x_{-1} = 0$ and $r \in \mathrm{N}^*$. For simplicity purposes and without loss of generality, we assume that r is a divider of N. In Eq. 3.43, the two's complement representation of x is split into n/r two's complement slices (Q_j), each of r+1 bit length. Each pair of two contiguous slices has one overlapping bit.

In fact, SD representation (Eq. 3.27) is a special case of radix-2^r representation (Eq. 3.43) for r=1.

The sign of the term Q_j is given by the bit x_{rj+r-1}, and $|Q_j| = 2^{k_j} \times m_j$, with $k_j \in \{0,1,2,...,r-1\}$ and $m_j \in OM(2^r) \cup \{0\}$, where $OM(2^r) = \{1,3,5,...,2^{r-1}-1\}$. $OM(2^r)$ is the set of odd positive digits in radix-2^r recoding, with $|OM(2^r)| = 2^{r-2}$. To $|Q_j| = 0$ corresponds m_j=0. Finally, x can be expressed as follows:

$$x = \sum_{j=0}^{(n/r)-1} (-1)^{x_{rj+r-1}} \times m_j \times 2^{rj+k_j} . \qquad (3.44)$$

Illustrative Example

In order to express x=(10599)$_{10}$ in radix-2^r, a two's complement representation of x is necessary, which is (010100101100111)$_2$. Thus, in two's complement notation, the constant size is equal to n=15. Let us choose r=4. As 15 is not a multiple of 4, the sign-bit (0 in this case) is extended by one position so as n=16. For C=10599, Eq. 3.43 and 3.44 become respectively:

$$x = \sum_{j=0}^{3} Q_j \times 2^{4j} \quad \text{, and} \quad x = \sum_{j=0}^{3} (-1)^{c_{4j+3}} \times m_j \times 2^{4j+k_j} .$$

Fig. 3.11 depicts the four Q_j terms. To determine the unknown values c_{4j+3}, m_j, and k_j, the radix-2^4 look-up table (Table 3.4) is indexed by Q_j terms. Referring to Table 3.4, the triplets (c_{4j+3}, m_j, k_j) corresponding to Q_0, Q_1, Q_2, and Q_3 are (0,7,0), (0,3,1), (1,7,0), and (0,3,0), respectively. Consequently, we can write:

$$x = (3\times2^{12})-(7\times2^{8})+(3\times2^{5})+(7\times2^{0}) = (10599)_{10}.$$

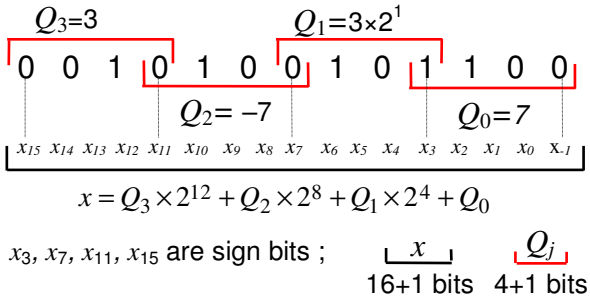


FIGURE 3.11 – Partitioning of (10599)$_{10}$ in radix-2^4.

TABLE 3.4 – Radix-2^4 look-up table.

Q_j					m_j	k_j
x_{4j+3}	x_{4j+2}	x_{4j+1}	x_{4j}	x_{4j-1}		
0	0	0	0	0	0	0
0	0	0	0	1	1	0
0	0	0	1	0	1	0
0	0	0	1	1	1	1
0	0	1	0	0	1	1
0	0	1	0	1	3	0
0	0	1	1	0	3	0
0	0	1	1	1	1	2
0	1	0	0	0	1	2
0	1	0	0	1	5	0
0	1	0	1	0	5	0
0	1	0	1	1	3	1
0	1	1	0	0	3	1
0	1	1	0	1	7	0
0	1	1	1	0	7	0
0	1	1	1	1	1	3
1	0	0	0	0	1	3
1	0	0	0	1	7	0
1	0	0	1	0	7	0
1	0	0	1	1	3	1
1	0	1	0	0	3	1

1	0	1	0	1	5	0
1	0	1	1	0	5	0
1	0	1	1	1	1	2
1	1	0	0	0	1	2
1	1	0	0	1	3	0
1	1	0	1	0	3	0
1	1	0	1	1	1	1
1	1	1	0	0	1	1
1	1	1	0	1	1	0
1	1	1	1	0	1	0
1	1	1	1	1	0	0

Note that for radix-2^4, $k_j \in \{0,1,2,3\}$, and $m_j \in \{0,1,3,5,7\}$

3.3.4.1 Canonical Radix-2^r Representation

Radix-2^r is a *redundant* representation since further simplifications are possible:

$$\cdots + 2^{r(j+1)} - 2^{rj+(r-1)} \pm \cdots = \cdots + 2^{rj+(r-1)} \pm \cdots \quad (3.45)$$

$$\cdots - 2^{r(j+1)} + 2^{rj+(r-1)} \pm \cdots = \cdots - 2^{rj+(r-1)} \pm \cdots \quad (3.46)$$

These two simplifications occur in Eq. 3.44 in case two consecutive terms with opposite signs, Q_j and Q_{j+1}, exhibit (m_j,k_j) pairs of the form (1, r–1) and (1,0), respectively.

Whether Eq. 3.45 and 3.46 are the unique simplifications that can be performed in radix-2^r needs a mathematical proof. In any case, the minimal form does exist, but the canonical form (unique) remains an *open research problem*.

3.4 Comparison of the Number Systems

So far, we have summarized the main arithmetic features of the Binary, SD, CSD, DBNS, RNS, and Radix-2^r number systems. It is premature at this stage to comment their hardware features without an effective implementation of a given arithmetic function. Instead, we only try to give an approximate idea through the expression of the value, $x = (10599)_{10}$, in different number systems. The comparison is based on the number of digits required by each number system (Table 3.5). Generally, the lower the number of digits, the less hardware resources are required, and the more speed is achieved.

TABLE 3.5 – Number of digits required by each number system for the integer value $(10599)_{10}$.

Number System	Arithmetic expression	Number of Digits
Binary	$x = 2^{13}+2^{11}+2^8+2^6+2^5+2^2+2^1+2^0$	8
SD	$x = 2^{13}+2^{12}-2^{11}+2^9-2^8+2^7-2^5+2^3-2^0$	9
CSD	$x = 2^{13}+2^{11}+2^9-2^7-2^5+2^3-2^0$	7
DBNS	$x = (3^2\times2^8)+(3\times2^5)+(3^0\times2^{13})+(3^0\times2^3)-(3^0\times2^0)$	5+2=7
RNS	x is represented by $(7 / 28 / 9)_{RNS(32 \mid 31 \mid 15)}$	3+3+2=8
Radix-2^r	$x = (3\times2^{12})-(7\times2^8)+(3\times2^5)+(7\times2^0)$	4+2=6

Note that in DBNS and Radix-2^r, the terms 3^2, 3, and 7 are considered as extra-digits.

As already mentioned at the beginning of this chapter, the objective of a number system is the *maximum absorption* of logic-redundancy in the representation of numbers. This will lead to fewer digits, and therefore to fewer hardware resources and more speed. This absorption process may be simple such in the case of CSD, or very complicated such in the case of DBNS and RNS systems. Depending on the type of arithmetic function to be implemented, the absorption process is carried out either in software or in hardware. For example, in the constant multiplication ($C \times X$), the reduction process is realized in software, while in the variable multiplication ($Y \times X$), it is implemented in hardware. Consequently, the benefit of reducing the number of digits may be outweighed by the absorption effort, requiring too much computational-time (software), or too many hardware resources (Table 3.6). This issue will be carefully studied in the two subsequent chapters, based on effective software/hardware implementations.

TABLE 3.6 – Main features of number systems.

Number System	Weighted System	Radix System	Fixed/Mixed Radix System	Canonical Form	Ease of use & Implementation	Frequently Used	Hardware Optimization
Binary	Yes	Yes	Fixed	Yes	A	A	E
SD	Yes	Yes	Fixed	Yes	B	A	D
CSD	Yes	Yes	Fixed	Yes	C	A	C
DBNS	Yes	No	–	U*	E	C	B
RNS	Yes	Yes	Mixed	U*	F	D [3]	D
Radix-2^r	Yes	Yes	Fixed	U*	D	B	A

A–F: A, the best; F, the worst. U: Unknown. *: the existence of the canonical form has to be proved.

3.5 Conclusion

The binary arithmetic is a vast topic. That is why we have deliberately restricted our review to the main ideas and essential concepts directly involved in the design of FWL linear systems. We focused more particularly on the fixed-point arithmetic and the main number systems relying on it. Special care has been devoted to the addition and multiplication operations as they are the two main building-blocks of any linear-system architecture.

In the binary arithmetic, the theory is tightly related to the design aspect. Some solutions are theoretically very attractive, but inefficient when it comes to the hardware implementation. A typical example is the RNS arithmetic. We could be fascinated by the speed and simplicity of addition and multiplication in RNS, but when considering the necessary conversions from binary to RNS and vice-versa, we are rapidly discouraged by the extra-amount of hardware needed. Being aware of such misleading choices due to an anterior experience in arithmetic design, we have given priority to simplicity and ease-of-use over potentially-efficient but difficult-to-implement solutions.

It is well-known that CSD arithmetic is employed in designing the vast majority of linear systems (controllers and DSP). As radix-2^r is the generalization of SD arithmetic (r=1) and more likely the generalization also of CSD (the mathematical proof is missing for the time being), it *could* lead to higher speed and less logic resources than CSD since r bits are processed simultaneously. However, the major drawback of radix-2^r is that 2^{r-2} odd-numbers are necessary, which outweighs the speed and area benefits. Consequently, the odd-numbers in radix-2^r arithmetic constitute a *research-lock* that deserves a special attention. This issue is deeply investigated in the subsequent chapters.

Bibliography

[1] M.D. Ercegovac and T. Lang, "Digital Arithmetic," Morgan Kaufman Publishers, an Imprint of Elsevier Science, ISBN: 1-55860-798-6, San Francisco, USA, © 2004.

[2] J.M. Muller et al. "Handbook of Floating-Point Arithmetic," Birkhäuser Boston, a part of Springer Science+Bussiness Media, ISBN: 978-0-8176-4704-9, © 2010.

[3] B. Perhami, "Computer Arithmetic, Algorithms and Hardware Design," Oxford University Press, ISBN: 0-19-512583-5, New-York, USA, © 2000.

[4] R. Kastner, A. Hosangadi, and F. Fallah, “Arithmetic Optimization Techniques for Hardware and Software Design,” Cambridge University Press, ISBN-13 978-0-521-88099-2, © 2010.

[5] B. Lopez, T. Hilaire and L.S. Didier, “Sum-of-products Evaluation Schemes with Fixed-Point arithmetic, and their application to IIR filter implementation,” Proceedings of the International Conference on Design and Architecture for Signal and Image Processing (DASIP), Karlsruhe, Germany, Oct. 2012.

[6] A. Avizienis, “Signed-digit number representation for fast parallel arithmetic,”. IRE Trans. on Electronic Computers, vol. EC-10, No. 3, pp. 389–400, September 1961.

[7] V.S. Dimitrov, G.A. Jullien, and W.C. Miller, “Theory and Applications of the Double-Base Number System,” IEEE Trans. on Computers (TC), vol. 48, No. 10, pp. 1098-1106, October 1999.

[8] H. Sam, and A. Gupta, “A Generalized Multibit Recoding of Two’s Complement Binary Numbers and its Proof with Application in Multiplier Implementation,” IEEE Trans. on Computers, vol. 39, N° 8, August 1990.

Chapter 4

Multiplication by a Constant

Chapter 4

Multiplication by a Constant

This chapter addresses the problem of hardware optimization of linear-time-invariant (LTI) systems. It provides a thorough discussion of the issues surrounding the multiplication by a constant, focusing mainly on the single/multiple constant multiplication (SCM/MCM). We formalize the SCM/MCM problem, and present a list of the most common solutions. We then introduce a new fully predictable SCM/MCM heuristic, accompanied with its upper-bound, average, and adder-depth complexities. Further optimizations of the proposed heuristic are also provided and compared to existing ones.

4.1 Optimizations of LTI Systems

A linear system is a mathematical model based on linear operations. Linear operations adhere to two properties, namely additivity and homogeneity. Given two vectors x and y, and a scalar c, these properties are formally described as:

- Additivity: $f(x+y)=f(x)+f(y)$; (4.1)
- Homogeneity: $f(c\times x)=c\times f(x)$. (4.2)

Another way to state this is that for vectors x_n and scalars c_n the following equality holds:

$$f(c_1\times x_1+c_2\times x_2+\cdots+c_n\times x_n)=c_1\times f(x_1)+c_2\times f(x_2)+\cdots+c_n\times f(x_n). \quad (4.3)$$

Linear systems have roles as mathematical abstractions or models of computation in many applications, including mainly: automatic control theory, signal processing, and telecommunications. The primary focus of this chapter is on control theory; however, the techniques are applicable across any application that computes linear systems.

4.1.1 Formulation of LTI Systems

Based on equation (4.3), an LTI system is formalized as follows. If X and Y are input and output vectors, respectively, and C is a transformation matrix, the LTI system can be written as

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & \dots & C_{1n} \\ C_{21} & C_{22} & \dots & C_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ C_{m1} & C_{m2} & \dots & C_{mn} \end{bmatrix} \times \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix}. \quad (4.4)$$

The transformation matrix C is an $m\times n$ matrix, where C_{ij} represents the (i,j) constant. An output signal Y_i is the product of the ith row of the transformation matrix C and the n input samples of X:

$$Y_j = \sum_{j=1}^{n} C_{ij} \times X_j \,. \tag{4.5}$$

In Chapter 2, the state-space representation has been retained as the control computational-model to be optimized. Note that the explicit state-space LTI Eq. 2.1 is not more than a concatenation of four $Y=C\times X$ operations given by Eq. 4.4. As illustrative example, we have already provided the discrete time state-space representation of a cantilever controller (Fig. 2.6 and Eq. 2.3).

4.1.2 Single-Constant Multiplication (SCM)

The constant multiplication problem first emerged in the 1970s for implementing constant multiplication *in software.* Many microprocessors at the time (such as Intel's 8008) did not have multipliers, so multiplication had to be done with additions, subtractions, and shifts. Even when the multiply instruction first became available in software, it would typically take more clock cycles to execute than an addition, subtraction, or shift, thus solving the constant multiplication problem could lead to a reduction in execution time. Today, with high throughput pipelined multipliers, solving the constant multiplication problem provides no benefit in software.

Conversely, the whole benefit is rather in hardware implementation. The idea is to propose algorithms which *run in software,* but the solutions that these algorithms produce enable one to efficiently implement constant coefficient multiplication *in hardware.* Given a set of constant coefficients, the proposed algorithms *search* for good hardware realizations.

In the whole Eq. 4.4, SCM addresses only the problem of multiplication of one variable (X_j) by one constant (C_{ij}). To be efficiently handled, the hardware implementation of $C_{ij}\times X_j$ must be *multiplierless*, that is, using exclusively additions, subtractions, and shifts. The shift operation is “free” in a *hardware* implementation (it only involves rewiring), therefore, only the number of additions and subtractions is important. We assume that additions and subtractions have the same area/speed cost, which is a quite reasonable assumption in hardware design. Careful decomposition into shifts and additions leads to tremendous benefits with respect to *execution time, area, throughput, and power/energy.*

Actually, the computational complexity of SCM still seems to be unknown. It is only conjectured (no proof) to be NP-Hard [1]. But because the solution-space to explore is so huge, one has to use heuristics to minimize the number of additions.

Illustrative Examples

Let us perform for instance $45\times X_j$. While the solution space allows many possibilities, only four solutions are presented:

$$45\times X_j = (101101)_2 \times X_j = X_j\times 2^5 + X_j\times 2^3 + X_j\times 2^2 + X_j \;; \tag{4.6}$$

$$45\times X_j = (63-18)\times X_j = \left[(111111)_2 - (010010)_2\right]\times X_j = \left(X_j\times 2^6 - X_j\right) - X_j\times 2^4 - X_j\times 2 \;; \tag{4.7}$$

$$45\times X_j = \left(10\bar{1}0\bar{1}01\right)_2 \times X_j = X_j\times 2^6 - X_j\times 2^4 - X_j\times 2^2 + X_j \;; \tag{4.8}$$

$$45\times X_j = U\times 2^2 + U \quad \text{with} \quad U = X_j\times 2^3 + X_j \,. \tag{4.9}$$

Eq. 4.6 is the most straightforward method for single-constant multiplication. It transforms the constant C_{ij} into its binary representation, converts the 1s into shifts based on their positions, and sums the shifted values. For C_{ij}=45, Eq. 4.6 requires three additions and three shift operations. The number of additions using the binary representation is one less than the number of "1" instances.

Another method (Eq. 4.7) uses shifts and subtractions by translating the "0" values into shift operations while subtracting them from the constant of the same length consisting of only 1s. The constant 45 requires six bits and the corresponding six-bit constant of all 1s (111111) is 63. The term $\left(X_j \times 2^6 - X_j\right)$ represents the number 63 and the following two terms, $X_j \times 2^4$ and $X_j \times 2$, represent the terms 16 and 2, respectively (16+2=18).

The CSD representation (Eq. 4.8) encodes a constant number using the minimal number of nonzero digits. Therefore, when transforming a constant multiplication into a sequence of shifts and additions, the CSD representation yields the minimum number of additions. But in this special case (C_{ij}=45), does not provide any benefit over Eq. 4.6 and 4.7.

Eq. 4.9 allows to obtain the minimal number of addition for C_{ij}=45. The reduction in additions comes from the sharing of the term $U = 9 \times X_j$. This is well illustrated by Fig. 4.1b, where the grey nodes indicate a shift operation, and the red ones denote addition. The total number of operations is two additions. Note that several solutions with 2 additions might exist (Fig. 4.1a, 4.1b, and 4.1c).

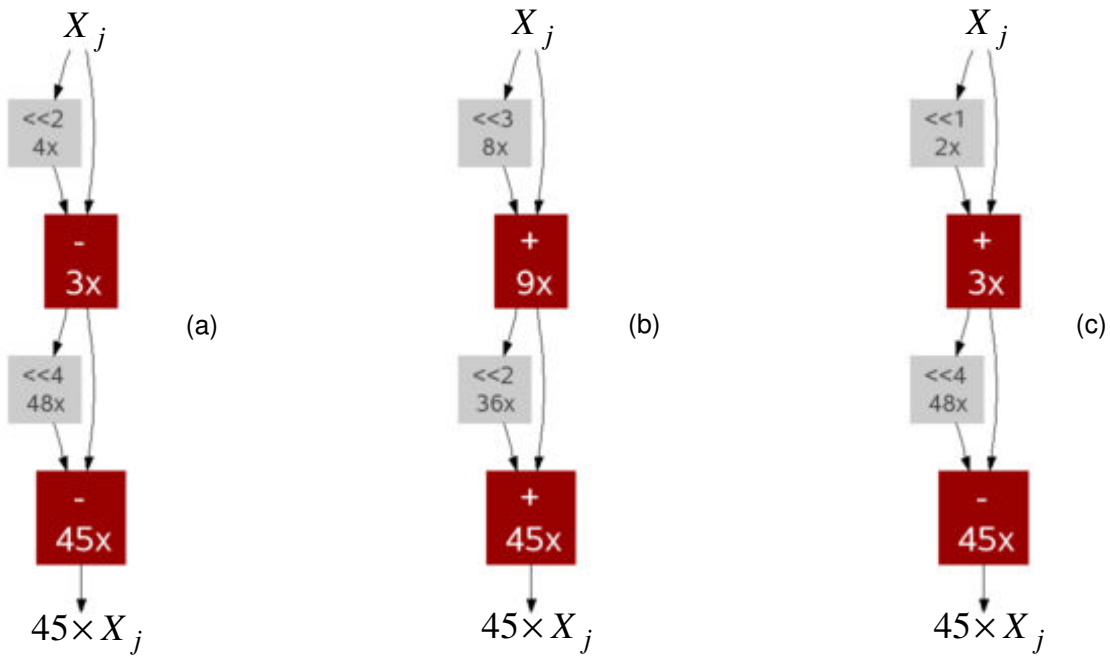


FIGURE 4.1 – The minimal number of addition for $45 \times X_j$. The solutions are given by the Spiral web site (www.spiral.net). The sign "<<a" means a shift of a positions (×2^a).

The objective of SCM heuristic is to provide optimal solutions in a reasonable computational time. Predictability is another important feature of SCM heuristic. Depending on the constant bit-size, it allows to know in advance (before implementation) the maximum number of additions (area) and the maximum number of additions forming the critical path (speed). Despite the big number of proposed heuristics, only three SCM heuristics are predictable. This issue will be thoroughly investigated in the coming sections of this chapter.

4.1.3 Multiple-Constant Multiplication (MCM)

Consider the linear Eq. 4.4; MCM is an extension of SCM where the single variable X_j is multiplied by the set of constants $C_{1j}, C_{2j}, C_{3j}, \cdots, C_{mj}$ (the entire column j). Defined as such, the MCM problem is at least as hard as the SCM problem since the latter is a subset of the MCM problem; since the single-constant multiplication problem is conjectured NP-Hard, the MCM problem is also conjectured NP-Hard. One potential solution to the MCM problem is simply to optimize each single-constant multiplication independently. However, in general, by sharing the intermediate computations required by each constant, MCM can be decomposed into fewer operations than the total number of SCM operations that would be needed.

Illustrative Example

Let us perform the two following multiplications: $81 \times X_j$ and $23 \times X_j$. Fig. 4.2a and 4.2b depict the best optimization for each case individually. The nodes denote addition, while the edges show the required amount of shifting. Fig. 4.2a and 4.2b require two additions, resulting in four additions to perform both multiplications. Fig. 4.2c shows the simultaneous optimization of the two variables. The variables can share a common multiplication 9x, hence, the overall number of additions for both variables is reduced by one (i.e., total of three additions).

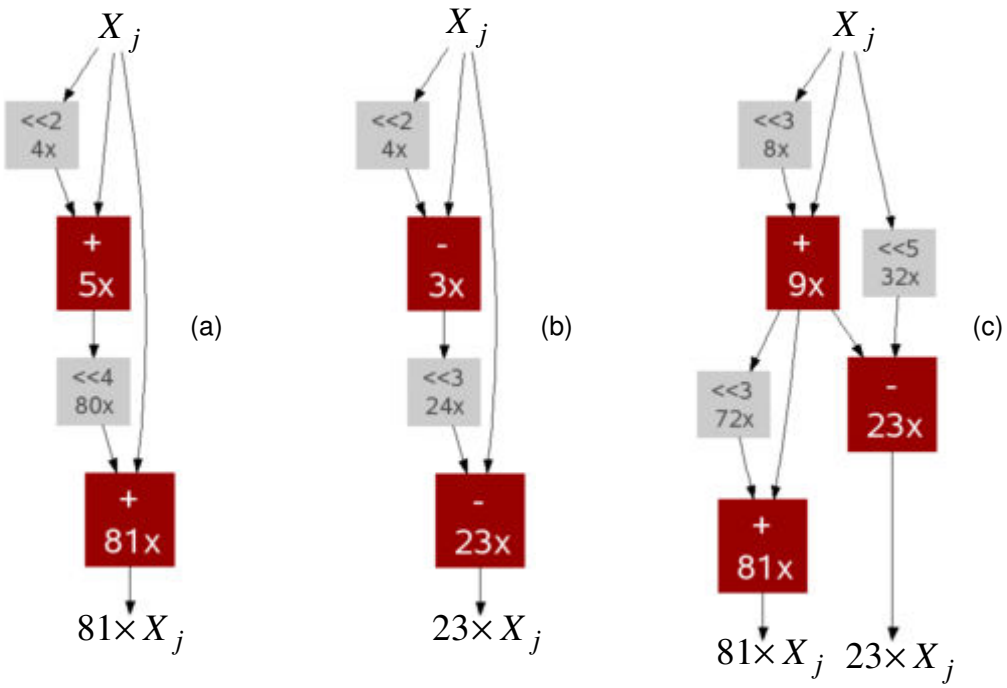


FIGURE 4.2 – Multiplication of the constants 81 and 23. (a), (b) The optimization of each variable independently; this requires two additions per constant for a total of four additions. (c) The simultaneous optimization of both variables. The variables can share one addition resulting in three additions for the multiplication of both constants.

Obviously, the optimization of the linear Eq. 4.4 taken as a whole is much better using MCM than SCM. However, this added performance is obtained at the expense of an extra computational time which may be prohibitive in case of an important number of constants. To the best of our knowledge there is no predictable MCM heuristic.

4.1.4 Subexpression Sharing between Output Variables

MCM addresses only a part of the problem. It involves the optimization of the multiplication of each input variable X_j by its corresponding constants of the column j of Eq. 4.4. Optimizations can also be carried out considering common subexpressions between output variables

$$Y_i = C_{i1} \times X_1 + C_{i2} \times X_2 + \cdots C_{in} \times X_n, \quad \text{for } i = 1, m.$$

In either case, i.e. optimization by column or by line of Eq. 4.4, the scope for optimization is limited, making it possible that the algorithm is not taking full advantage of the solution space.

Illustrative Example

Consider the simple example shown in Fig. 4.3. Fig. 4.3a shows a simple linear system with two input and two output variables and a 2×2 constant matrix. The constants are encoded using simple binary representation though they could just as easily use any other number representation. Looking at the common digit patterns in the constants multiplying variable X_2 (7="0111" and 12="1100"), the digit pattern "11" is detected and extracted. This results in a reduction of one operation as shown in Fig. 4.3c. However, if the scope of the optimization is expanded to include multiple variables, the number of operations can be reduced even further as shown in Fig. 4.3d.

(a)

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} 5 & 7 \\ 4 & 12 \end{bmatrix} \times \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

$$5 = (0101)_2$$
$$4 = (0100)_2$$
$$7 = (0111)_2$$
$$12 = (1100)_2$$

(b)

$$Y_1 = X_1 + X_1 \times 2^2 + X_2 + X_2 \times 2^1 + X_2 \times 2^2$$
$$Y_2 = X_1 \times 2^2 + X_2 \times 2^2 + X_2 \times 2^3$$

(c)

$$D_1 = X_2 + X_2 \times 2^1$$
$$Y_1 = X_1 + X_1 \times 2^2 + D_1 + X_2 \times 2^2$$
$$Y_2 = X_1 \times 2^2 + D_1 \times 2^1$$

(d)

$$D_1 = D_2 + X_2 \times 2^1$$
$$D_2 = X_1 + X_2$$
$$Y_1 = D_1 + D_2 \times 2^2$$
$$Y_2 = D_1 \times 2^2$$

Source: [2]

FIGURE 4.3 – An example that shows the benefit obtained by considering common subexpressions that span across multiple output variables. (a) the example of a linear system; (b) decomposing constant multiplications into shifts and adds; (c) extracting common bit patterns across constants multiplying a single output variable; (d) extending common subexpressions to include multiple output variables.

4.1.5 Matrix Decomposition

The idea [3] consists in modifying the constant coefficient matrices and then performing common subexpression elimination. The modification of the coefficients is based on the fact that the complexity of the multiplier is dependent on the value of the coefficient and correspondingly transforms the linear system by splitting the constant matrices such that the overall area is reduced. The matrix

decomposition technique splits the transformation matrix into the product of 2×z+1 matrices $C = C_z \times C_{z-1} \times C_{z-2} \times \cdots \times C_0 \times C_{-1} \times C_{-2} \times \cdots \times C_{-z+1} \times C_{-z}$. The matrix is split through row and column transformations.

Illustrative Example

Fig. 4.4 shows an example of matrix splitting in a linear system. The matrix is split with the intention of reducing the number of multiplications that are required. In the example of Fig. 4.4a, six multiplications are needed, one for each of the values not equal to 1 or –1. Fig. 4.4b splits this matrix into two parts. As a consequence of this splitting, only three multiplications are now required, two multiplications for the two occurrences of 0.33 and one multiplication for –0.66. Note that the 1, –1, and –2 values can be converted to addition, subtraction, and shift, respectively. The matrix splitting is followed by a decomposition of the multiplications into shifts and additions along with an algorithm to eliminate common subexpressions.

(a)

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 & 0.33 \\ -1 & -0.66 & 0.33 & -0.33 \\ 1 & 0.33 & -0.66 & 0 \end{bmatrix} \times \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$

(b)

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & -1 & 0.33 \\ -2 & 0 & 0 & 0 \\ 1 & 0.33 & -0.66 & 0 \end{bmatrix} \times \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$

Source: [2]

FIGURE 4.4 – Matrix decomposition. The constant matrix in (a) is split into two matrices in (b). This results in a reduction of multiplications for the overall system. Part (a) requires six multiplications corresponding to the six positive and negative values of 0.66 and 0.33. Part (b) requires only three multiplications for the two values of 0.33 and the one value of –0.66.

The aforementioned approaches attempt to minimize the number of additions/subtractions required to compute the constant multiplications. In fact, the problem is even *more difficult* than this. Every addition is not equivalent. The size and the speed of a specific addition highly depend on the numbers being summed. For example, consider summing X+4×X, where the variable X has two bits. Simply concatenating the variable X with itself will perform this addition. If X=(01)$_2$, then X+4×X =(0101)$_2$, which is simply the string “01” concatenated with itself, i.e., replicated twice. The higher-order two bits correspond to 4×X and the lower two bits represent X. This requires no hardware and no delay for computation. Therefore, not all additions are the same; thus, it is possible to reorder additions to take advantage of this fact. This problem is not treated in this thesis. However, all the effort is rather concentrated on the optimization of SCM/MCM problem.

4.2 Formal Definition of the SCM Problem

Prior formalizing the SCM Problem, we must clearly define:

- The constant type: positive, negative, odd, or even. Almost all proposed SCM heuristics handle only odd/positive constants as the even/negative ones can be simply derived using negation and shift. Note that in this case an extra-processing of the constants is necessary.
- The allowed operations: addition, subtraction, left-shift, right-shift, and "or" operation. Restricting the number of operations makes the problem harder to solve. Most of the proposed heuristics allow only addition, subtraction, and left-shift. Just few heuristics [1] allows right-shift. But right-shift has some drawbacks, while bringing little value: for instance, they do not work with arithmetic modulo 2^k.

The following formal definition apply to all types of constants, i.e., positive/negative and odd/even, allowing addition, subtraction, and left-shift only.

Let c be our constant, $c \in \mathbb{Z}$. A finite sequence of ***signed*** integers u_0, u_1, u_2, . . . , u_q is said to be acceptable for c if it satisfies the following properties:

- Initial value: $u_0 = 1$;
- For all i>0, $u_i = s_i \times u_j \times 2^{a_i} + r_i \times u_k \times 2^{b_i}$; with $j,k < i$; $s_i, r_i \in \{-1,0,1\}$; and $a_i, b_i \in \mathbb{N}$;
- Final value: $u_q \times 2^{c_q} = c$, with $c_q \in \mathbb{N}$.

The problem is to find an algorithm that takes the number c and that generates an acceptable sequence $(u_i)_{0 \le i \le q}$ that is as short as possible. q is called the *quality*, or *length*. Note that by delaying the shifts, we can restrict b_i to 0 (this breaks the symmetry, but makes one less variable).

Thus, an arbitrary number X being given, the corresponding solution iteratively computes $u_i \times X$ from already computed values $u_j \times X$ and $u_k \times X$, till finally obtain $c \times X$. Note that with this formulation, we are allowed to reuse any already computed value. Therefore some generated solutions may need to store temporary results.

SCM/MCM is a fundamental problem in control, DSP, and telecommunications. Because of this, a big number of heuristics have been proposed. Only the most cited ones are summarized hereafter.

4.3 Existing SCM/MCM Algorithms

The existing MCM algorithms can be divided into four general classes:

- Digit-recoding algorithms such as the canonical signed digit (CSD) representation [4], Booth recoding [5], and Dimitrov's DBNS recoding [6];
- Common subexpression elimination (CSE) using pattern matching performed after an initial digit-recoding. Typical examples are Hartley [7], Lefèvre [8], and Boullis [9];
- Directed acyclic graph (DAG) based algorithms. This category includes Bernstein [10], MAG [11], H(k) [12], and Hcub [13];
- Hybrid algorithms combining CSE and DAG such as the recent optimal algorithm BIGE [1].

Interesting surveys and detailed comparative studies showing pros and cons of various algorithms are given in [1][13].

4.3.1 Digit-Recoding Algorithms

This category includes simple methods like CSD and the traditional binary method. They generate the decomposition directly from the digit representation of the constant. These methods are the easiest to implement but achieve the worst results. A drawback of digit-recoding methods is the impossibility to reuse intermediate values. Nevertheless, the main advantage of digit-based recoding is their low computational cost, typically linear in the number of bits. As a consequence, these methods can be easily applied to constants with thousands of bits. In addition, these methods allow an easy control of lower bounds on the maximum left-shifts to ovoid overflow.

4.3.1.1 Avizienis's CSD Algorithm

CSD [4] has the least number of nonzero digits among all possible SD representations, therefore it creates a solution with the smallest number of additions/subtractions for a single constant. The CSD representation is unique, and therefore non-redundant. For a n-bit constant, the number of additions is bounded by (n+1)/2−1 in the worst case, and is roughly equal to n/3 on average (the exact value is n/3–8/9). CSD recoding is not optimal though it minimizes the number of nonzero elements for the constant representation. It is possible to decompose the result to further reduce the number of operations. The value 45 is the smallest number for which the CSD representation does not create the optimal number of operations. Three additions are required by CSD (Eq. 4.8) while the optimal solution (Eq. 4.9) needs only two.

CSD has been thoroughly described in Section 3.3.1, and we have come to the conclusion that the *vast majority* of LTI system optimizations use the CSD representation for constant encoding [2]. It is important at this stage to illustrate how CSD is used in practice. Let us consider for instance the following LTI system:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0.926 & 0.383 & -0.383 & -0.926 \\ 0.707 & -0.711 & -0.711 & 0.707 \\ 0.383 & -0.926 & 0.926 & -0.383 \end{bmatrix} \times \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}. \tag{4.10}$$

The constants of the transformation matrix C are first scaled to convert them into integers using nine bits. For example, to 1 we associate the value 256, which is the largest representable nine-bit integer number. Thus, for the constant 0.926 corresponds the value 256×0.926≈237.Transforming the remainder of the numbers yields the constant matrix:

$$C = \begin{bmatrix} 256 & 256 & 256 & 256 \\ 237 & 98 & -98 & -237 \\ 181 & -182 & -182 & 181 \\ 98 & -237 & 237 & -98 \end{bmatrix}. \tag{4.11}$$

Next, we need to encode each of these constants using CSD. It results in:

$$C_{CSD} = \begin{bmatrix} 100000000 & 100000000 & 100000000 & 100000000 \\ 100\bar{1}0\bar{1}01 & 010\bar{1}00010 & 0\bar{1}0100\bar{1}0 & \bar{1}00010\bar{1}0\bar{1} \\ 10\bar{1}0\bar{1}0101 & \bar{1}01001010 & \bar{1}01001010 & 10\bar{1}0\bar{1}0101 \\ 010\bar{1}00010 & \bar{1}00010\bar{1}0\bar{1} & 100\bar{1}0\bar{1}01 & 0\bar{1}0100\bar{1}0 \end{bmatrix}. \quad (4.12)$$

The number –98 at position C_{44} is represented as $\left(0\bar{1}0100\bar{1}0\right)_2$. Therefore,

$$\begin{aligned} C_{44} \times X_4 &= -98 \times X_4 \\ &= \left(0\bar{1}0100\bar{1}0\right)_2 \times X_4 \\ &= -128 \times X_4 + 32 \times X_4 - 2 \times X_4 \\ &= -X_4 \times 2^7 + X_4 \times 2^5 - X_4 \times 2 \end{aligned} \quad (4.13)$$

The number of additions required by each element c_{ij} of C_{csd} is equal to the Hamming weight (i.e., the number of non-zero digits of c_{ij} minus 1). Thus, the implementation of C_{csd} requires 34 additions. This number can be drastically reduced applying one of the techniques described above: MCM, subexpression sharing among output variables, or matrix decomposition.

4.3.1.2 Dimitrov's DBNS Algorithm

Given p, q, two distinct prime integers; the double-base number system (DBNS) is a representation scheme into which every positive integer n is represented as the sum or difference of numbers of the form $p^a \times q^b$ as follows [6]:

$$n = \sum_{i=1}^{l} s_i \times p^{a_i} \times q^{b_i}, \quad (4.14)$$

with $s_i \in \{-1,1\}$, $a_i, b_i \geq 0$ and $(a_i, b_i) \neq (a_j, b_j)$ for $i \neq j$. The length of a DBNS decomposition is equal to the number l of terms in Eq. (4.14). In the following, we will only consider p=2 and $q \in \{3,5,7\}$. DBNS representation system has already been described in Section 3.3.2, but for p=2 and q=3 only.

Whether one considers signed (s_i = ±1) or unsigned (s_i =1) expansions, this representation scheme is highly redundant. Indeed, if one considers unsigned double-base representations (DBNR) only, with bases 2 and 3, then one can prove that 10 has exactly 5 different DBNR; 100 has exactly 402 different DBNR; and 1000 has exactly 1295579 different DBNR. The following theorem gives the number of unsigned DBNR for a given number n:

Theorem 4.1– *Let n be a positive integer and let q be a prime > 2. The number of unsigned DBNR of n with bases 2 and q is given by $f(1)=1$, and for $n \geq 1$* $f(n) = \begin{cases} f(n-1) + f(n/q) & \text{if} \quad n \equiv 0 \bmod(q), \\ f(n-1) & \text{otherwise.} \end{cases}$

The proof of Th. 4.1 is given in [6]. Not only DBNS is highly redundant, but it is also very *sparse*.

Th 4.1 tells us that there exists very many ways to represent a given integer in DBNS. Some of these representations are of special interest, most notably the ones that require the minimal number of

$\{p,q\}$-integers; that is, an integer can be represented as the sum of m terms, but cannot be represented with $(m-1)$ or fewer terms. These so-called *canonic* representations are extremely sparse. For example, with bases 2 and 3, Th. 4.1 tells us that 127 has 783 different unsigned representations, among which 6 are canonic requiring only three {2, 3}-integers. Finding one of the canonic DBNS representations in a reasonable amount of time, especially for large integers, seems to be a very difficult task. Fortunately, one can use a greedy approach to find a fairly sparse representation very quickly. Given $n > 0$, Alg. 4.1 returns a signed DBNR for n.

Although Alg. 4.1 sometimes fails in finding a canonic representation (the smallest example is 41; the canonic representation is 32 + 9, whereas the algorithm returns 41 = 36 + 4 + 1) it is very easy to implement. The complexity of the greedy algorithm mainly depends on the complexity of step 3: finding the best approximation of n of the form $p^a \times p^b$. It finishes in $O(\log(n)/\log(\log(n)))$ iterations.

ALGORITHM 4.1 – Greedy algorithm.

INPUT: A positive integer n

OUTPUT: The sequences $(s_i, a_i, b_i)_{i\geq 0}$ for $n = \sum_i s_i \times p^{a_i} \times q^{b_i}$

with $s_i \in \{-1,1\}$, $a_i, b_i \geq 0$ and $(a_i, b_i) \neq (a_j, b_j)$ for $i \neq j$.

1: $s \leftarrow 1$ // to keep track of the sign

2: while $n \neq 0$ do

3: Find the best approximation of n of the form $z = p^a \times p^b$

4: print (s, a, b)

5: if $n < z$ then

6: $s \leftarrow -s$

7: $n \leftarrow |n - z|$

In the following, a generic algorithm for constant multiplication that takes advantage of the sparseness of the double-base encoding scheme is presented (Dimitrov's algorithm [6]). The algorithm computes a special DBNS representation of the constants, where the largest exponent of the second base q is restricted to an arbitrary (small) value B. It uses a divide and conquer strategy to operate on separate blocks of small sizes. For each block, it is possible to generate those specific DBNS representations using the Greedy algorithm.

Before giving the formal description of Dimitrov's algorithm, it is first illustrated on a small example. The value c = 10599 = $(10100101100111)_2$ is expressed in radix 2^7; that is, c is split in two blocks of 7 bits each. We obtain c = 82×27+103. The "digits" 82 and 103 are expressed in DBNS with bases 2 and 3 using as few terms as possible, where the exponents of the second base q=3 are at most equal to 2. Using Alg. 4.1, it has been found that 82 can be written using two terms as 64+18 and 103 using only three terms as 96+8–1. These two solutions are optimal. By sticking the two parts together, we obtain the representation given in Table 4.1.

TABLE 4.1– A DBNS representation of c = 10599 obtained using two blocks of 7 bits each.

	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}
3^0	-1			1										1
3^1						1								
3^2									1					
	103							82						

Using this representation, the product $c \times X$ is decomposed in radix-2^7 DBNS as follows:

$$P_{DBNS}=((X_1\times 2^1)+X_1)+(X\times 2^{13})+(X\times 2^3)-X \;; \tag{4.15}$$

with $X_1=((X_0\times 2^1)+X_0)+(X\times 2^5)$ and $X_0=(X\times 2^8)$.

The formal formulation of Dimitrov's algorithm is as follows. The constant c is written in DBNS as:

$$c=\sum_{j=0}^{b_{\max}}\sum_{i=0}^{a_{\max}} c_{i,j}\times 2^i\times q^j\,, \tag{4.16}$$

with digits $c_{i,j}\in\{\bar{1},0,1\}$. Alg. 4.2 below can be used to compute $c \times X$. We remark that each step of the algorithm requires a multiplication by q. It is therefore important to select the second base q such that the multiplication by q only requires a single addition; i.e., with q=3, we have $3\times X = X\times 2+X$. At the end, the result is given by $X_{b_{\max}} = c\times X$. If l is the length of the double-base expansion; i.e, the number of non-zero digits $c_{i,j}$ in Eq. 4.16, and if b_{max} is the largest exponent of q, then the overall number of additions is equal to $l+b_{\max}-1$. The goal is to set B, the predefined upper bound for the exponents of q, such that the overall number of addition is minimal. Note that b_{max} might be different from B, but $b_{\max}\le B$.

ALGORITHM 4.2 – Double base constant multiplication.

INPUT: A constant $c=\sum_{i,j} c_{i,j}\times 2^i\times 3^j$, with $c_{i,j}\in\{\bar{1},0,1\}$;
and an integer X

OUTPUT: $c \times X$

1: $X_{-1}\leftarrow 0$

2: for j=0 to $b_{\max}$ do

3: $X_j \leftarrow q\times X_{j-1}$

4: $X_j \leftarrow X_j+\sum_i c_{i,b_{\max}-j}\times X\times 2^i$

5: Return $X_{b_{\max}}$

It has been proved in [6] that the number of additions generated by Alg. 4.2 is bounded by $O(n/\log(n))$. While this limit shows that multiplication by a constant is *sublinear*, it says nothing about the constant hidden within the big-O notation, which might be very important by the way. In 2011, Dimitrov [14] evaluated the hidden constant as being equal to 2. Since then, $2n/\log(n)$ is considered as the lowest analytic upper-bound known so far for the multiplication by a constant.

4.3.2 Common Subexpression Elimination (CSE)

As previously mentioned, the major drawback of digit-recoding methods is their inability to reuse intermediate values. To solve this problem, the basic idea is to find common subpatterns in the representations of the constants after the constants are converted to a convenient number system such as SD or CSD. Thus, CSE are considered as direct descendants of digit-recoding methods. The typical example is given by Eq. 4.9 where the use of a common term allows the saving of one addition over CSD (Eq. 4.8).

A significant drawback is that the performance of CSE algorithms depends on the number representation. Some works [12][15] showed how using different representations for the constant encodings can lead to better results. However, the exponential number of representations further complicates the CSE problem which is itself conjectured to be NP-complete. Even if the solution to the CSE problem is optimal, it does not necessarily provide an optimal solution to the SCM/MCM problem. We will begin by examining some cases in which CSE algorithms cannot find the optimal solution.

4.3.2.1 Optimization Problems Due to the Initial SD Form

As previously stated, Park and Kang [15] found better solutions by applying CSE to all MSD forms of the constant (SD forms of the constant with the minimum number of nonzero digits). Even better solutions are obtained with H(k), which applies CSE to all SD forms of the constant with up to k more digits than the CSD form. For example, the CSD form of 105xX, $X0\overline{X}0X00X$, has no patterns that occur at least twice. Without being able to factor common terms, 3 adders are needed to add the 4 terms. However, one MSD form of 105xX is $X00\overline{XX}00X$, in which the pattern $Y = X00\overline{X}$ can be substituted twice to yield $000Y00\overline{Y}$. In this case, 2 adders are used (one to create Y and one to add the remaining terms). The problem was due to the CSD form being constrained to have no adjacent nonzero digits. Notice the leftmost nonzero digit of $000\overline{X}000X$ is adjacent to the rightmost nonzero digit of $X000\overline{X}000$, which obviously cannot be represented by the CSD form.

The first case in which H(0) produces a non-optimal solution is 363xX. There are only two MSD forms, $X0\overline{X}00\overline{X}0\overline{X}0\overline{X}$ and $XX00\overline{X}0\overline{X}0\overline{X}$, neither of which have a common pattern, thus H(0) requires 4 adders. However, H(1) can find the optimal solution by using the representation $X0XX0X0XX$. The pattern $Y = X0000X$ is substituted to produce $00000Y0YY$, thus the total cost is 3 adders (1 to create Y, 2 to add the remaining terms). If a pattern with 2 nonzero digits is substituted n times, a total of $2n$ old digits will be replaced with n new digits. This results in a saving of $n-1$ adders.

4.3.2.2 CSE digit clashing problem

The first case in which H(k) produces a non-optimal solution *for any value of* k is 805xX. The optimal solution requires 3 adders. As indicated in Section 4.3.1.1, if we start with more than 8 nonzero digits, then more than 3 adders will be needed by CSD. The CSD form of 805xX has 5 nonzero digits, thus we only need to consider all SD forms with up to 3 extra nonzero digits. H(3) cannot find the optimal solution, and this is sufficient proof that H(k) is not an optimal algorithm *for any value of* k.

In order to find the optimal solution for 805xX, we will need to consider an unusual case. Notice that $((X0X)\times 2^2)+X0X = XX00X$. This translates to $(5\times X)\times 2^2+(5\times X)=25\times X$. The left digit of $X0X$ aligns with the right digit of $(X0X)\times 2^2$ to produce a zero in this position and a carry one position to the left. Thus, if $Y = X0X$, we can substitute Y in $XX00X$ to get $00Y0Y$ even though $X0X$ does not appear at either location of where Y was substituted. This is an example of a class 1 overlapping digit pattern. Now consider representing 805xX as $XX00X00X0X$. We substitute $Y = X0X$ to get $00Y0Y0000Y$, thus only 3 adders are needed (1 to make Y, 2 to add the remaining terms). This digit alignment problem was recognized and identified as *clashing.* We thus refer to this problem as the *CSE digit clashing problem.*

4.3.2.3 Lefèvre's Common Subpattern (CSP) Algorithm

In Lefèvre's CSP algorithm [8], the number of nonzero digits is called the weight (w). The digit –1 and 1 are denoted P and N respectively. Given a set of constants in SD representation, the problem is to find the maximal-weighted pattern that appears twice, either in the same constant (SCM) or in two different constants (MCM). Note that if such a pattern appears more than twice, it can naturally be reused later.

CSP algorithm looks for a triple (i, j, d) where i and j are numbers identifying the two constants n_i and n_j (possibly the same, i.e., i=j) in the set, and d a distance (or shift count), with the following restrictions for symmetry reasons: $i \le j$; and if $i = j$, then $d > 0$. Once the pattern, denoted n_p, of weight greater or equal to 2 has been found, we compute the binary expansion of the new constants n_i' and n_j' (n_i' only, if $i = j$) satisfying $n_i = n_i' \pm 2^c \times n_p$ and $n_j = n_j' \pm 2^{c-d} \times n_p$ for some integer c (or $n_i = n_i' \pm 2^c \times n_p \pm 2^{c-d} \times n_p$ if $i = j$), then we remove n_i and n_j from the set, then add n_i', n_j' and n_p to the set (with the exception that 0 and 1 are never added to the set, because they cannot yield a pattern). We reiterate until there are no more patterns of weight greater or equal to 2; we use the binary method for the remaining constants.

Now, let us illustrate this algorithm more precisely with an example: $(47804853381)_{10}$, which is written $(10110010000101100100000101100100000101)_2$ in binary, and after Booth's recoding: P0N0N00P000P0N0N00P000P0N0N00P0000P0P. We find the triple (0, 0, 11), the corresponding pattern is P0N0N00P, and the remainder is P0N0N00P00000000000000000000000000P0P (there would be other choices). The set is now:

{ P0N0N00P00000000000000000000000000P0P, P0N0N00P }.

We find the triple (0, 1, 29), the corresponding pattern is P0N0N00P (i.e., still the same one), and the remainder is P0P. The set is now: { P0P, P0N0N00P }.

We find the triple (0, 1, –3), the corresponding pattern is P0P, and the remainder is P000000P. The set is now: { P000000P; P0P }.

Now, we cannot find any longer repeating pattern whose weight is greater or equal to 2. Therefore we use the binary method for the remaining constants.

The computational complexity of Lefèvre's CSP algorithm for a n-bit constant is $O(n^3)$.

4.3.3 Directed Acyclic Graphs (DAG) Algorithms

DAG are bottom-up methods that iteratively construct the graph (as in Fig. 4.2c) representing the multiplier block. The graph construction is guided by a heuristic that determines the next graph vertex to add to the graph. The nodes in the graph denote the coefficients. The edges are directed and represent values that are equal to the coefficients in the parent node shifted by some amount. The coefficient value that is stored at every node is obtained by adding up the values on the incoming edges to the node. Graph-based algorithms offer more degrees of freedom by not being restricted to a particular representation of the coefficients, or a predefined graph topology (as in digit-based algorithms), and typically produce solutions with the lowest number of operations. Examples of DAG algorithms include Bernstein [10] and Voronenko [13] (Hcub Algorithm).

4.3.3.1 Bernstein's Algorithm

In 1986, Bernstein [10] proposed a SCM algorithm. It can be described by the recursive formulas given by Eq. 4.17. Note that every input argument x to the function *Cost(x)* must be an odd integer. If the SCM target t is even, then we must incur an extra cost of *ShiftCost(w)* , where $t/2^w$ is an odd integer (and Bernstein's algorithm would be applied to $t/2^w$). In Eq. 4.17, *a, b,* c, and *d* are integers, $c \geq 1$, and $d \geq 1$.

$$Cost(1) = 0$$

$$Cost(t) = 1 + min \begin{cases} Cost\left((t-1)/2^a\right) + AddCost + ShiftCost(a) \\ Cost\left((t+1)/2^b\right) + SubCost + ShiftCost(b) \\ Cost\left(t/\left(2^c - 1\right)\right) + SubCost + ShiftCost(c) \\ Cost\left(t/\left(2^d + 1\right)\right) + AddCost + ShiftCost(d) \end{cases}. \quad (4.17)$$

The cost of the shifts in Eq. 4.17 is a function of how many bits the operand was shifted. In microprocessors that only support single bit shifts, *ShiftCost(x)* is proportional to x. For microprocessors that support shifting by an arbitrary number of bits, *ShiftCost(x)* is a constant. For hardware implementation, *ShiftCost(x)*=0 and *AddCost*=*SubCost*. Although Eq. 4.17 is expressed in a depth-first manner, the search is computed breadth first since we are interested in finding the minimum cost of t. Bernstein's algorithm is a branch and prune heuristic. The pruning arises from only allowing certain values of *a, b,* c and *d* in Eq. 4.17.

The average time complexity seems to be exponential $O\left(2^n\right)$ [8], where n is the bit-size of the constant. This algorithm is too slow for large constants.

4.3.3.2 Voronenko's Hcub Algorithm

Hcub stands for Heuristic of Cumulative Benefit. It was developed by Voronenko and Püshel [13] in 2007. Hcub is considered as the best heuristic for MCM [1]. The heuristic is split into two parts: the optimal and the heuristic part. It is a *non-trivial* heuristic, requiring a number of definitions.

In Hcub the constants are called the fundamentals. The MCM implementation uses additions, subtractions, and shifts. These three distinct operations are consolidated into a single operation called the A-operation. The A-operation operates on the fundamentals. It performs a single addition or a subtraction, and an arbitrary number of shifts, which do not truncate non-zero bits of the fundamental. Because it is possible to merge two consecutive shifts, the most general definition is stated as follows:

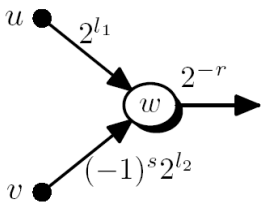
Definition 4.1 – General A-operation: *Let* $l_1, l_2 \geq 0$ *be integers (left shifts),* $r \geq 0$ *be an integer (right shift) and let* $s \in \{0,1\}$ *(sign). An* A*-operation is an operation with two integer inputs* u*,* v *(fundamentals) and one output fundamental, defined as*

$$A_p(u,v) = \left| u \times 2^{l_1} + (-1)^s \times v \times 2^{l_2} \right| \times 2^{-r} \tag{4.18}$$

where $p = (l_1, l_2, r, s)$ *is the parameter set or the* A*-configuration* A_p*. To preserve all significant bits of the output,* 2^r *must divide* $u \times 2^{l_1} + (-1)^s \times v \times 2^{l_2}$ *.*

Hcub considers positive fundamentals only and use right shifts just to normalize the output fundamentals to be odd. The absolute value in the definition of A enforces positive fundamentals, and enables the subtraction to be done in only one direction, which simplifies the definition.

The structure of a multiplier block can be represented as a directed graph. Such graph is called an A-graph, since it is built out of the A-operations shown in Fig. 4.5. The vertices of an A-graph are labelled with their respective fundamentals; hence the input vertex has label 1. The edges are labelled with a 2-power scaling factor equivalent to the performed shift. Negative edge values are used to indicate subtractions at the following vertex.



$$p = (l_1, l_2, r, s)$$
$$w = \mathcal{A}_p(u,v) = |2^{l_1}u + (-1)^s 2^{l_2} v| 2^{-r}$$

Source: [13]

FIGURE 4.5 – A-operation: u and v are the input fundamentals and w is the output fundamental. A-operations directly connected to the input of the multiplier block have $u = v = 1$.

The problem of constructing multiplier blocks can now be formally stated as follows.

Definition 4.2 – MCM problem: *Given a set of positive target constants* $T = \{t_1, \cdots, t_n\} \subset \mathrm{N}$*. Find the smallest set* $R = \{r_0, r_1, \cdots, r_m\}$ *with* $T \subset R$*, such that* $r_0 = 1$*, and for all* r_k *with* $1 \leq k \leq m$ *there exist* r_i, r_j *with* $0 \leq i, j \leq k$*, and an* A*-configuration* p_k *such that*

$$r_k = A_{p_k}(r_i, r_j). \tag{4.19}$$

The set of A-graph fundamentals R and the set of A-configuration p_k uniquely define an A-graph for an MCM block with $m = |R| - 1$ add/sub operations.

Next, the notion of A-distance is introduced. This is the key component in Hcub algorithm.

Definition 4.3 – A-distance: *Let $c \in \mathrm{N}$ be a constant, and let $R \subset \mathrm{N}$ be a set of constants (fundamentals of an A-graph). Then, the A-distance of c from the set R, denoted with dist(R, c), is the minimum number of extra A-operations required to obtain c given R.*

Afterwards, to express the degree of freedom in the output of an A-operation when different A-configurations are chosen, the vertex fundamental set is defined.

Definition 4.4 – Vertex fundamental set: *The set of all possible outputs (not equal to the inputs) of an A-operation with fixed inputs (u and v) under different A-configurations is called the vertex fundamental set, written as*

$$A_*(u, v) = \{A_p(u, v) \mid p \text{ is a valid configuration}\} - u - v. \tag{4.20}$$

Hcub Algorithm

Now that the main ingredients have been defined, let us describe Hcub algorithm. Actually, the main idea behind Hcub is to use a better heuristic for synthesizing intermediate fundamentals. Hcub is therefore computationally more expensive than its DAG counterparts since it explores a very large space of possible intermediate vertices. It does not require a pre-generated optimal SCM lookup table as other algorithms do. Thus, Hcub is storage efficient and in its applicability is only limited by the computation time.

The A-operation in Hcub is A^{odd}, allowing $A_p(u, v) \leq 2^{n+1}$, where n is the maximal bitwidth of constants in T. Hcub is shown in Algorithm 4.3. The heuristic is split into two parts: the optimal and the heuristic part. The optimal part of Hcub synthesizes at each iteration, all distance-1 targets, i.e., S ∩ T. To avoid computing in each iteration the entire set S, which can become rather large, it is computed incrementally. This necessitates an additional set, the worklist W. When a constant is synthesized, it is added to W, first without being accounted for in neither R nor S. Then, in steps 9–10 an incremental update of R and S based on W is performed. The update of R is straightforward (step 9): $R_{new} = R \cup W$.

Alg. 4.3 gives enough details on how to efficiently construct the successor set S. The heuristic part uses S and the A-distance tests and estimators to select new successors s to be added to R. Hcub heuristic part adds only a single successor to R at each iteration (outer loop consisting of steps 5–18).

Given m constants with n bit-size, the computational complexity of Hcub is $O(m^4 n^5 \log(m\,n) + m^3 n^6)$. For SCM, it corresponds to $O(n^6)$.

ALGORITHM 4.3 – Hcub Algorithm. Given the target set of constant T. Compute the set $R = \{r_1, \cdots, r_m\}$, with $T \subset R$, as given in Definition 4.2. There is a degree of freedom in choosing the heuristic function H(R, S, T) for the algorithm.

Synthesize multiplier block (T)

1: Right shift elements of T until odd
2: $R \leftarrow \{1\}$
3: $W \leftarrow \{1\}$
4: $S \leftarrow \{1\}$
5: While $T \neq 0$ do
6: { *optimal part* }
7: While $W \neq 0$ do
8: { *update* S *and* R }
9: $R \leftarrow R \bigcup W$
10: $S \leftarrow \left(S \bigcup A_*(R,W)\right) - W$
11: $W \leftarrow 0$
12: { *if* S *contains targets, synthesize them* }
13: for $t \in S \bigcap T$ do
14: Synthesize(t)
15: { *heuristic part* }
16: if $T \neq 0$ then
17: $s \leftarrow H(R,S,T)$
18: Synthesize(s)

Synthesize(s)

1: $W \leftarrow W + s$
2: $T \leftarrow T - s$

The update formula for S is derived as follows:

$$\begin{aligned} S_{new} &= A_*(R_{new}, R_{new}) = A_*(R_{new}, R \bigcup W) \\ &= A_*(R_{new}, R) \bigcup A_*(R_{new}, W) \\ &= A_*(R \bigcup W, R) \bigcup A_*(R_{new}, W) \\ &= A_*(R, R) \bigcup A_*(W, R) \bigcup A_*(R_{new}, W) - W \\ &= S \bigcup A_*(R,W) \bigcup A_*(R_{new}, W) - W \end{aligned}$$

Since $A_*(R,W) \subset A_*(R_{new}, W)$ we get

$$S_{new} = \left(S \bigcup A_*(R_{new}, W)\right) - W \tag{4.21}$$

which is step 10 in Alg. 4.3.

4.3.4 Hybrid Algorithms(CSE & DAG)

Hybrid algorithms combine different algorithms from different classes. CSE are already considered as mixed algorithms since they rely on a digit recoding followed by a pattern matching. But what is generally meant by "hybrid" in the literature, are those algorithms combining CSE and DAG. A Typical example is the recent Thong's BIGE algorithm [1].

4.3.4.1 Thong's BIGE Algorithm

BIGE was developed by Thong and Nicolici in 2011 [1]. It is the shortened form of Bounded Inverse Graph Enumeration (BIGE) algorithm. It is an exhaustive SCM algorithm, based on the combination of two CSE heuristics (H(k)-ODP) and an optimal DAG algorithm. Overlapping digit pattern (ODP) was introduced in [16] to partially resolve the digit clashing problem. ODP enables CSE algorithms to find and substitute non-standard patterns. For example, if we find $XX00X$, we can substitute Y=X0X to get 00Y0Y even though X0X does not appear in $XX00X$.

An heuristic (H(k)-ODP) is used to obtain an upper bound (the “B” in BIGE) and an exhaustive DAG search to obtain the lower bound. Both bounds are progressively tightened until they meet or until a solution is found by the lower bounding exhaustive search. If any heuristic finds a solution with n adders, the exhaustive search only needs to consider up to n–1 adders. The exhaustive search is done via lookup table for up to 4 adders. For 5 adders and above, inverse graph enumeration (the “IGE” in BIGE) is used to prune the exhaustive search. Prior to describe the high-level operation of BIGE, the following definition is necessary.

Definition 4.5 – Complexity-n constant: *We denote by C_n the set of all constants with complexity n, i.e. those, for which an optimal SCM solution requires exactly n A-operations. For example, $C_0 = \{2^a \mid a \geq 0\}$, because precisely all 2-power constants require a single left shift and no adds/subtracts. Although the sets C_n are infinite, we will always limit our discussion to constants up to certain bitwidth b, which is always explicitly stated. The set of complexity n constants obeying this constraint is then finite and, by abuse of notation, will also be denoted by C_n.*

The C_n sets are independent of the desired SCM constant t and thus are precomputed and stored. Given t, the BIGE algorithm operates as described in Alg. 4.4 and stops as soon as a solution is verified as optimal. For practical reasons, the exhaustive search is limited to 6 adders, although it can be extended.

ALGORITHM 4.4 – The optimal BIGE algorithm.

1) For n = 1, 2, 3, 4 (in that order), if $t \in C_n$ then return the optimal solution from the lookup table.
2) Initialize the upper bound with the H(k)+ODP heuristic [12][16], use k = 1, if this solution has 5 adders it must be optimal.
3) Exhaustively search for a solution with exactly 5 adders.
4) If the solution found earlier by H(1)+ODP had 6 adders, it is now confirmed as optimal.
5) Try to tighten the upper bound with H(2)+ODP, if this solution has 6 adders it must be optimal.
6) Exhaustively search for a solution with exactly 6 adders.
7) If the upper bound is 7, the heuristic solution is now confirmed as optimal.

We have only covered the algorithms which, either are closely related to the contributions in this thesis (CSD, DBNS, Lefèvre's CSP), or the algorithms which proposed the fundamental approaches and ideas (Bernstein, Hcub, BIGE) that were frequently reused and/or improved by later algorithms. In addition to the main features (Table 2.2) of the most known SCM/MCM algorithms in the literature, Table 4.2 recapitulates the information regarding the constant type and allowed operations.

TABLE 4.2 – Constant type and allowed operations.

Algorithm	Constant Type				Allowed Operations			
	Positive	Negative	Odd	Even	Addition	Subtraction	Left-shift	Right-shift
Avizienis's CSD	+	+	+	+	+	+	+	–
Dimitrov's DBNS	+	+	+	+	+	+	+	–
Lefèvre'CSP[1]	+	–	+	–	+	+	+	–*
Bernstein's[1]	+	–	+	–	+	+	+	–*
Voronenko'sHcub[1]	+	–	+	–	+	+	+	–*
Thong's BIGE[1]	+	–	+	–	+	+	+	+

+: Yes; –: No; *: Right-shift is allowed to normalize the constant to be odd, only for that.
1: Some algorithms normalize the negative/even constants to become positive/odd in order to remove redundancy within their search space. This technique improves runtime with no penalty. The sign of the constant can usually be adjusted elsewhere, for instance by using subtraction in the accumulation after multiplication.

4.4 Metrics Definition for SCM/MCM Algorithms

The objective of the proposed software SCM/MCM algorithms is to produce an efficient hardware implementation in a reasonable amount of time. By *efficient* we mean a small area, high speed, and low power architecture. Some abstraction is therefore needed in order to solve problem sizes of the most practical importance within an acceptable amount of time. Logic resources are used as an abstraction of the amount of silicon required to implement a logic function. As for the speed, the abstraction is to consider the longest path that goes through the logic resources.

While there is a direct correlation for area and speed with their respective abstractions, power consumption poses a complex problem. Power consumption decreases only if both the number of logic resources and the length of the critical path decrease. The rational is that there are two sources to power consumption in CMOS circuits: static power-consumption due to the leakage-current; and dynamic power consumption caused by the switching activity. Roughly speaking, the leakage current depends on the area, while the switching activity depends on the topology of the architecture (number of stages of cascaded logic elements). Thus, increasing one element of the power while decreasing the other one results in an unknown consumption state of the power, where only the use of an efficient power-estimation tool can inform on the situation.

Although the number of additions/subtractions is an abstraction of the amount of silicon required to implement the logic circuit, it is conjectured that finding good solutions with this metric typically results in good solutions in terms of minimizing the amount of silicon. This metric is the most commonly used metric in this area of research [1]. Instead of the number of additions/subtractions, the number of single bit adders/subtractors can be used as a more accurate metric (note even this still has some abstraction from the amount of silicon). As shown in the experimental results in [17], a significant amount of extra time is required to solve the same problems using this more accurate metric. This translates into needing impractical amounts of time to solve larger but still real-sized problems. This

also implies that further increasing the metric accuracy will result in longer run times, meaning only smaller problem sizes (which are less relevant in practice) can be solved within reasonable amounts of time. Furthermore, using the amount of silicon as the metric makes the solution dependent on the implementation fabric of the logic circuit, thus the solution would be non-portable and also dependent on the performance of many other computer-aided design (CAD) tools, which perform logic synthesis, place and route, etc.

Adders and subtractors require the same amount of logic resources in custom hardware, so we will simply refer to both as “adders.” Shifts are hardwired and thus incur no cost.

Given a N-bit constant, let us give the formal definitions of the metrics discussed above:

Definition 4.6 – Upper-bound (*Upb*): For each N-*bit constant* C_i, *corresponds* A_i *additions for the implementation of* $C_i \times X$. $Upb = \max(A_i)$.

Definition 4.7 – Adder-Depth (*Ath*): *Let* D_i *be the number of adders that we pass through along any path* i *from the input to any of the outputs in the constant multiplication logic circuit.* $Ath = \max(D_i)$.

Definition 4.8 – Average (*Avg*): *For each* N-*bit constant* C_i, *corresponds* A_i *additions for the implementation of* $C_i \times X$. $Avg = \sum_{i=1}^{m} A_i / m$, *where* m *is the total number of* C_i *constants.*

Actually, though formally defined, the adder depth is an estimate of the longest path through the logic circuit, which is known as the *critical path*. Because of the physical construction, logic gates have a propagation delay, which is the length of time between when stable inputs are asserted to when all of the outputs become stable. As more logic gates are placed serially between the input and the output, the critical path becomes longer and the logic circuit must be clocked at a slower speed, which results in a lower computational throughput. The critical path is also a function of other things like the delay of each gate and the transit time along wires, however we will make abstraction of this in order to solve real-sized SCM and MCM problems within reasonable amounts of time. Most, if not all of the work in this area of research uses the adder depth to estimate the length of the critical path [17]. Given the same problem instance, the number of adders increases as the depth constraint is made smaller. A solution may not exist if the depth is overly constrained. The depth constraint can also be used to prune the search space.

A "reasonable" amount of time is difficult to quantify because it depends on the design flow of the system. For example, if the system is intended to satisfy an existing standard, the constants will be defined and thus each constant multiplication problem only needs to be solved *once* (even if other parts of the system are modified). In this case, one may be willing to wait hours or days for each problem instance. Conversely, a faster algorithm is needed if the design specifications are not finalized (for example, the constants may need to be updated as other parts of the system are modified). Depending on how finalized the design is, one may only be willing to wait a few seconds for each constant multiplication problem, for example. If a large part of the system involves constant multiplication, it is sensible to allocate a large portion of the time in the total CAD flow to solving constant multiplication. In conclusion, a "reasonable" amount of time is highly application specific, as discussed above.

Nevertheless, independently of the design flow, the inherent computational complexity of each proposed algorithm is assessed using the big-O notation as defined in [18].

4.5 Key Limitations of the Existing SCM/MCM Algorithms

The optimization of linear systems is a critical topic which has been the focus of continuous efforts over decades, resulting in an impressive number of SCM/MCM algorithms. Only some ones considered in the literature as milestone algorithms have been described above. However, we shall discuss hereafter some of the limitations and shortcomings of the existing algorithms, where the most important one is the predictability.

4.5.1 Predictability

Despite the large number of proposed heuristics, to our knowledge, only three heuristics are accompanied with their respective addition-cost complexity [1][19]. This issue is very important as it informs on the heuristic capabilities and limitations with regard to the constant bit-size (N). For low values of N (N≤32), H(k) [12] and Hcub [13] are, up to date, considered as the best heuristics for SCM and MCM, respectively. As long as their respective addition complexities are unknown, there is no guarantee that they will preserve their leading positions for high values of N.

It was shown in [20] that the number of additions for an N-bit constant in CSD is bounded by (N+1)/2–1 and tends asymptotically to an average value of (N/3)–8/9, which yields 33% saving over the naive add-and-shift approach. Pinch [21] was the first to prove that the multiplication by a constant is sublinear: $O(N/(\text{Log}(N)^{\alpha}))$ with α<1. Based on the DBNS arithmetic [22], Dimitrov [6] showed that the condition α<1 in Pinch's complexity is not necessary, decreasing therefore the upper limit to $O(N/(\text{Log}(N)))$. Even more, in 2011, Dimitrov [14] evaluated the hidden constant in the big-O notation as being equal to 2. Since then, $2.N/\text{Log}(N)$ is considered as the *lowest* analytic upper-bound known so far. For all remaining heuristics, *no* addition complexity does exist. This is a real handicap as there is no visibility on how the heuristic evolves with respect to N, unless to exhaustively calculate *Upb* and *Avg*, but this is still limited to low values of (N≤32) as an excessive compute power is required.

On the other hand, according to [8], Ross Donelly was the first to determine in 2000 via an exhaustive search that 699829 is the smallest value (20 bits) that can not be obtained with 5 adders or less. Thong [1] did better with the exact BIGE algorithm as he conjectured (no proof) that 7 additions are enough up to 32 bits. Though BIGE guarantees optimality via an exhaustive search, it requires an exponential runtime and storage with respect to *N* [1]. Nevertheless, with BIGE we can observe how much any heuristic is far from optimality up to 32 bits.

4.5.2 Runtime and Memory Storage

For N≥128, only Lefèvre's CSP algorithm remains practical $O(N^3)$ for SCM, because even when neglecting the hidden constant α in $O(N^6)$, Hcub requires more than 4398 billions of iterations. The situation is even worse for optimal algorithms, such as MAG and BIGE, requiring exponential $O(2^N)$ runtime and memory storage.

For applications involving huge constants such as in cryptography, only linear heuristics $O(N)$ are practical (CSD and DBNS). It seems there is a strong correlation between performance and runtime. However, by extending the analysis of prior works and providing new insight, it is possible to improve both the runtime and the performance (in terms of minimizing logic resources) as will be shown later.

4.5.3 Overflow Risk

In fixed-point representation, an overflow may occur if shift spans are not precisely controlled, especially in the last partial product. While overflow is an important problem, as far as we are aware, it has never been addressed in the proposed heuristics [6]. Contrary to recoding-heuristics (CSD, DBNS, ...) where the shift span is fixed, the non-recoding heuristics are very prone to overflow risk because of their variable shift spans. Therefore, lower bounds on the maximum left-shifts must be carefully considered to minimize overflow; this is may be to the detriment of the optimization.

4.5.4 Ease of Use

Beyond the provided performances, ease of use is a decisive criterion in the selection of any heuristic. Not only it facilitates the implementation, but also makes the latter non-prone to programming errors. Some of the proposed heuristics, for instance Hcub or BIGE, rely on complex mathematical concepts. A deep understanding of the concept is a necessary condition prior any programming. Such a required effort would require too much time which might discourage the user.

This is not a surprise that despite the large number of published heuristics; CSD is not only used in designing the vast majority of LTI systems [2], but incorporated in most of the commercial tools as well, such as in Synopsys and Cadence synthesis tools, and Matlab numerical-computing tool.

Our proposed SCM/MCM heuristics attempt to address the weaknesses of the existing heuristics. We propose new *linear* runtime and fully *predictable* heuristics with high *compression* capabilities. The proposed heuristics are based on radix-2^r arithmetic. They are easy to use and overflow-safe.

4.6 New Recoding Algorithm (RADIX-2^r)

Radix-2^r has been concisely introduced in Section 3.3.4. We reconsider hereafter the most essential elements and provide further details and explanations. An N-bit constant C is expressed in radix-2^r as follows:

$$C = \sum_{j=0}^{(n/r)-1} \left(c_{rj-1} + 2^0 c_{rj} + 2^1 c_{rj+1} + 2^2 c_{rj+2} + \cdots + 2^{r-2} c_{rj+r-2} - 2^{r-1} c_{rj+r-1} \right) \times 2^{rj}$$

$$= \sum_{j=0}^{(n/r)-1} Q_j \times 2^{rj} \ , \qquad (4.22)$$

where $c_{-1} = 0$ and $r \in \mathrm{N}^*$. For simplicity purposes and without loss of generality, we assume that r is a divider of N. In Eq. 4.22, the two's complement representation of the constant C is split into N/r two's complement slices (Q_j), each of r bit length because it goes from 2^0 to 2^{r-1}. However, Q_j needs an additional bit (c_{rj-1}) equal to the most significant bit of the previous digit (Q_{j-1}), which could

be seen as some form of carry due to the use of signed digits; it comes from the following formula:

$$-2^{r-1}c_{rj+r-1}\times 2^{rj}+c_{rj+r-1}\times 2^{r(j+1)}=c_{rj+r-1}\times 2^{rj+r-1}. \quad (4.23)$$

This formula expresses the transformation of the conventional radix-2^r representation to the signed-digit radix-2^r one. A digit-set $DS(2^r)$ corresponds to Eq. 4.22, such as

$$Q_j \in DS(2^r)=\{-2^{r-1},-2^{r-1}+1,\ldots,-1,0,1,\ldots,2^{r-1}-1,2^{r-1}\}. \quad (4.24)$$

Thus, the product becomes: $$C\times X=\sum_{j=0}^{(N/r)-1} X\times Q_j\times 2^{rj}\ . \quad (4.25)$$

The sign of the Q_j term is given by the c_{rj+r-1} bit, and $|Q_j|=2^{k_j}\times m_j$, with $k_j\in\{0,1,2,\ldots,r-1\}$ and $m_j\in OM(2^r)\cup\{0\}$, where $OM(2^r)=\{1,3,5,\ldots,2^{r-1}-1\}$. $OM(2^r)$ is the set of odd positive digits in radix-2^r recoding, with $|OM(2^r)|=2^{r-2}$. To $|Q_j|=0$ corresponds m_j=0. Finally, the product can be expressed as follows: $$C\times X=\sum_{j=0}^{(N/r)-1}(-1)^{c_{rj+r-1}}\times(m_j\times X)\times 2^{rj+k_j}\ . \quad (4.26)$$

Unlike the multiplication by a variable ($Y\times X$) where the entire set of partial-products ($m_j\times X$) must be precomputed, only a subset is needed in the case of the multiplication by a constant ($C\times X$). In fact, the number of partial-products is equal to the number of different values m_j induced by the encoding process of the N/r slices (terms Q_j). Therefore, the generation of partial products (PP) consists first, if $m_j\neq 0$, in computing the PP $m_j\times X$ *if it has not been precomputed before*. It is then submitted to a hardwired left-shift of $rj+k_j$ positions, and finally, conditionally negated $(-1)^{c_{rj+r-1}}$ depending on the sign bit c_{rj+r-1} of Q_j. An illustrative example is given farther.

4.6.1 Maximum Number of Additions (*Upb*)

On the one hand, there are *N*/*r* iterations in Eq. 4.26. Each iteration generates one PP. Thus, the maximal number of PP is N/r, which requires a maximum of $N_{pp}=N/r-1$ additions. On the other hand, a maximum of $2^{r-2}-1$ non-trivial PP $\{3\times X,5\times X,7\times X,\cdots,(2^{r-1}-1)\times X\}$ can be invoked during the PP generation process. They are built using the binary method, from the least significant bit to the most significant bit. That is, the m_j elements 3, 5, 7, ..., $2^{r-1}-1$ are built one after the other, each time by using a single addition between an element that has already been built and a power of two. This process is summarized by the following recurrence relation: $m_j=2^p+d$, where $p\leq r-2$ because $m_j\leq 2^{r-1}-1$, and $0<d<2^p$.

Theorem 4.2 – *In radix-2^r, the precomputation of the entire set of non-trivial PP {3×X, 5×X, 7×X,…,(2^{r-1}–1)×X} yields an adder-cost and an adder-depth of $2^{r-2}-1$ and r–2, respectively.*

Proof – Since each new non-trivial digit requires only one addition (recurrence relation), the adder-cost

is the number of non-trivial digits: $N_{om} = \left|OM\left(2^r\right)\right| - 1 = 2^{r-2} - 1$. As the binary method is used, the adder-depth is deduced from the maximum number of non-zero bits in the binary representation of a digit: (r–1)–1=r–2. Since there are N/r PP, the maximum adder-depth (*Ath*) in cascaded adders is:

$$Ath(r) = \left\lceil \frac{N}{r} - 1 + r - 2 \right\rceil = \left\lceil \frac{N}{r} + r - 3 \right\rceil, \tag{4.27}$$

where $\lceil\ \rceil$ is the ceiling function (e.g. $\lceil 5.29 \rceil = 6$).

We illustrate the construction process of non-trivial PP with the following radix-2^6 example:

$$\begin{aligned} OM\left(2^6\right) &= \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31\} \\ &= \{1\} \cup \{2^1 + 1 = 3\} \cup \{2^2 + 1 = 5, 2^2 + 3 = 7\} \cup \{2^3 + 1 = 9,\ 2^3 + 3 = 11,\ 2^3 + 5 = 13, 2^3 + 7 = 15\} \cup \\ &\quad \{2^4 + 1 = 17, 2^4 + 3 = 19, 2^4 + 5 = 21,\ 2^4 + 7 = 23, 2^4 + 9 = 25, 2^4 + 11 = 27, 2^4 + 13 = 29, 2^4 + 15 = 31\}. \end{aligned}$$

Thus, the PP ($m_j \times X$) corresponding to $OM\left(2^6\right)$ are subsequently calculated in the following order (6–2=4 steps):

$$\{3\times X\}\ ;\ \{5\times X, 7\times X\}\ ;\ \{9\times X, 11\times X, 13\times X, 15\times X\}\ ;$$
$$\{17\times X, 19\times X, 21\times X, 23\times X, 25\times X, 27\times X, 29\times X, 31\times X\}.$$

Fig.4.6 provides all necessary details for hardware implementation. It now becomes clear that Eq. 4.26 involves only *additions*, *subtractions*, and *left-shifts*. Note that right-shifts are not allowed since r, j, and k_j, are positive integers.

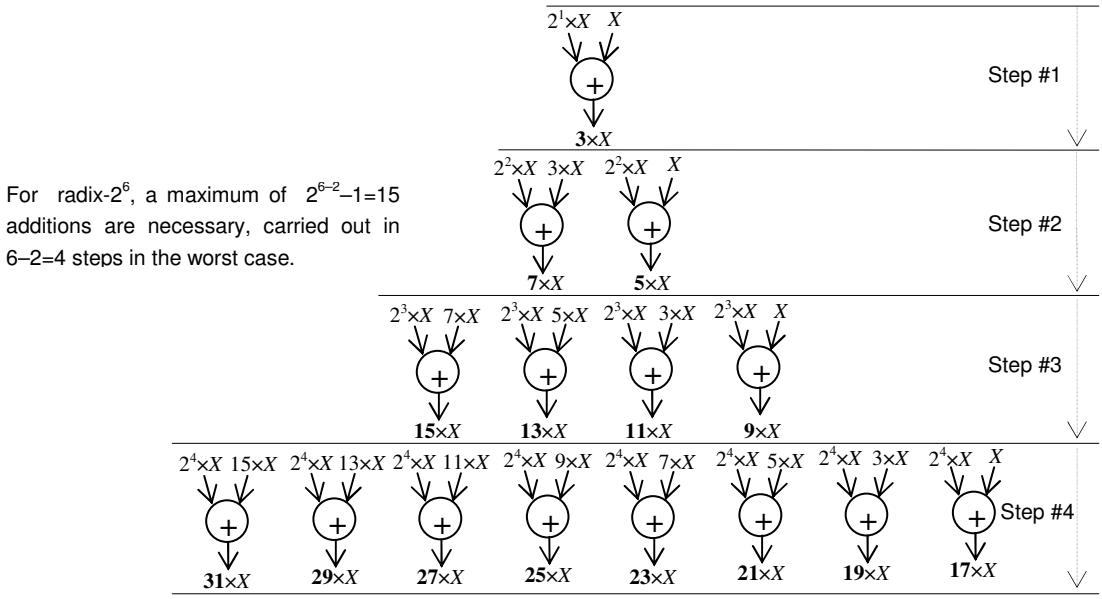


FIGURE 4.5 – Sequential order of computation of the entire set of partial-products needed by radix-2^6.

Consequently, the total number of additions required by radix-2^r is equal to:

$$U_{pb}(r) = N_{pp} + N_{om} = \left\lceil \frac{N}{r} + 2^{r-2} - 2 \right\rceil. \tag{4.28}$$

$U_{pb}(r)$ is minimal for $r = 2 \cdot W\left(\sqrt{N \cdot log(2)}\right) / log(2)$, where W is the Lambert Function. Table 4.3 gives the values of r that lead to the minimum number of additions for N ranging from 8 to 8192, while Fig. 4.6 depicts the upper-bounds in number of additions for CSD, DBNS, and RADIX-2^r.

TABLE 4.3 – Upper-bound (*Upb*) and r values for an N-bit constant using RADIX-2^r.

N	8	16	32	64	128	256	512	1024	2048	4096	8192
r	3	4	4	4	5	6	6	7	8	8	9
Upb(r)	3	6	10	18	32	57	100	177	318	574	1037

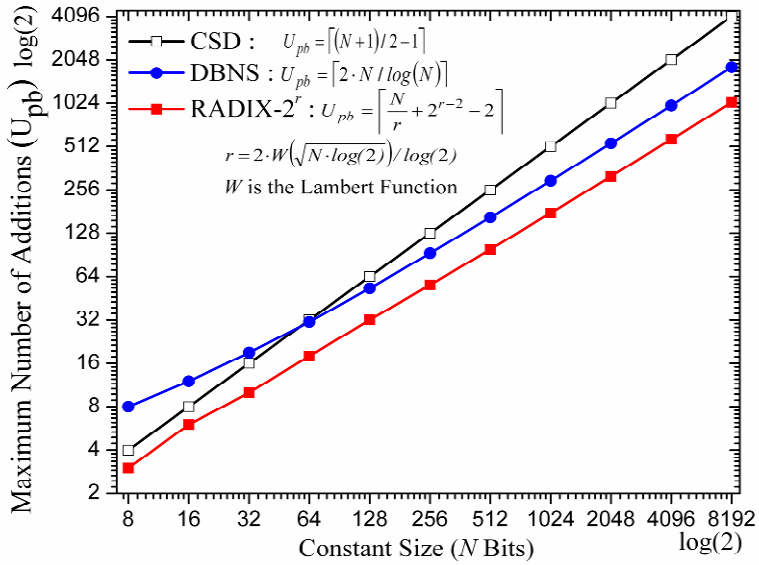


FIGURE 4.6 – *Upb* comparison for an N-bit constant.

As for the average number of additions (*Avg*), it has been *exhaustively* calculated for values of C varying from 0 to 2^N–1, for N=8, 16, 24, and 32. But for N=64, we have calculated *Avg* using 10^5, 10^6, 10^9 and 10^{10} *uniformly distributed random values* of C. While the difference between the four obtained results is insignificant (<10^{-3}), the value *Avg* oscillates around 15.5037 additions. Results are reported in Table 4.4. For N=64, RADIX-2^r uses 24.17 % less additions than CSD. This gain seems to grow linearly for low values of N.

TABLE 4.4 – RADIX-2^r versus CSD: average number of additions (*Avg*) and upper-bound (*Upb*).

Constant Bit-width N	CSD		RADIX-2^r		Saving (*Avg*,%)
	Avg	*Upb*	*Avg*	*Upb*	
8	1.7882	4	1.7843	3	0.2180
16	4.4445	8	4.2518	6	4.3356
24	7.1111	12	6.5314	8	8.1520
32	9.7777	16	8.6855	10	11.1703
64	20.4444	32	15.5037*	18	24.1666

*: Obtained from 10^{10} uniformly distributed random values of C.

CSD *Avg* = (N/3)–8/9 and CSD *Upb* = (N+1)/2–1.

Regarding DBNS, Dimitrov [6] calculated *Avg* and *Upb* from 10^5 uniformly distributed *random* constants, for 32 and 64 bits only (Table 4.5). Note that DBNS *Upb* will be higher if the worst cases are not attained by the pattern of 10^5 constants.

TABLE 4.5 – RADIX-2^r versus DBNS: average number of additions (*Avg*) and upper-bound (*Upb*).

Constant Bit-width N	DBNS [6]		RADIX-2^r		Saving (*Avg*,%)
	Avg	*Upb*	*Avg*	*Upb*	
32	≈9.05+*	13*	8.6855	10	4.0276
64	16.2151*	21*	15.5037	18	4.3872

+: Taken from Fig.1 in [6]; *: Obtained from 10^5 uniformly distributed random values of C.

We have also compared RADIX-2^r to some non-recoding heuristics (CSE and DAG) based on programs and numeric data kindly provided by Lefèvre and Voronenko. While Fig. 4.7 shows lower values of *Avg* for non-recoding heuristics as expected due to a larger exploration of the solution space, Table 4.6 exhibits rather a *higher* value of *Upb* for Bernstein's heuristic. *Significant conclusion*: a lower *Avg* does not guarantee a lower *Upb*. Another performance indicator of the recoding is the smallest value that requires q additions, for q varying from 1 to the upper-bound of the recoding. Table 4.7 summarizes this information for a 32-bit constant. One can note that starting from q=7, higher values are given by RADIX-2^r compared to CSD.

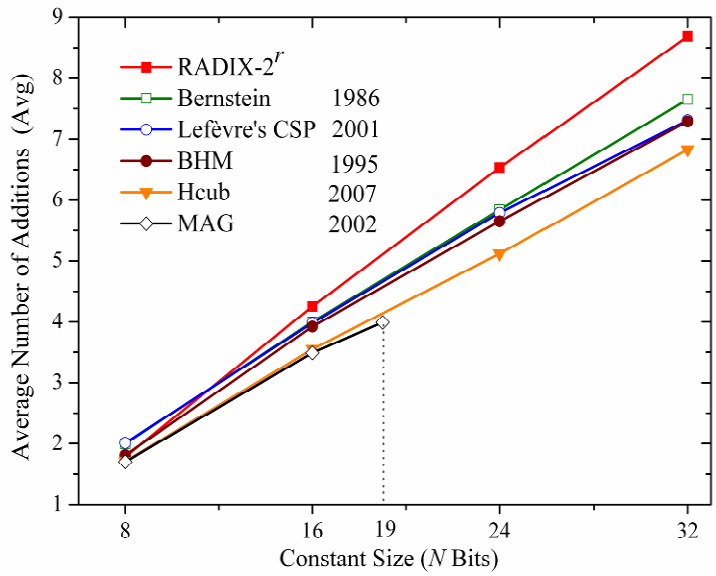


FIGURE 4.7 – *Avg* comparison for an N-bit constant.

Illustrative Example

The product $10599 \times X$ is first calculated in CSD, DBNS, and RADIX-2^r. Let us note that $(10599)_{10}=(10100101100111)_2$.

$P_{CSD}=(X\times2^{13})+(X\times2^{11})+(X\times2^{9})-(X\times2^{7})-(X\times2^{5})+(X\times2^{3})-X$.

$P_{DBNS}=((X_1\times2^{1})+X_1)+(X\times2^{13})+(X\times2^{3})-X$ with $X_1=((X_0\times2^{1})+X_0)+(X\times2^{5})$ and $X_0=(X\times2^{8})$ [6].

TABLE 4.6 – RADIX-2^r versus non-recoding algorithms: runtime complexity and number of additions of some special cases.

Algorithm	$(84AB5)_H$ N=20	$(64AB55)_H$ N=23$^+$	$(5959595B)_H$ N=31$^+$	Runtime [13]
BIGE [1]	4	5	6	$O(2^N)$
Bernstein [10]	8 ^{G}	7	8	$O(2^N)$ [8]
Hcub* [13]	4	6	–	$O(N^6)$
BHM* [23]	5	7	–	$O(N^4)$
Lefèvre's CSP [8]	4	6	9	$O(N^3)$
RADIX-2^r	5	7	10	$O(N)$

N: Constant bit-size; +: In RADIX-2^r, for $16 \le N \le 64$, r=4 (Table 4.3). A zero bit is added in the MSB position to make N a multiple of r (N=24, N=32). G: Greater than RADIX-2^r *Upb*; RADIX-2^r *Upb*=7, 8, and 10, for N=20, 24, and 32, respectively; *: Values are delivered by the Spiral web version (www.spiral.net), limited to 26 bits; x: optimal number of additions.

The BIGE optimal solutions for the indicated values are obtained as follows:
$(84AB5)_H$: 15 = $(2^4)-1$; 3825 = $(15\times 2^8)-15$; 19125 = $(3825\times 2^2)+3825$; 543413 = $(2^{19})+19125$.
$(64AB55)_H$: 255 = $(2^8)-1$; 65281 = $(255\times 2^8)+1$; 1109777 = $(65281\times 2^4)+65281$; 5548885 = $(1109777\times 2^2)+1109777$; 6597461 = $(2^{20})+5548885$.
$(5959595B)_H$: 257 = $(2^8)+1$; 16843009 = $(257\times 2^{16})+257$; 16843011 = (2)+16843009; 50529027 = $(16843009\times 2)+16843009$; 421075227 = $(50529027\times 2^3)+16843011$; 1499027803 = $(16843009\times 2^6)+421075227$.

In order to express the product in P_{RADIX}, a two's complement representation of $(10599)_{10}$ is necessary, which is $(010100101100111)_2$. Thus, in two's complement notation, the constant size is equal to N=15, to which corresponds r=4 (Table 4.3). As 15 is not a multiple of 4, the sign-bit (0 in this case) is extended by one position so as N=16. For $C=(10599)_{10}$, Eq. 4.22 and 4.26 become respectively:

$$C = \sum_{j=0}^{3} Q_j \times 2^{4j} \quad \text{, and} \quad P_{RADIX} = \sum_{j=0}^{3} (-1)^{c_{4j+3}} \times \left(m_j \times X\right) \times 2^{4j+k_j} .$$

Fig. 3.11 depicts the four terms Q_j. To determine the unknown values c_{4j+3}, m_j, and k_j, the radix-2^4 look-up table (Table 3.4) is indexed by the terms Q_j. Referring to Table 3.4, the triplets (c_{4j+3}, m_j, k_j) corresponding to Q_0, Q_1, Q_2, and Q_3 are (0,7,0), (0,3,1), (1,7,0), and (0,3,0), respectively. The recoding of C=10599 involves the precomputation of two PP only {3×X, 7×X}, while a maximum of three PP {3×X, 5×X, 7×X} can be invoke by radix-2^4 recoding. Consequently, we can write:

P_{RADIX} = $((3\times X)\times 2^{12})-((7\times X)\times 2^8)+((3\times X)\times 2^5)+(7\times X)$
= $(X_0\times 2^{12})-(X_1\times 2^8)+(X_0\times 2^5)+X_1$ with $X_0 = (X\times 2)+X$ and $X_1 = (X\times 2^2)+X_0$.

It has to be noted that for C=10599, P_{CSD} and P_{DBNS} require both 6 additions, while P_{RADIX} requires 5. The naive shift-and-add approach would have required 7 additions. We assume that addition and subtraction have the same area/speed cost, and that shift is costless since it can be realized without any gates, i.e. just by using hard wiring.

TABLE 4.7 – RADIX-2^r versus CSD, Lefevre's CSP, and exhaustive search: smallest values up to a 32-bit constant.

Number of Additions (q)	CSD	RADIX-2^r	Lefèvre's CSP* [8]	Exhaustive search [8]
1	3	3	3	3
2	11	11	11	11
3	43	43	43	43
4	171	139	213	683
5	683	651	1703	14709
6	2731	2699	13623	699829
7	10923	33419	174903	171398453+
8	43691	526491	1420471	–
9	174763	8422027	13479381	–
10	699051	134744219	–	–
11	2796203	–	–	–
12	11184811	–	–	–
13	44739243	–	–	–
14	178956971	–	–	–
15	715827883	–	–	–

*: Lefèvre calculated the values for q up to 9. This means that the common subpattern algorithm (CSP) exhibits an *Upb* ≥ 9 among all 32-bit constants.

+: This is the sole value which has not been confirmed by Lefèvre's exhaustive algorithm. It has been found only by Donelly [8], using left-shifts exclusively. If "right-shifts" are allowed, the value is strictly higher since the BIGE solution using right-shifts gives 6 additions, as follows: $5 = (2^2)+1$; $639 = (5\times2^7)-1$; $317 = (639-5)\times2^{-1}$; $5194045 = (317\times2^{14})+317$; $171393341 = (317\times2^{19})+5194045$; $171398453 = (639\times2^3)+171393341$.

Thong [1] conjectured that 7 additions are enough up to 32 bits, allowing right-shifts (exhaustive BIGE algorithm). It has been proved via RADIX-2^r heuristic that 10 additions are sufficient up to 32 bits, using left-shifts only.

Simplifications in Eq. 4.26 are possible in case two consecutive terms Q_j and Q_{j+1} with opposite signs exhibit pairs (m_j , k_j) of the form (1 , r–1) and (1 , 0), respectively. This is illustrated by the two following possibilities:

$$\cdots + X\times 2^{r(j+1)} - X\times 2^{rj+(r-1)} \pm \cdots = \cdots + X\times 2^{rj+(r-1)} \pm \cdots$$

$$\cdots - X\times 2^{r(j+1)} + X\times 2^{rj+(r-1)} \pm \cdots = \cdots - X\times 2^{rj+(r-1)} \pm \cdots$$

Another interesting idea is to include *redundancy* in the terms Q_j of Eq. 4.22 as will be shown farther. These two tricks will decrease the average number of additions in RADIX-2^r (Table 4.4, 4.5, and Fig. 4.7).

In addition to higher compression capabilities of RADIX-2^r compared to CSD and DBNS, its runtime complexity is linearly proportional to N as shown by Eq. 4.22. Moreover the required memory space is very small (for a 8192-bit constant corresponds a look-up table of 2^{9+1}=1024 entries). These two features make RADIX-2^r very useful for huge constants.

Since the introduction of H(k) [12] in 2004, CSE heuristics have outperformed DAGs at SCM [1]. This was achieved by applying CSE to each possible signed-digit (SD) form of the constant. Likewise, the search space of CSE can be expanded considering RADIX-2^r recoding instead of SD representation. For such a goal (SCM/MCM), Lefèvre's CSP heuristic [8] stands as the best CSE candidate for its lower computational complexity $O(N^3)$ in comparison to its CSE counterparts [13].

4.6.2 Average Number of Additions (*Avg*)

$OM(2^r)$ is the required set of odd-multiples $OM(2^r)=\{3,5,...,2^{r-1}-1\}$ in radix-2^r recoding, with $|OM(2^r)|=2^{r-2}-1$. Since each slice Q_j comprises r+1 bits, the total number of different combinations is then 2^{r+1}. According to Eq. 4.22, only two combinations produce Q_j =0: in case the r+1 bits are all "0" or all "1". Hence, the average number of non-null Q_j terms is equal to $(2^{r+1}-2)/2^{r+1}=1-2^{-r}$. Each Q_j≠0 generates one partial product (PP). Thus, the average number of PP in the $\lceil N/r \rceil$ slices is:

$$Avg_{pp}=(1-2^{-r})\times\lceil N/r \rceil. \tag{4.29}$$

For each $m_j \in OM(2^r)$ there exists an integer $k \in \{1, 2, ..., |OM(2^r)|\}$, such as $m_j=2\times k+1$. To set the correspondence between j and k, m_j is denoted m_{jk}. The exact number of occurrences (O_{cc}) of m_{jk} in the 2^{r+1} combinations of Q_j is :

$$O_{cc}(m_{jk})=\left\lfloor \frac{2^{r-1}}{2\times k+1} \right\rfloor \times 4, \tag{4.30}$$

where $\lfloor\ \rfloor$ is the floor function. We need to multiply by 4 because each occurrence of m_{jk} in the positive part of $DS(2^r)$ is double due to the fact that c_{rj-1} and c_{rj} bits have the same influence on Q_j term $(c_{rj-1}\times 2^0+c_{rj}\times 2^0+\cdots)$. Symmetrically, when considering the negative part of $DS(2^r)$, each occurrence of m_{jk} becomes then quadruple (see Table 3.4). Therefore, the probability (P) that m_{jk} occurs among 2^{r+1} combinations is:

$$P(m_{jk})=\frac{O_{cc}(m_{jk})}{2^{r+1}}. \tag{4.31}$$

We deliberately employ "probability" instead "average" to make easier the demonstration, but actually the two notions have *exactly the same* meaning in *this case*. Now, the probability that m_{jk} occurs in slice Q_j knowing that it has *never* occurred in the slices before slice j is (Bayes's theorem):

$$P(m_{jk}/j)=\frac{P(m_{jk}\cap j)}{P(j)}=\frac{P(m_{jk})\times[1-P(m_{jk})]^j}{1}=P(m_{jk})\times[1-P(m_{jk})]^j. \tag{4.32}$$

The probability that *any* m_{jk} for $k=1..|OM(2^r)|$ occurs in slice Q_j knowing that it has never occurred in the slices before slice j is:

$$P(\forall\, m_{jk}/j)=\sum_{k=1}^{|OM(2^r)|}P(m_{jk}/j). \tag{4.33}$$

Note that $P(m_{jk}/j)$ are mutually exclusive, since one and only one odd-multiple (m_{jk}) can occur in slice j. Consequently, the average number of generated odd-multiples considering all slices is:

$$Avg_{om} = \sum_{j=0}^{\lceil N/r \rceil - 1} P(\forall\, m_{jk} / j). \tag{4.34}$$

Hence, the average number of additions for RADIX-2^r is:

$$\begin{aligned} Avg(r) &= -1 + Avg_{pp} + Avg_{om} \\ &= -1 + (1 - 2^{-r}) \times \lceil N/r \rceil + \sum_{j=0}^{\lceil N/r \rceil - 1} \left\{ \sum_{k=1}^{2^{r-2}-1} P(m_{jk}) \times [1 - P(m_{jk})]\, j \right\}, \end{aligned} \tag{4.35}$$

with $P(m_{jk}) = \dfrac{\left\lfloor \dfrac{2^{r-1}}{2 \times k + 1} \right\rfloor}{2^{r-1}}$. We proved in the last section that to get the minimum number of additions, r must be equal to:

$$r = 2 \cdot W\left(\sqrt{N \cdot log(2)}\right) / log(2), \tag{4.36}$$

where W is the Lambert function.

Using *Avg* expression 4.35, we calculated the average for N varying from 8 to 8192. Results are reported in Table 4.8. Note that *insignificant* differences exist between the values exhaustively calculated in Tables 4.4 and those of Table 4.8. The reason is due to the fact that before calculating the number of additions, even values of C constant was reduced till to get an odd value. This is why *Avg* values in Tables 4.4 are slightly lower than those delivered by *Avg* expression. We rerun the same program but without the reduction of even number and got *exactly* the same values as *Avg* expression.

We observe that for RADIX-2^r, *Avg* values are not far from *Upb* ones as in the case of CSD. The reason is that the average number of null Q_j is very low: $Avg(Q_j = 0) = \dfrac{2}{2^{r+1}} \times \lceil N/r \rceil = \dfrac{\lceil N/r \rceil}{2^r}$. Note that 50% saving over CSD is attained for N=1148.

The maximum adder-depth (*Ath*) in cascaded adders is given by Eq. 4.27. Based on r values given by equation 4.36, we calculated *Ath*(r) and grouped the results in Table 4.8. A saving of 50% over CSD is achieved for N=80.

As for *Upb*, 50% saving is attained at N=128.

4.6.3 Length of the Critical-Path in Cascaded Adders (*Ath*)

Equation 4.36 assures minimum *Avg* and *Upb*, whereas lower *Ath* values are still possible. Any r value, such as $r \le 2 \cdot W\left(\sqrt{N \cdot log(2)}\right) / log(2)$ produces higher *Avg, Upb,* and *Ath.* While any r value, such as $r \ge 2 \cdot W\left(\sqrt{N \cdot log(2)}\right) / log(2)$ produces lower *Ath* but higher *Avg* and *Upb*. To garantee a reasonable balance, we set as condition that the entire number of odd-multiples must be equal, or less than the

total number of the slices Q_j $\left(\left|OM(2^r)\right| \le \lceil N/r \rceil\right)$. This condition avoids generating more odd-multiples than it is actually invoked. Thus, a balanced solution for a lower *Ath* is found for:

$$r = W(4.N.log(2))/log(2)\ , \qquad (4.37)$$

where W is the Lambert function.

TABLE 4.8 – RADIX-2^r versus CSD: *Avg*, *Ath*, and *Upb* for N-bit constant.

N		8	16	32	64	128	256	512	1024	2048	4096	8192
r		3	4	4	4	5	6	6	7	8	8	9
Avg	RADIX-2^r	1.78	4.26	8.71	16.75	30.40	54.08	98.12	174.22	312.49	571.41	1033.39
	CSD	1.78	4.44	9.77	20.44	41.77	84.44	169.77	340.44	681.77	1364.44	2729.77
	Saving (%)	0.39	4.12	10.89	18.05	27.22	35.95	42.20	48.82	54.16	58.12	62.14
Ath	RADIX-2^r	3	5	9	17	28	46	89	151	261	517	917
	CSD	4	8	16	32	64	128	256	512	1024	2048	4096
	Saving (%)	25	37.5	43.75	46.87	56.25	64.06	65.23	70.50	74.51	74.75	77.61
Upb	RADIX-2^r	3	6	10	18	32	57	100	177	318	574	1037
	CSD	4	8	16	32	64	128	256	512	1024	2048	4096
	Saving (%)	25	25	37.5	43.75	50	55.46	60.93	65.42	68.94	71.97	74.68

N is the constant bit-size; $r = 2 \cdot W\left(\sqrt{N \cdot log(2)}\right)/log(2)$, where W is the Lambert function.

Table 4.9 indicates r values that yield lower *Ath*, accompanied with its corresponding *Upb* and *Avg*. Note that both equations 4.36 and 4.37 provide exactly the same results for $N \le 32$, either in *Ath*, *Avg*, or *Upb*. Starting from $N \ge 64$, lower *Ath* are obtained using Eq. 4.37 but at the expense of higher *Upb* and *Avg* as indicated by Tables 4.8 and 4.9. For instance, for N=256 Eq. 4.37 achieves a reduction of 22.5% in *Ath* over equation 4.36, while it causes an increase of 26% and 2.5% in *Upb* and *Avg*, respectively. Contrary to *Avg* corresponding to equation 4.36, the ones of Eq. 4.37 are relatively far from *Upb*. Compared to CSD, a saving of 50% in *Ath* is obtained by Eq. 4.37 for N=56.

Finally, to decide which r expression to use depends actually on design requirements. If priority is given to area, Eq. 4.36 must be used. But in case speed is a concern, Eq. 4.36 must be employed.

TABLE 4.9 – Upper-bound and r values for N-bit constant using RADIX-2^r.

N	8	16	32	64	128	256	512	1024	2048	4096	8192
r	3	4	4	5	6	7	8	8	9	10	11
Ath(r)	3	5	9	15	25	41	69	133	234	417	753
Upb(r)	3	6	10	19	36	67	126	190	354	664	1255
Avg(r)	1.78	4.26	8.71	16.61	30.77	55.47	100.56	175.99	322.83	594.90	1104.27

N is the constant bit-size; $r = W\left(4.N.log(2)\right)/log(2)$, where W is the Lambert function.

4.6.4 Overflow Safety

In RADIX-2^r, overflow safety is easy to prove. We consider C and X with n and m bit-lengths, respectively. In two's complement representation, the product $P = C \times X$ needs $n+m$ bits to be

complete, i.e., without truncation. We can write: $P = p_{n+m-1}\ p_{n+m-2}\ \cdot\ \cdot\ \cdot\ p_1\ p_0$; where p_{n+m-1} is the sign-bit. To be sure there is no overflow risk; we must prove that the sign-bit of the last partial product (PP) is set *at most* at the $n+m-1$ position. We write:

$$P = \sum_{j=0}^{(n/r)-1} Q_j \times X \times 2^{rj} = \sum_{j=0}^{(n/r)-1} (-1)^{c_{rj+r-1}} \times |Q_j| \times (-1)^{x_{m-1}} \times |X| \times 2^{rj} = \sum_{j=0}^{(n/r)-1} PP_j ,$$

where the last PP is equal to: $PP_{n/r-1} = (-1)^{c_{n-1}} \times |Q_j| \times (-1)^{x_{m-1}} \times |X| \times 2^{n-r}$. The maximal positive values that $|Q_j|$ and $|X|$ can take are 2^{r-1} and 2^{m-1}, respectively, to which corresponds a maximal PP of:

$$\max(PP_{n/r-1}) = (-1)^{c_{n-1}+x_{m-1}} \times 2^{n+m-2}.$$

In this case, 2^{n+m-2} occupies the $n+m-2$ position, plus the sign bit just after at $n+m-1$ position. This proves that in RADIX-2^r overflow never occurs.

4.7 New Redundant Radix-2^r Recoding (R3) Algorithm

The objective is to decrease *Avg* without increasing *Upb*. *Avg* is successively reduced in two steps: by the utilization of a redundant recoding, followed by a Common Digit Elimination (CDE) step on the PP set. In RADIX-2^r, CDE is already applied on the odd-multiples (m_j) by the recoding itself. A second order of CDE can be applied again on Q_j terms thanks to redundancy.

We present hereafter a linear runtime Redundant Radix-2^r Recoding (R3) with better *Avg* and same *Upb* as RADIX-2^r.

Eq. 4.22 can be rewritten as

$$C = \sum_{j=0}^{(N/r)-1} (-1)^{c_{rj+r-1}} \times \left(m_j \times 2^{k_j}\right) \times 2^{rj}, \qquad (4.38)$$

with $m_j \in \{0,1,3,5,\ldots, 2^{r-1}-1\}$ and $k_j \in \{0,1,2,\ldots, r-1\}$.

To enable CDE at Q_j level, we announce the two following theorems.

Theorem 4.3 – *Any digit* $Q_j \in DS(2^r)$ *can be represented in a combination of digits* $P_{ji} \in DS(2^s)$, *such as* s *is a divider of r.*

Theorem 4.4 – *Any digit* $Q_j \in DS(2^r)$ *can be represented in a combination of digits* $P_{ji}+T_{jk}$ *such as* $P_{ji} \in DS(2^s)$ *and* $T_{jk} \in DS(2^t)$ *with* $s+t$ *a divider of* r, *and* $t < s$.

Proofs – see Appendix A.

When Th. 4.3 is applied to Eq. 4.22, it gives:

$$C = \sum_{j=0}^{(N/r)-1} \left[\sum_{i=0}^{(r/s)-1} P_{ji}\ 2^{si} \right] 2^{rj}, \qquad (4.39)$$

where $P_{ji} \in DS(2^s) = \{-2^{s-1}, -2^{s-1}+1, \ldots, 0, \ldots, 2^{s-1}-1, 2^{s-1}\}$, $OM(2^s) = \{1, 3, \ldots, 2^{s-1}-1\}$ such as $|OM(2^r)|/|OM(2^s)| = 2^{(k-1)s}$ with $r/s=k$. Th. 4.3 allows an exponential reduction ($1/2^{k-1}$) of the number of odd-multiples in Eq. 4.39 in comparison to Eq. 4.22, but at the expense of a linear increase (k–1) in the number of additions.

Corollary 4.1 – In radix-2^r, $|Q_j| = u_j \times 2^{l_j} + (-1)^{e_j} \times v_j \times 2^{h_j}$, where: u_j, $v_j \in \{0, 1, 3, 5, \ldots, 2^{(r/2)-1}-1\}$; $l_j \in \{0, 1, 2, \ldots, 2^{r-1}-1\}$; $h_j \in \{0, 1, 2, \ldots, 2^{(r/2)-1}-1\}$; and $e_j \in \{0, 1\}$.

Proof – This corollary is a direct consequence of Th. 4.3 applied for $s=r/2$. This means that Q_j digit, which is r+1 bit-length, is split into two overlapping sub-digits P_{j0} and P_{j1}, each of $r/2$+1 bit-length. This assumes that r is even. If r is odd, Th. 4.4 is applied instead of Th. 4.3. For $s=r/2$, Eq. (4.39) becomes:

$$C = \sum_{j=0}^{N/r-1} \left(P_{j0} + P_{j1} \times 2^{r/2}\right) \times 2^{rj}. \tag{4.40}$$

Note that $Q_j = P_{j0} + P_{j1} \times 2^{r/2}$, and that P_{j0} and P_{j1} have exactly the same properties as Q_j, which means that they can be expressed in the same way as Q_j is written in equation (4.38). Thus, we get:

$$C = \sum_{j=0}^{N/r-1} (-1)^{c_{rj+r-1}} \times \left[u_j \times 2^{l_j} + (-1)^{e_j} \times v_j \times 2^{h_j}\right] \times 2^{rj}. \tag{4.41}$$

Because addition is a *non-injective* function, the quintuplet (u_j, l_j, e_j, v_j, h_j) is not unique; several ones might exist for the same $|Q_j|$ value. For instance, the term $|Q_j| = 35$ can be expressed as: $35 = 1 \times 2^5 + 3 \times 2^0$, or $35 = 5 \times 2^3 - 5 \times 2^0$, or $35 = 7 \times 2^2 + 7 \times 2^0$. Consequently, Eq. (4.41) is a redundant radix-2^r recoding (R3) of the constant C.

Corollary 4.1 is just one case ($s=r/2$) among many others. A number of Q_j partitionings are possible ($s=r/3$, $r/4$,...), but lower values of s increase the number of sub-digits, which makes equation (4.41) difficult to handle.

R3 is illustrated hereafter for the particular case of $8 < N \leq 64$. To preserve optimality in *Upb* and *Avg*, r must be equal to 4 (Table 4.8). But as R3 comprises two sub-digits, r must be doubled (r=8), which means that s=4. Hence, with (r,s)=(8,4) optimality is guaranteed.

For r=8, $0 \leq |Qj| \leq 128$, and equation (4.41) becomes:

$$\begin{aligned} C &= \sum_{j=0}^{(N/8)-1} \left(u_j \times 2^{l_j} + (-1)^{e_j} \times v_j \times 2^{h_j}\right) \times (-1)^{c_{8j+7}} \times 2^{8j} \\ &= \sum_{j=0}^{(N/8)-1} (Z_1 + Z_2)_j \times (-1)^{c_{8j+7}} \times 2^{8j}, \end{aligned} \tag{4.42}$$

where $Z_1 = u_j \times 2^{l_j}$; $Z_2 = (-1)^{e_j} \times v_j \times 2^{h_j}$; $u_j, v_j \in \{0, 1, 3, 5, 7\}$; $p_j \in \{0, 1, 2, \ldots, 7\}$; $h_j \in \{0, 1, 2, 3\}$; and $e_j \in \{0, 1\}$.

Note that $|Q_j|=(Z_1+Z_2)_j$. Thus, the product $C\times X$ becomes:

$$C\times X=\sum_{j=0}^{(N/8)-1}\left[\left(u_j\times X\right)\times 2^{p_j}+(-1)^{e_j}\times\left(v_j\times X\right)\times 2^{h_j}\right]\times(-1)^{c_{8j+7}}\times 2^{8j}\ . \qquad (4.43)$$

The partitioning of the constant C according to equation (4.42) is depicted in Fig. 4.8.

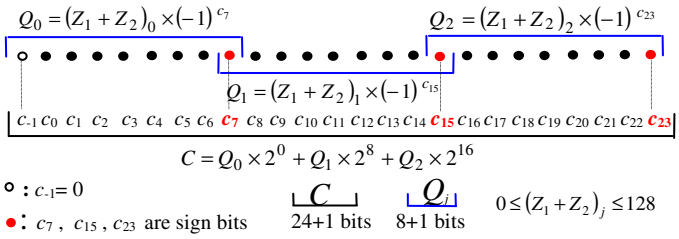


FIGURE 4.8 – Partitioning of a 24-bit C constant using R3 algorithm.

Since $|Q_j|$ may have several notations in (Z_1, Z_2), we must carefully select among a big number of possibilities, the recoding (R3) that yields optimal *Avg*. We have shown that for RADIX-2^r, $Avg(Q_j=0)=\lceil N/r\rceil/2^r$ and we can easily prove applying the same reasoning developed in Section 4.6.2 that $Avg(Q_j=1)=(2\times r-1)\times\lceil N/r\rceil/2^r$. Thus, we can write: $Avg(Q_j=0,1)=r\times\lceil N/r\rceil/2^{r-1}$. Preserving the same $Avg(Q_j=0,1)$ value in R3 is a proof of optimality of *Avg*, because the number of PPs as well as the odd-multiple set are exactly the same in R3 and RADIX-2^r.

Optimal R3 recoding is obtained using a C-program which exhaustively explores for each odd $|Q_j|$ varying from 1 to 127, all (u_j, l_j, e_j, v_j, h_j) possibilities and selects the least adder consumer combination according to the following priority order: (u_j , v_j)=(u_j , 0); (u_j , v_j)=(1 , 1); (Z_1,Z_2)=(1×2^7, Z_2); and finally (Z_1 , Z_2)=(Z_1 ,1×2^0). These two latter couples allow the following simplifications:

$$\ldots+\left[(1\times 2^7+Z_2)\times 2^{8j}\right]+\left[(Z_1-1\times 2^0)\times 2^{8j+8}\right]\pm\ldots=\ldots+\left[Z_2\times 2^{8j}-2\right]+\left[Z_1\times 2^{8j+8}\right]\pm\ldots$$

$$\ldots-\left[(1\times 2^7+Z_2)\times 2^{8j}\right]+\left[(Z_1+1\times 2^0)\times 2^{8j+8}\right]\pm\ldots=\ldots-\left[Z_2\times 2^{8j}-2\right]+\left[Z_1\times 2^{8j+8}\right]\pm\ldots$$

In case none of those cases is encountered, C-program pursues in the following priority order: (u_j , v_j)=(1,3) or (3,1); (u_j , v_j)=(3 , 3); (u_j , v_j)= (1,5) or (5,1); (u_j , v_j)=(5, 5); (u_j , v_j)= (1, 7) or (7, 1); (u_j , v_j)=(7, 7); (u_j , v_j)= (3,5) or (5,3); (u_j , v_j)= (3,7) or (7,3); (u_j , v_j)= (5,7) or (7,5). This order maximizes the occurrences of 1, then of 3, and minimizes those of 5 and 7 in $|Q_j|$ digits, which will more likely reduce the number of adders in the whole C recoding. Optimized odd $|Qj|$ combinations are grouped in Table 4.10. Even combinations of $|Qj|$ are directly derived from odd ones using shift operations.

For a given 8<N≤64, optimality for RADIX-2^r and R3 is guaranteed for r=4 and r=8, respectively. To RADIX-2^r corresponds $Avg(Q_j=0,1)=\lceil N/4\rceil/2$. Counting the number of u_j=1, v_j=0, and v_j=1 in both odd and even $|Q_j|$ of Table 4.10, we can easily prove that for R3, $Avg(v_j=0)=24\times\lceil N/8\rceil/128$ and $Avg(u_j=1)+Avg(v_j=1)=52\times\lceil N/8\rceil/64$. This gives $Avg(u_j=1)+Avg(v_j=0,1)=\lceil N/8\rceil$ which is equal to $Avg(Q_j=0,1)$. This is a formal proof that R3 (Table 4.10) is optimal.

TABLE 4.10 – Odd and even $|Q_j|$ digit recoding using R3 algorithm.

Odd $\|Q_j\|$	$Z_1=u_j\times 2^{l_j}$	$Z_2=(-1)^{e_j}\times v_j\times 2^{h_j}$	$(Z_1+Z_2)_j$	Even $\|Q_j\|$	$(Z_1+Z_2)_j$
1	1×2^0	0×2^0	U_1	2	$2^1\times U_1$
3	3×2^0	0×2^0	U_3	4	$2^2\times U_1$
5	5×2^0	0×2^0	U_5	6	$2^1\times U_3$
7	7×2^0	0×2^0	U_7	8	$2^3\times U_1$
9	1×2^3	1×2^0	U_9	10	$2^1\times U_5$
11	3×2^2	-1×2^0	U_{11}	12	$2^2\times U_3$
13	3×2^2	1×2^0	U_{13}	14	$2^1\times U_7$
15	1×2^4	-1×2^0	U_{15}	16	$2^4\times U_1$
17	1×2^4	1×2^0	U_{17}	18	$2^1\times U_9$
19	5×2^2	-1×2^0	U_{19}	20	$2^2\times U_5$
21	5×2^2	1×2^0	U_{21}	22	$2^1\times U_{11}$
23	3×2^3	-1×2^0	U_{23}	24	$2^3\times U_3$
25	3×2^3	1×2^0	U_{25}	26	$2^1\times U_{13}$
27	7×2^2	-1×2^0	U_{27}	28	$2^2\times U_7$
29	7×2^2	1×2^0	U_{29}	30	$2^1\times U_{15}$
31	1×2^5	-1×2^0	U_{31}	32	$2^5\times U_1$
33	1×2^5	1×2^0	U_{33}	34	$2^1\times U_{17}$
35	1×2^5	3×2^0	U_{35}	36	$2^2\times U_9$
37	1×2^5	5×2^0	U_{37}	38	$2^1\times U_{19}$
39	5×2^3	-1×2^0	U_{39}	40	$2^3\times U_5$
41	5×2^3	1×2^0	U_{41}	42	$2^1\times U_{21}$
43	5×2^3	3×2^0	U_{43}	44	$2^2\times U_{11}$
45	3×2^4	-3×2^0	U_{45}	46	$2^1\times U_{23}$
47	3×2^4	-1×2^0	U_{47}	48	$2^4\times U_3$
49	3×2^4	1×2^0	U_{49}	50	$2^1\times U_{25}$
51	3×2^4	3×2^0	U_{51}	52	$2^2\times U_{13}$
53	3×2^4	5×2^0	U_{53}	54	$2^1\times U_{27}$
55	7×2^3	-1×2^0	U_{55}	56	$2^3\times U_7$
57	7×2^3	1×2^0	U_{57}	58	$2^1\times U_{29}$
59	1×2^6	-5×2^0	U_{59}	60	$2^4\times U_{15}$
61	1×2^6	-3×2^0	U_{61}	62	$2^1\times U_{31}$
63	1×2^6	-1×2^0	U_{63}	64	$2^6\times U_1$
65	1×2^6	1×2^0	U_{65}	66	$2^1\times U_{33}$
67	1×2^6	3×2^0	U_{67}	68	$2^2\times U_{17}$
69	1×2^6	5×2^0	U_{69}	70	$2^1\times U_{35}$
71	1×2^6	7×2^0	U_{71}	72	$2^3\times U_9$
73	5×2^4	-7×2^0	U_{73}	74	$2^1\times U_{37}$
75	5×2^4	-5×2^0	U_{75}	76	$2^4\times U_{19}$
77	5×2^4	-3×2^0	U_{77}	78	$2^1\times U_{39}$
79	5×2^4	-1×2^0	U_{79}	80	$2^4\times U_5$
81	5×2^4	1×2^0	U_{81}	82	$2^1\times U_{41}$
83	5×2^4	3×2^0	U_{83}	84	$2^2\times U_{21}$
85	5×2^4	5×2^0	U_{85}	86	$2^1\times U_{43}$
87	5×2^4	7×2^0	U_{87}	88	$2^3\times U_{11}$
89	3×2^5	-7×2^0	U_{89}	90	$2^1\times U_{45}$
91	3×2^5	-5×2^0	U_{91}	92	$2^2\times U_{23}$
93	3×2^5	-3×2^0	U_{93}	94	$2^1\times U_{47}$
95	3×2^5	-1×2^0	U_{95}	96	$2^5\times U_3$
97	3×2^5	1×2^0	U_{97}	98	$2^1\times U_{49}$
99	3×2^5	3×2^0	U_{99}	100	$2^2\times U_{25}$
101	3×2^5	5×2^0	U_{101}	102	$2^1\times U_{51}$
103	3×2^5	7×2^0	U_{103}	104	$2^3\times U_{13}$
105	7×2^4	-7×2^0	U_{105}	106	$2^1\times U_{53}$
107	7×2^4	-5×2^0	U_{107}	108	$2^2\times U_{27}$
109	7×2^4	-3×2^0	U_{109}	110	$2^1\times U_{55}$
111	7×2^4	-1×2^0	U_{111}	112	$2^4\times U_7$
113	7×2^4	1×2^0	U_{113}	114	$2^1\times U_{57}$
115	7×2^4	3×2^0	U_{115}	116	$2^3\times U_{29}$
117	7×2^4	5×2^0	U_{117}	118	$2^1\times U_{59}$
119	7×2^4	7×2^0	U_{119}	120	$2^3\times U_{15}$
121	1×2^7	-7×2^0	U_{121}	122	$2^1\times U_{61}$
123	1×2^7	-5×2^0	U_{123}	124	$2^2\times U_{31}$
125	1×2^7	-3×2^0	U_{125}	126	$2^1\times U_{63}$
127	1×2^7	-1×2^0	U_{127}	128	$2^7\times U_1$

Note that $9=1\times 2^3+1\times 2^0$ in R3 (1 addition) and $9=1\times 2^4-7\times 2^0$ in RADIX-2^r (2 additions), taking into account that the recoding is on 8+1=9 bits (Fig. 4.8). There are many cases where the number of additions is lower, as in 10, 40, …

As for *Upb*, R3 comprises $\lceil N/8 \rceil$ Q_j, each one groups two digits (Z_1, Z_2). Thus, the number of PP is $\lceil N/4 \rceil$. Since 3 odd-multiples are required, $Upb=\lceil N/4 \rceil+2$ which is equal to *Upb* of RADIX-2^r. It is important to mention that $8<N\leq64$ was chosen just to make the demonstration simpler, but the proof holds true for any value of N.

CDE is performed in a linear runtime on the $\lceil N/2\times r \rceil$ U_k digits as an ultimate optimization step. It is illustrated by the product $P=(2631689)_{10}\times X$. We first calculate the product (P) in RADIX-2^r and then in R3.

$P_{\text{RADIX}} = X_0\times2^{20}-X\times2^{19}+X_0\times2^{12}-X\times2^{11}+X\times2^4-X_1$ with $X_0=(X\times2)+X$ and $X_1=(X\times2^3)-X$.

$P_{\text{R3}}=U_{40}\times2^{16}+U_{40}\times2^8+U_9$ with $U_{40}= U_5\times2^3$; $U_5=(X\times2^2)+X$ and $U_9=(X\times2^3)+X$.

Note that P_{RADIX} requires 7 additions, while P_{R3} needs only 4. A saving of 2 additions is due to redundancy (U_9 and U_{40}) and a saving of 1 addition is due to CDE (U_{40}).

Avg has been *exhaustively* calculated for C values varying from 0 to 2^N-1, for N=8, 16, 24, and 32. But for N=64, we calculated *Avg* using 10^{10} uniformly distributed random C values. For N=64, R3 uses 13.49 % less additions than RADIX-2^r (Table 4.11). For $N\leq32$, the saving is not substantial because the number of Q_j digits is very low (≤4). But for N=64, it is equal to 8, offering more possibilities to CDE. Note that for N=8, the saving of 3.51% is due *exclusively* to the use of redundancy since there is only one Q_j digit.

TABLE 4.11 – R3 versus RADIX-2^r: average number of additions (*Avg*).

N	*Avg*		Saving %
	RADIX-2^r	R3	
8	1.78	1.72	3.51
16	4.26	4.10	3.66
24	6.54	6.28	4.04
32	8.71	8.31	4.51
64	16.75	14.49*	13.49

*:Obtained from 10^{10} uniformly distributed random C values. N is the bit-size of the constant. For N=8, the saving is exclusively due to redundancy (Table 4.9).

TABLE 4.12 – R3 versus RADIX-2^r: smallest values up to 32-bit constant.

q	RADIX-2^r	R3
1	3	3
2	11	11
3	43	43
4	139	139
5	651	651
6	2699	2699
7	33419	34971
8	526491	559259
9	8422027	17336475
10	134744219	143163547

q: number of additions.

We also determined the smallest value that requires q additions, for q varying from 1 to the *Upb* of the recoding. Table 4.12 summarizes the results for 32-bit constant. Note that starting from q=7, higher values are given by R3.

We have compared R3 to a number of well-known non-recoding algorithms for which neither *Avg* nor *Upb* are analytically known. While they exhibit lower *Avg* (Fig. 4.9), their respective *Upb* may be higher such as in the case of Bernstein's algorithm (Table 4.13).

In Fig. 4.9 the comparison is limited to 32 bits. Beyond that bit-length, and especially for large constants (N≥64), it would be impossible to perform any comparison since the analytical expression of the average is unknown for all heuristics, except RADIX-2^r. Therefore, there is no guarantee that Hcub will preserve its leading position for high values of N. As for R3, CDE becomes more effective since the number of Q_j digits increases. This may enable R3 to rapidly outperform Bernstein's heuristic. Finally, for huge constants, only R3 remains practical due to its linear computational time $O(N)$.

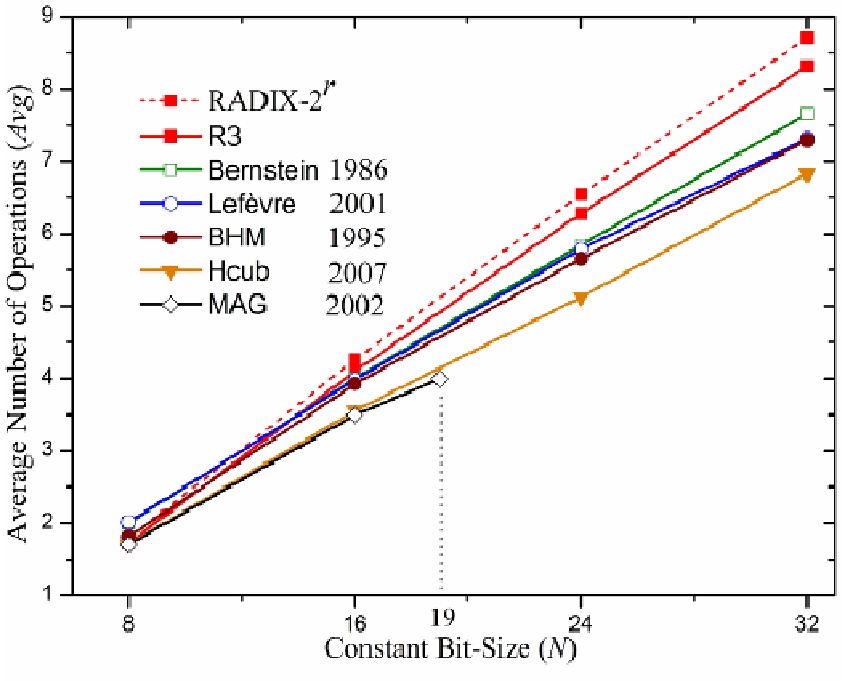


FIGURE 4.9 – *Avg* comparison for a N-bit constant.

TABLE 4.13 – R3 and RADIX-2^r versus non-recoding algorithms: runtime complexity and number of additions of some special cases.

Algorithm	$(84AB5)_H$ N=20	$(64AB55)_H$ N=23+	$(5959595B)_H$ N=31+	Runtime [13]
BIGE [1]	4	5	6	$O(2^N)$
Bernstein [10]	8^G	7	8	$O(2^N)$ [8]
Hcub* [13]	4	6	–	$O(N^6)$
BHM* [23]	5	7	–	$O(N^4)$
Lefèvre'CSP [8]	4	6	9	$O(N^3)$
RADIX-2^r	5	7	10	$O(N)$
R3	4	6	8	$O(N)$

N: Constant bit-size ; +: In RADIX-2^r, for 16≤N≤64, r=8 (Table 4.9). A zero bit is added in the MSB position to make N a multiple of r (N=24, N=32).

G: Greater than R3 *Upb*; R3 *Upb* = 7, 8, and 10 for N=20, 24, and 32, respectively; *: Values are delivered by Spiral web version, limited to 26 bits; x: Lowest number of additions.

4.8 New MCM Algorithm (RADIX-2^r MCM)

RADIX-2^r SCM algorithm (Section 4.6) can be easily extended to MCM. In MCM, the single variable X is multiplied by a set of N-bit constants $C_1, C_2, C_3, \cdots, C_M$. Therefore, the same non-trivial PP set $\{3\times X, 5\times X, 7\times X, \cdots, (2^{r-1}-1)\times X\}$ can be shared among the M constants C_i. Using the same reasoning developed in Section 4.6, we can easily prove that the upper-bound is equal to:

$$U_{pb}(r) = \left\lceil \frac{M \times N}{r} + 2^{r-2} - 1 - M \right\rceil, \tag{4.44}$$

with $r = 2 \cdot W\left(\sqrt{M \cdot N \cdot log(2)}\right)/log(2)$, where W is the Lambert Function.

We can also easily prove using Bayes's theorem that the average number of additions is equal to:

$$Avg(r) = -M + (1-2^{-r}) \times \lceil (M \times N)/r \rceil + \sum_{j=0}^{\lceil M \times N/r \rceil - 1} \left\{ \sum_{k=1}^{2^{r-2}-1} P(m_{jk}) \times \left[1 - P(m_{jk})\right] j \right\}, \tag{4.45}$$

with $P(m_{jk}) = \dfrac{\left\lfloor \dfrac{2^{r-1}}{2\times k+1} \right\rfloor}{2^{r-1}}$ and $r = 2 \cdot W\left(\sqrt{M.N \cdot log(2)}\right)/log(2)$.

As for the adder-depth (*Ath*), it is exactly the same as in SCM (Eq. 4.27) since the M constants C_i are implemented independently of each other, sharing only the non-trivial PP set.

Fig. 4.10 compares the new RADIX-2^r MCM algorithm to the state-of-the-art for 32-bit constants, with a number of constants varying from 1 up to 100. Note that for Lefèvre, BHM and Hcub, the average is taken *only* over 50 uniformly distributed random constant sets [13]. In RADIX-2^r the average is rather exactly calculated using Eq. 4.45. Nevertheless, RADIX-2^r surpasses Bernstein and Lefèvre, and has a big potential to surpass BHM and competes with Hcub if the optimization techniques (*redundancy and CDE*) employed to R3 are integrated into RADIX-2^r MCM heuristic.

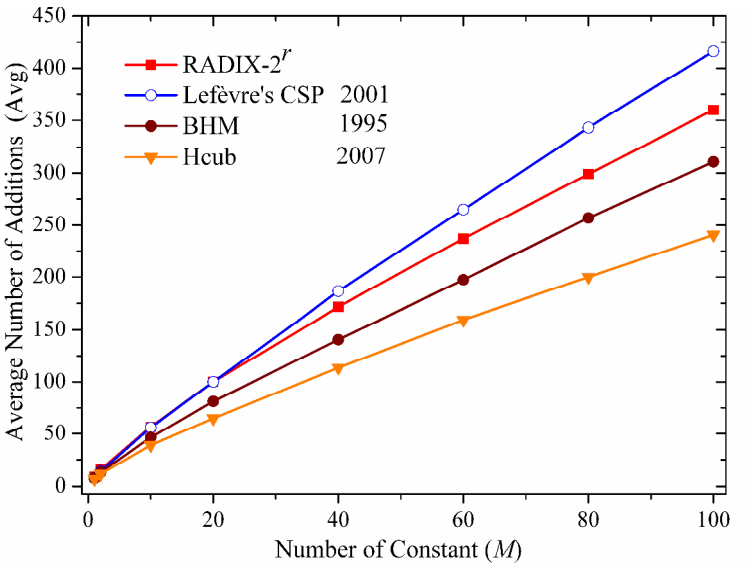


FIGURE 4.10 – *Avg* comparison for 32-bit constants.

Table 4.14 gives the computational complexity of each MCM algorithm that served for comparison. It is important to note that the hidden constant in the big-O notation of Hcub is very important [13].

TABLE 4.14 – RADIX-2^r MCM versus non-recoding MCM algorithms: runtime complexity for a number of M constants with N-bit each.

Algorithm	Runtime [13]
Hcub* [13]	$O(M^3.N^6)$
BHM* [23]	$O(M^3.N^4)$
Lefèvre's CSP [8]	$O(M^3.N^3)$
RADIX-2^r	$O(M.N)$

Table 4.15 compares RADIX-2^r MCM heuristic to the standard CSD technique for 32 and 64-bit constants, with a number of constants varying from 1 up to 100.

TABLE 4.15 – RADIX-2^r MCM versus CSD: *Avg* comparison

M		1	2	10	20	40	60	80	100
N=32	RADIX-2^r	8.71	15.75	56.61	100.05	171.65	236.92	299.07	360.02
	CSD	9.77	19.55	97.77	195.55	391.11	586.66	782.22	977.77
	Saving (%)	10.84	19.43	42.09	48.83	56.11	59.61	61,76	63.17
N=64	RADIX-2^r	16.75	29.40	110.05	191.65	339.07	480.37	607.72	733.58
	CSD	20.44	40.88	204.44	408.88	817.77	1226.7	1635.6	2044.4
	Saving (%)	18.05	28.08	46.17	53.12	58.53	60.84	62.84	64.11

4.9 Conclusion

Based on radix-2^r arithmetic, we have introduced a new *linear-runtime* and *fully-predictable* heuristic (RADIX-2^r) for the multiplication by a constant. An improved version (R3) of RADIX-2^r has also been introduced. It is based on the utilization of a redundant radix-2^r recoding in conjunction with a common-digit-elimination technique. Compared to the existing algorithms, R3 is very competitive especially for large constants. Because of its linear-runtime complexity, R3 remains the unique practical heuristic for huge constants.

On the other hand, we have determined the analytic expressions for the maximum number of additions (upper-bound), the average number of addition (average), and the maximum number of additions forming the critical path (adder-depth). These bounds are the *lowest* bounds known so far for the multiplication by a constant (SCM and MCM). While the bounds are for a minimal set of operations (additions, subtractions, and left-shifts), they remain valid if any other operation (such as right-shifts) is allowed. It is noteworthy to mention that asymptotic worst-case cost of the optimal decomposition remains an *open research problem*.

The predictability feature of RADIX-2^r and R3 (bounds of R3 are lower) enables the generation of fully-predictable LTI-controllers capable of satisfying different requirements, such as:

- Generate a controller comprising the minimum number of additions (most compact controller);
- Generate a controller with the shortest critical-path (fastest controller);
- Enable a trade-off between the number of adders and the number of adder-steps, i.e., between the area and the speed:
  - Given a maximal number of adder-steps, generate a controller that needs a minimal number of adders/subtractors and does not violate the number of adder-steps;
  - Given a delay constraint, generate a controller satisfying the delay constraint such that the number of adders/subtractors is minimal; etc.

CSD has two very-attractive features: ease-of-use and linear runtime complexity. Because of this, CSD is used in designing the vast majority of LTI systems. With the same features and much more interesting compression capabilities, RADIX-2^r and R3 will certainly replace CSD, and therefore allows the design of more compact, faster, and less power consumer LTI-systems.

Since the introduction of H(k) in 2004, CSE heuristics have outperformed DAGs at SCM. This was achieved by applying CSE to each possible signed-digit (SD) form of the constant. Likewise, the search space of CSE can be expanded considering RADIX-2^r recoding instead of SD representation. For such a goal (SCM/MCM), Lefèvre's CSP heuristic stands as the best CSE candidate for its lower computational complexity $O(N^3)$ in comparison to its CSE counterparts. Thus, the combination of Lefèvre's CSP with RADIX-2^r will be a very competitive heuristic.

Finaly, radix-2^r arithmetic is a simple and powerful mathematical tool that might be further explored to derive even *tighter* addition-cost complexities.

Bibliography

[1] J. Thong and N. Nicolici, “An optimal and practical approach to single constant multiplication,” IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 9, pp. 1373–1386, September 2011.

[2] R. Kastner, A. Hosangadi, and F. Fallah, “Arithmetic Optimization Techniques for Hardware and Software Design,” Cambridge University Press, ISBN-13 978-0-521-88099-2, © 2010.

[3] A. Chatterjee, R. K. Roy, and M. A. d’Abreu, "Greedy Hardware Optimization for Linear Digital Systems Using Number Splitting and Repeated Factorization," Proceedings of the sixth IEEE International Conference on VLSI Design, pp. 154-159 , Bombay, India, January 1993.

[4] A. Avizienis, “Signed-digit number representation for fast parallel arithmetic,” IRE Trans. on Electronic Computers, vol. EC-10, No. 3, pp. 389–400, September 1961.

[5] Y.E. Kim et al., “Efficient Design of Modified Booth Multipliers for Predetermined Coefficients,” Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), pp. 2717-2720, Island of Kos, Greece, May 2006.

[6] V.S. Dimitrov, L. Imbert, and A. Zakaluzny, “Multiplication by a Constant is Sublinear,” Proceedings of the 18th IEEE Symposium on Computer Arithmetic (ARITH), pp. 261-268, June 2007.

[7] R.I. Hartley, “Subexpression Sharing in Filters Using Canonic Signed Digit Multipliers,” IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing, vol. 43, No. 10, pp. 677-688, October 1996.

[8] V. Lefèvre, “Multiplication by an Integer Constant,” INRIA Research Report, No. 4192, Lyon, France, May 2001.

[9] N. Boullis and A. Tisserand, “Some Optimizations of Hardware Multiplication by Constant Matrices,” IEEE Trans. on Computers (TC), vol. 54, No. 10, pp. 1271-1282, October 2005.

[10] R.L. Bernstein, “Multiplication by Integer Constant,” Software– Practice and Experience 16, 7, pp. 641-652, 1986.

[11] O. Gustafsson, A.G. Dempster, and L. Wanhammar, “Extended Results for Minimum-Adder Constant Integer Multipliers,” Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), vol. 1, pp. I-73 I-76, Scottsdale Arizona, USA, May 2002.

[12] A. Dempster and M. Macleod, “Using Signed-Digit Representations to Design Single Integer Multipliers Using Subexpression Elimination,” Proceedings of the IEEE International Symp.m on Circuits and Systems (ISCAS), vol. 3, pp. III-165-168, Vancouver, Canada, May 2004.

[13] Y. Voronenko and M. Püschel, “Multiplierless Multiple Constant Multiplication,” ACM Trans. on Algorithms (TALG), vol. 3, No. 2, article 11, pp. 1-38, May 2007.

[14] V.S. Dimitrov, K.U. Järvinen, and J. adikari, “Area Efficient Multipliers Based on Multiple-Radix Representations,” IEEE Trans. on Computers (TC), vol. 60, N° 2, pp 189-201, February 2011.

[15] I.C. Park and H.J. Kang, "Digital filter synthesis based on an algorithm to generate all minimal signed digit representations," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 21, No. 12, pp. 1525-1529, Dec 2002.

[16] J. Thong and N. Nicolici, “Time-efficient single constant multiplication based on overlapping digit patterns,” IEEE Trans. Very Large Scale Integr. Syst., vol. 17, No. 9, pp.1353–1357, Sep. 2009.

[17] J. Thong, "New Algorithm for Constant Coefficient Multiplication in Custom Hardware," Master Thesis, No 4292, McMaster University, Hamilton, Ontario, Canada, October 2009.

[18] Available at: http://en.wikipedia.org/wiki/Big_O_notation

[19] O. Gustafsson, “Lower Bounds for Constant Multiplication Problems,” IEEE Trans. on Circuits and Systems II: Express Brief, vol. 54, No. 11, pp. 974-978, November 2007.

[20] R. W. Reitwiesner, “Binary Arithmetic,” Advances in Computers, New York: Academic, vol. I , pp. 231-308, 1966.

[21] R. G. E. Pinch, “Asymptotic Upper Bound for Multiplier Design,” Electronics Letters, vol. 32, N° 5, pp. 420-421, February 1996.

[22] V.S. Dimitrov, G.A. Jullien, and W.C. Miller, “Theory and Applications of the Double-Base Number System,” IEEE Trans. on Computers (TC), vol. 48, No. 10, pp. 1098-1106, October 1999.

[23] A.G. Dempster and M.D. Macleod, “Use of Minimum Adder Multiplier Blocks in FIR Digital Filters,” IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing 42, 9, pp. 569-567, 1995.

Chapter 5

Multiplication by a Variable

Chapter 5

Multiplication by a Variable

This chapter deals with the problem of hardware optimization of linear-time-variant (LTV) systems. The optimization focuses on the multiplication by a variable (MV) which is the main building block of LTV systems. We formalize the MV problem in radix-2^r arithmetic, and present the two most-known state-of-the-art solutions. We then introduce a series of fully predictable MV algorithms, accompanied with their respective speed and area complexities. Next, we set up the general equation of the space/time partitioning and derive the optimal radix-2^r architecture. At the end, a number of multi-precision-multiplication algorithms are proposed as an extension of the MV problem.

5.1 Optimizations of LTV Systems

The general definition of a linear system has already been given in Section 4.1. An LTV system is rather a system in which certain quantities governing the system's behavior change with time, so that the system will respond differently to the same input at different times.

An LTV system is formalized as follows. If X and Y are input and output vectors, respectively, and V is a transformation matrix, the linear system can be written as

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} & \cdots & V_{1n} \\ V_{21} & V_{22} & \cdots & V_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ V_{m1} & V_{m2} & \cdots & V_{mn} \end{bmatrix} \times \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix}. \tag{5.1}$$

The transformation matrix V is an $m \times n$ matrix, where V_{ij} represents the (i,j) value which can change with time. An output signal Y_i is the product of the ith row of the transformation matrix V and the n input samples of X:

$$Y_j = \sum_{j=1}^{n} V_{ij} \times X_j . \tag{5.2}$$

In Chapter 2, the state-space representation has been retained as the control computational-model to be optimized. Note that the explicit state-space LTV Eq. 2.2 is no more than a concatenation of four $Y=V \times X$ operations given by Eq. 5.1. In Chapter 6 we provide an illustrative example consisting in a PID-controller dedicated to LTV systems. The PID-controller can be tuned on the fly to handle changing environmental conditions.

The computational model of the LTV Eq. 2.2 involves mainly matrix-multiplications (MM), based on the very space/time critical MV operations. In an earlier work, MM operation was the subject of a

thorough investigation in order to optimize the throughput and latency. The results were published in [1][2]. Therefore, MM operation is not reconsidered in this thesis. The whole effort is concentrated on the optimization of the MV operation.

5.2 Formal Definition of the MV Problem

In Section 2.5.3.2, we have given the state-of-the-art of MV problem and have outlined the need for the *high-radix* paradigm to achieve high-speed, low-power, and highly-scalable architecture of the multiplier. These are the three main requirements for today's general purpose multipliers [3]. We have also formulized the radix-2^r multiplier in number of PPGs, number of non-trivial PPs, and number of additions forming the critical path (Fig. 2.9). We have especially insisted on the number of non-trivial PPs as being the major hurdle to designing high-radix multipliers (responsible for very dense PPGs). Reader is encouraged to read Section 2.5.3.2 for a better understanding of the present discussion context.

In fact, non-trivial PPs are not the only problem for a compact PPG. Recoding large slices (r≥8) in a mono-bloc PPG such as in [4][5], requires the use of an RTL "case statement" with r+1 entries. In this case, 2^{r+1} combinations must be processed, which yields to a huge amount of multiplexer resources. Thus, mono-bloc PPG recoding is incompatible with high-radix (r≥8) approach whose purpose is to reduce the multiply-time (N/r) of large operand size (N ≥32) multipliers.

To overcome these two above-mentioned shortcomings, a new radix-2^r design concept is proposed.

5.2.1 New Radix-2^r Design concept

To achieve such a goal, the multibit recoding multiplication algorithm is revisited [6]. Its design space is extended by the introduction of a new recursive version that enabled to solve the hard problem of radix-2^r two's complement multiplication for any value of r. The solution consists essentially in dividing the high radix-2^r mono-bloc PPG_j (Fig. 5.1a) into a number of lower sub-radix-2^s odd-multiple free PPG_{ji} (Fig. 5.1b), such as s is a divider of r. The direct benefits of the new partitioning of Fig. 5.1b are:

- There is no need to pre-compute non-trivial PP, which drastically reduces the required amount of hardware resources and routing;
- Since the size of PPG_{ji} entry is much smaller than the size of PPG_j one (s≤r/2), the total multiplexing logic required by RTL "case statements" to recode the entries is greatly reduced;
- The possibility to simultaneously process larger bit slices (r≥16) radically shortens the critical path in terms of adder levels, especially for very large operand sizes (N≥64).

Formal problem statement. Aided by Fig. 5.1b, now we can formally state the problem of constructing multiplier blocks.

Definition 5.1 – MV problem: *Given N, find the couple (r, s) that leads to the shortest critical path in adder stages and to the minimum logic resources inside PPGs.*

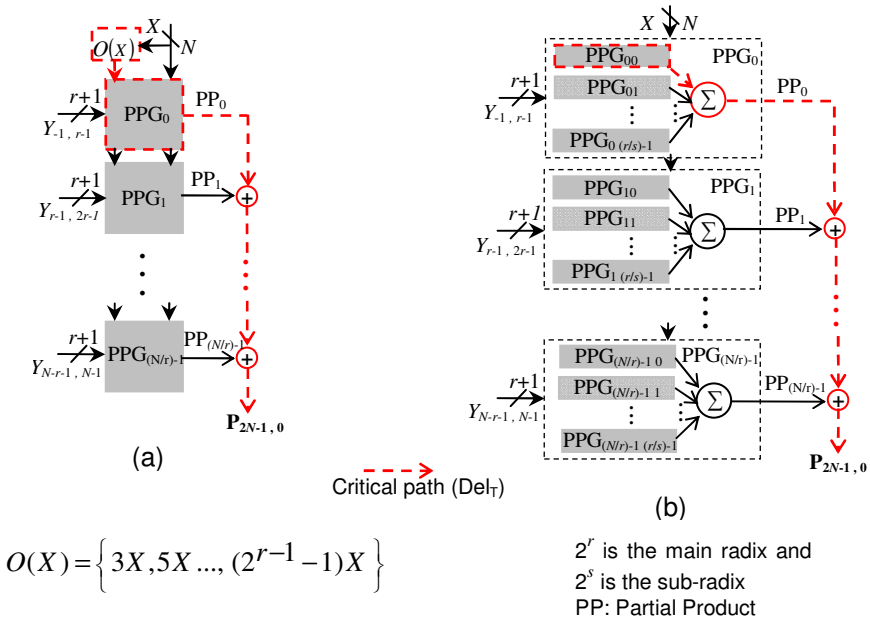


FIGURE 5.1 – Generalized $N \times N$ bit radix-2^r parallel multiplier.

a) Critical path in conventional [6][7][8][9][10][11] and recent [4][5][12][13][14] radix-2^r multipliers. $O(X)$ is the necessary set of odd-multiples corresponding to radix-2^r recoding. PPG_j of [4][5] includes a number of adders to accumulate intermediary partial product.

b) Critical path in our proposed radix-2^r multipliers. Main features are: no odd-multiples, much more compact PPG_j, much shorter critical path.

5.2.2 New MV Complexity

In [5] a number of MV complexities are cited in a chorological order. They are summarized hereafter and compared to our new MV complexity.

In the mid-1950s, Kolmogorov made a conjecture that any multiplication algorithm will require $\Omega(N^2)$ elementary additions, where N is the binary length of the operands. This conjecture has been disproved in a constructive manner by Karatsuba who proposed an algorithm that uses $O(N^{1.585})$ additions. in 1971, Schönhage and Strassen published an algorithm with asymptotic complexity $O(N \log N \log\log N)$. For almost 40 years, this result has not been improved; however, in 2007 Fürer designed an algorithm with lower asymptotic complexity, namely, $O(N \log N \log\log \cdots \log N)$ operations. Finally, in 2011, using DBNS arithmetic, Dimitrov proved that MV can be performed in $2N^2 / \log N$ additions.

In Eq. 4.28 we have proved an upper-bound for SCM. The latter can be easily extended to MV problem, giving:

$$\left\lceil \frac{N^2}{r} + N \times \left(2^{r-2} - 2\right) \right\rceil \tag{5.3}$$

with $r = 2 \cdot W\left(\sqrt{N \cdot log(2)}\right) / log(2)$ and W is the Lambert's function.

Eq. 5.3 is actually the lowest upper-bound known so far for the MV problem. Whether or not it is possible to multiply two numbers in purely linear number $O(N)$ of additions is still an open problem.

The above-mentioned algorithms (Karatsuba's, Schönhage-Strassen's, and Fürer's) are all having a subquadratic complexity. However, the implicit constant, associated with the big-O notation, is very large and this severely limits their applicability to problems of practical importance. Karatsuba's multiplication outperforms classical shift-and-add algorithm if the size of the operands is around 1,000 bits. The exact break-even point is platform dependent. This makes it suitable for specific cryptographic applications. The algorithms by Schönhage-Strassen and Fürer are useful if one deals with extremely large numbers. Applications include computational number theory and computations associated with the search of large Mersenne primes and finding divisors of Fermat numbers. In those cases, we deal with numbers having more than 1 million decimal digits [5].

5.3 High-Radix Multiplication Algorithms

While the multibit recoding multiplication algorithm [6] is mathematically attracting, it suffers from a serious limitation: high-radices require high number of non-trivial PPs. This is the main reason why it was abandoned. Moreover, in industry commercial designs do not exceed r=4 (radix-16). A hybrid radix-4/8 is proposed in [12] for low-power multimedia applications. To increase the speed of the multiplier, most ancient processors employed radix-8, such as: Fchip [7], IBM S/390 [8], Alpha RISC [9], IA-32 [10] and AMDK7 [11]. While radix-16 is used only in the most recent Intel processors: 64 and IA-32 [13], and Itanium-Poulson [14].

In research, the highest radices are used in the algorithms proposed by Seidel [4] and Dimitrov [5].

5.3.1 Dimitrov's DBNS Algorithm

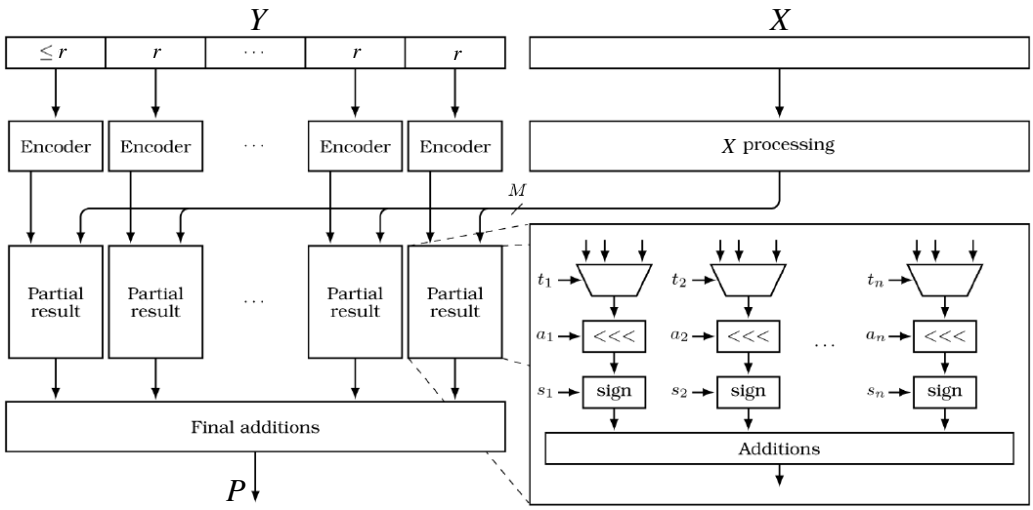
The biggest advantage of the DBNS multiplier is the fact that it has a provably subquadratic complexity ($2N^2/\log N$). The latter guarantees that, eventually, it will outperform the shift-and-add based algorithms for certain range of multiplicands. The biggest practical problems are: when will it happen; and how to apply the algorithm as efficiently as possible on hardware.

We give a general description of the proposed multipliers. Let Y and X be two N-bit unsigned integers, i.e., $Y, X \in [0\ ,\ 2^N - 1]$ and let P denote the 2N-bit result of the multiplication $P = Y \times X$. All proposed multipliers compute the entire product in parallel combinatorially, i.e., without registers or feedback loops. The general structure of all multipliers that are proposed is depicted in Fig. 5.2. Y is split using r-bit windows into $\lceil N/r \rceil$ blocks. Each r -bit block, Y_i, is fed into an encoder. It encodes a block, i.e., an integer in the interval $[0\ ,\ 2^r - 1]$, as the following sum of k terms:

$$(-1)^{s_1} 2^{a_1} 3^{t_1} + (-1)^{s_2} 2^{a_2} 3^{t_2} + \cdots + (-1)^{s_k} 2^{a_k} 3^{t_k}, \tag{5.4}$$

where $a_i \in [0, r]$, $s_i \in \{0, 1\}$, and $t_i \in [0, m]$, where m is the predefined highest power of three allowed by the representation. The encoder is essentially a table with 2^r rows (one for each integer represented by the block), each containing n triples of the form (s_i, a_i, t_i).

The operand X is fed into a circuit that computes $X_i = 3^{t_i} \times X$ for $i = 0, \cdots, m$. These computations are carried out with shifts, additions, and subtractions. Each partial result circuit computes Eq. 5.4 by, first, selecting the correct X_i from the values computed in the X processing. They are, then, shifted by a_i bit positions to the left (multiplication by 2^{a_i}), and, finally, added or subtracted as described by the sign bits, s_i, to receive the partial result $Y_i \times X$. The result of the entire multiplication, $P = Y \times X$, is computed by shifting and adding the partial results.



Source: [5]

FIGURE 5.2 – The general structure of the DBNS multiplier.

If for instance, we would like to use a window of size seven (r=7), then we have to make sure that any integer between 0 and 127 can be represented by using the corresponding number representation. For that purpose Eq. 5.4 has undergone a number of modifications to allow more degree of freedom in the choice of the associated digits [5]. Because if we want to use the double-base number representation in Eq. 5.4, then we will have to use three terms, because certain numbers less than 127 cannot be represented as the sum or difference of two {2,3}-integers (as it is pointed above, the smallest positive integer with this property is 103). On the other hand, if one uses representation in the modified form of Eq. 5.4, then it is sufficient to use digits $d_i \in \{1,3,5,7\}$ which guarantee a representation for every 7-bit integer in the modified form of Eq. 5.4 using at most two terms. The following fact will be used in the design, so we shall specifically acknowledge it:

Fact – *Every nonnegative 7-bit integer can be represented in the form* $z_1 \pm z_2$, *where* $z_1, z_2 \in \{1 \times 2^k, 3 \times 2^k, 5 \times 2^k, 7 \times 2^k, \}$ *with* $k = \{0,1,\cdots,7\}$.

Note – *The smallest number for which the above fact is not valid is* 137*, i.e., an* 8*-bit number.*

So, from a point of view of integer representations, this new number representation is more attractive compared to DBNS. In order to cover the same range (7-bit numbers) with the DBNS, one must use the digit set {1, 3, 9, 27, 81}, and more importantly three terms.

After examining many options (different digit sets with different numbers of terms), Dimitrov has concluded that for multipliers, it is optimal to have two summands (as in the above-explained case with

7-bit numbers) and a carefully selected set of digits. This particular encoding can be formally expressed as follows:

$$\pm z_1 \pm z_2 \tag{5.5}$$

where $z_1 \in \{a_1 \times 2^k, a_2 \times 2^k, \cdots, a_s \times 2^k,\}$ and $z_2 \in \{b_1 \times 2^k, b_2 \times 2^k, \cdots, b_l \times 2^k,\}$ for $k = 0,1,\cdots,r.$

The determination of the sets $D_a = \{a_1, a_2, \cdots, a_s\}$ and $D_b = \{b_1, b_2, \cdots, b_l\}$ is the cornerstone of the proposed algorithm. In the above example with 7-bit integers, the sets of digits were $D_a = D_b = \{1,3,5,7\}$. For a better implementation of the multiplication, it is preferable:

- To select those digits in such a way as to encode every *r*-bit number in the form $z_1 \pm z_2$ (that is, the first summand is always positive), then it will lead to smaller area complexity of the design.
- To fix the signs of both the summands, z_1 and z_2 (that is, if we represent every r -bit integer as either $z_1 + z_2$ or $z_1 - z_2$), then we can expect further hardware simplifications due to the elimination of the necessity to process the sign of the second summand.
- To use a window (*r*) of size six at least, and 11 at most, for multiplication of integers of medium size: $64 \times 64, 128 \times 128,$ and 256×256. For sizes that would be useful, e.g., in RSA cryptography, we may need larger windows if the hardware resources allow that.

Several multipliers using the above-discussed representations were described in VHDL in order to find out how they perform in practice. The representations used in these multipliers are collected in Table 5.1. They were carefully selected from many possibilities because they appeared to have very attractive features in theory and/or practice as discussed in the previous sections.

The multipliers of Table 5.1 were implemented on 0.18μm CMOS technology using Synopsis Design Compiler Ultra. They were compared to 32×32 and 64×64 bit add-and-shift and radix-8 references designs.

According to the results presented in [5], all of the multipliers are better than the add-and-shift reference multiplier in both area and power in 64-bit multiplication. Further, all the multipliers outperform radix-8 reference multiplier in 64-bit multiplication in terms of power consumption. Most of the multipliers have better area figures than radix-8 reference multiplier.

We can notice that some of the multipliers are better in terms of area than array-based reference multiplier in 32-bit multiplication. All of the multipliers are having more power consumption than both reference multipliers in 32-bit multiplication. The area consumption of 32-bit radix-8 multiplier is always better than any of the proposed multipliers. Mult_6bsms design is having the best performance in both area and power consumption among all designs, followed by Mult_7b2d multiplier.

The results show the delicacy of selecting the representations. The quality of the results varies considerably even between representations which, at first sight, have only little difference. The effects of the conditions discussed previously are clearly visible in the results. For instance, the “something-minus-something” (SMS) encodings, where the first term is always positive and the second term is

negative, show an advantage over other encodings with the same r. The sizes of the encoders start to play a significant role in the area complexity when r increases. This diminishes the feasibility of representations with large r, such as Mult_11b3d, although they appear attractive in theory because of the low total number of additions/subtractions.

It is important to mention the following:

- Although several multipliers have been proposed, there is no formal study (search of the optimum) on the size of the window r. We have no exact idea on how r evolves when the size of the operands N increases (there must be some sort of correlation);
- All recodings proposed in Table 5.1 are not DBNS recodings, though the starting Eq. 5.4 allows a DBNS recoding. Eq. 5.4 has been radically modified [5];
- All proposed algorithms are unsigned. Handling the two's complement representation will make the recoding harder.

TABLE 5.1 – Dimitrov's high-radix DBNS algorithms.

Name	Recoding	r	Digit Sets
Mult_6b2d7	$Y=\sum_{j=0}^{(N/r)-1}\left(2^k Q_j+(-1)^e 2^h P_j\right)2^{rj}$	6	$Q_j \in \{1,3,5,7\}$; $P_j \in \{1,3\}$
Mult_6b2d9	$Y=\sum_{j=0}^{(N/r)-1}\left((-1)^e 2^k Q_j+(-1)^f 2^h P_j\right)2^{rj}$	6	$Q_j \in \{1,3,5,7\}$; $P_j \in \{1,3\}$
Mult_6bsms	$Y=\sum_{j=0}^{(N/r)-1}\left(2^k Q_j-2^h P_j\right)2^{rj}$	6	$Q_j, P_j \in \{1,3,5,7\}$
Mult_7b2d	$Y=\sum_{j=0}^{(N/r)-1}\left(2^k Q_j+(-1)^e 2^h P_j\right)2^{rj}$	7	$Q_j, P_j \in \{1,3,5,7\}$
Mult_7bsms	$Y=\sum_{j=0}^{(N/r)-1}\left(2^k Q_j-2^h P_j\right)2^{rj}$	7	$Q_j, P_j \in \{1,3,5,7,89,125\}$
Mult_8b2dd	$Y=\sum_{j=0}^{(N/r)-1}\left(2^k Q_j+(-1)^e 2^h P_j\right)2^{rj}$	8	$Q_j \in \{1,3,5,7,11,89\}$ $P_j \in \{1,3,5,7,11,119\}$
Mult_8b2di	$Y=\sum_{j=0}^{(N/r)-1}\left((-1)^e 2^k Q_j+(-1)^f 2^h P_j\right)2^{rj}$	8	$Q_j, P_j \in \{1,3,7,11,15,19,25\}$
Mult_8b3d	$Y=\sum_{j=0}^{(N/r)-1}\left((-1)^e 2^k Q_j+(-1)^f 2^h P_j+(-1)^g 2^l S_j\right)2^{rj}$	8	$Q_j \in \{1\}$; $P_j \in \{3\}$ $S_j \in \{7,17\}$
Mult_8bsms	$Y=\sum_{j=0}^{(N/r)-1}\left(2^k Q_j-2^h P_j\right)2^{rj}$	8	$Q_j, P_j \in \{1,3,5,7,9,11,13,15\}$
Mult_9b2d	$Y=\sum_{j=0}^{(N/r)-1}\left(2^k Q_j+(-1)^e 2^h P_j\right)2^{rj}$	9	$Q_j, P_j \in \{1,3,5,7,9,11,13,15\}$
Mult_11b3d	$Y=\sum_{j=0}^{(N/r)-1}\left(2^k Q_j+(-1)^f 2^h P_j+(-1)^g 2^l S_j\right)2^{rj}$	11	$Q_j, P_j, S_j \in \{1,3,5,7\}$

$e, f, g \in \{0,1\}$; $k, h, l \in \{0,1,\cdots,r\}$

5.3.2 Seidel's RNS Algorithm

In Seidel's algorithms [15] [16], a secondary radix multiplier recoding of the N-bit integer $Y = \sum_{j=0}^{N-1} y_j \times 2^j$ is performed. It is illustrated as a 2-step process:

- Step 1 (High Radix Booth): The operand Y is recoded to a $N' = \lceil (N+1)/r \rceil$-digit minimally redundant (Booth) radix polynomial representation $Y = \sum_{j=0}^{N'-1} d_j \times (2^r)^j$ in a primary radix $\beta = 2^r$, where $-2^{r-1} \le d_j \le 2^{r-1}$ for $0 \le j \le N'-1$. Authors are particularly interested here in high primary radices determined by $5 \le r \le 16$, similar to the size of radices in high radix (byte) division.
- Step 2 (Secondary Radix Reduction): Each primary (Booth) digit value $-2^{r-1} \le d_j \le 2^{r-1}$ is recoded to a k-digit value in a secondary radix system. We limit our focus here to $2 \le k \le 4$. For the case $k = 2$ and secondary radix γ, we recode d by two digits so that $d = d_1 \times \gamma + d_0$, where digits d_1 and d_0 are chosen from a balanced complete residue system modulo γ having a novel design.

The N-bit integer can then be expressed as the weighted sum:

$$Y = \gamma \times \sum_{j=0}^{N'-1} d_{1j} \times (2^r)^j + \sum_{j=0}^{N'-1} d_{0j} \times (2^r)^j \tag{5.6}$$

Note that the generalization for arbitrary k of Eq. 5.6 partitions the sum of partial products into k independent secondary digit summations with respective weights $\gamma^0, \gamma^1, \cdots, \gamma^{k-1}$ with $2 \le k \le 4$ considered herein.

Illustrative Example (k=2)

The primary radix is $\beta = 2^5$ with Booth digit set $D = \{-16,-15,\cdots,16\}$. The secondary radix is $\gamma = 7$ with a digit set $D = \{-4,-2,-1,0,1,2,4\}$ having only signed binary power or zero digits. Note that every digit $-16 \le d_j \le 16$ of the primary radix system can be represented as a two digit radix-7 number $d_j = d_{1j} \times 7 + d_{0j}$, with $d_{1j}, d_{0j} \in \{-4,-2,-1,0,1,2,4\}$. Reader is referred to [15] [16] for the recoding table.

A 64×64 bit product $P = Y \times X$ using a secondary radix representation for Y can be expressed as:

$$P = (7 \times X) \times \sum_{j=0}^{12} d_{1j} \times (2^5)^j + X \times \sum_{j=0}^{12} d_{0j} \times (2^5)^j. \tag{5.7}$$

The right hand side of Eq. 5.7 has 26 partial products, achieving a reduction more than halfway that of Booth radix 4 and 8. These 26 partial products are partitioned into two groups, 13 of which employ the

primary ($7\times X$) and 13 of which employ (X) giving a PPG fanin of only 26. Two options are possible with these simplified partial products noting that $d_{ij}\times(2^5)^j=(-1)^s\times 2^n$ or 0 for all $0\le j\le 12,\ 0\le i\le 1$.

- Pre-compute ($7\times X$): The primary partial product can be pre-computed by a shift and add ($7\times X = X\times 2^3 - X$) while the d_{ij} are obtained from a recoder or recoding table.

- Post-compute ($7\times X$): The higher order summation can utilise a 13:2 adder tree compressing $\sum_{j=0}^{12} X\times\left[d_{1j}\times(2^5)^j\right]$ to a redundant (e.g. carry save) sum z. then the post computation can add $z\times 2^3 - z$ to the low order sum $t=\sum_{j=0}^{12} X\times\left[d_{0j}\times(2^5)^j\right]$ output from a second 13:2 adder tree. The value of $z\times 2^3 - z + t$ is completed by a 6:2 compressor and a 2–1 addition.

Note that the post-computation option (Fig. 5.3) utilizes only two more partial products and one additional level of 3-to-2 adder delay to avoid the complexity of a 2–1 adder to pre-compute ($7\times X$).

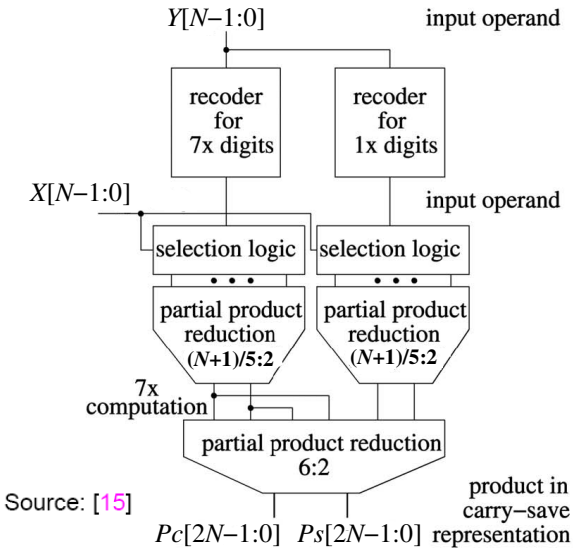


FIGURE 5.3 – Recoding (32,7) with postcomputation of $7\times$.

Illustrative Example (k=3)

The following encoding scheme is based on a radix-2^8 signed digit representation of the multiplier: $\sum_{j=0}^{N'-1} d_j\times(2^8)^j$, so that the multiplier is represented by $N'=\lceil(N+1)/8\rceil$ radix-2^8 digits $d_j\in\{-128,-127,\cdots,127,128\}$. As previously suggested each radix-2^8 d_j can be represented by three digits in the secondary radix-11: $d_j = d_{2j}\times 11^2 + d_{1j}\times 11 + d_{0j}$, where each of the digits d_{2j}, d_{1j}, and d_{0j} in the secondary radix representation is a power of two (see recoding table in [16]).

The high order radix-11 digits d_{2j} can only have values from the set { –4, –2, –1, 0,1,2,4} and the middle and the low order radix-11 digits d_{2j} and d_{1j} can only have the values {–16, –8, –4, –2, –l,0, 1,2,4,8,16}. In this case the partial products corresponding to the high order digits and the middle digits have to be weighted by 121 and 11, respectively.

In [15], three other recodings are proposed with secondary radices $\gamma = 17, 23, 31$, corresponding to k=4,2,3 and primary radices 2^{15}, 2^{8}, 2^{13}.

Selecting Secondary Radix Recoding

A secondary radix recoding procedure involves the joint selection of a primary radix $\beta = 2^r$, a secondary radix (modulus) γ, a balanced complete residue system modulo γ forming a secondary radix digit set D_γ, and a number of digits k, $2 \le k \le 4$, such that the k-digit secondary radix values include all integers in the Booth high radix digit range $\left[-2^{r-1}, 2^{r-1}\right]$.

The search for practically useful tuples $(\beta, \gamma, D_\gamma, k)$ starts by considering the relatively small number of secondary radix candidates γ that might have sufficient properties to support a reasonable secondary radix recoding in competition with standard Booth recodings. Several key criteria become evident to prune the candidate search space of values for γ [15].

To allow the representation of all members of the contiguous integer interval $\left[-2^{r-1}, 2^{r-1}\right]$ with from two to four digits, it is useful to first consider the case that D_γ is a complete residue system modulo γ. Since complements and shifts can be employed to increase the range of digit values as traditionally employed in Booth recoding, all nonzero digits should be of the form $\pm\delta 2^i$, where δ is a member of the store {1}, {1,3}, or {1,3,5}. Note that residue digit sets of the form $D_\gamma = \{0, \pm d_1, \pm d_2, \cdots, \pm d_{(\gamma-1)/2}\}$ are termed balanced complete residue systems when every integer i with $0 \le i \le \gamma - 1$ is congruent to some (and necessarily exactly one) member of D_γ.

It is important to note that the digit values of D_γ that must form a balanced complete residue system need *not* form a contiguous integer sequence as do the traditional primary Booth digit sets. It is only necessary that the k-digit values radix γ cover the contiguous integer interval $\left[-2^{r-1}, 2^{r-1}\right]$. This flexibility is best utilized by finding sets D_γ, where the maximum digit is not too large and, further, where the smallest odd magnitude that is not a digit of D_γ is not too small. For this latter reason, are not consider digit sets D_γ with the odd multiple store for δ having {1x, 5x} without 3x.

It should be noted that, for a 2-signed-bit secondary radix $\gamma = 2^i \pm 1$, the digit set D_γ will necessarily contain $D_\gamma = \{0, \pm 1, \pm 2, \cdots, \pm 2^{i-1}\}$. This set is a complete residue system for $\gamma = 7$ with $D_\gamma = \{0, \pm 1, \pm 2, \cdots, \pm 4\}$ and employs only the 1x multiplicand in the store.

5.4 New Radix-2^r Multiplication Algorithms

The equation (2.1.2) of the original multibit recoding algorithm presented in [6] does not offer hardware visibility. Let us rewrite it in a simpler hardware-friendly form, as follows:

$$Y = \sum_{j=0}^{(n/r)-1} \left(y_{rj-1} + 2^0 y_{rj} + 2^1 y_{rj+1} + 2^2 y_{rj+2} + \cdots + 2^{r-2} y_{rj+r-2} - 2^{r-1} y_{rj+r-1} \right) \times 2^{rj}$$

$$= \sum_{j=0}^{(n/r)-1} Q_j \times 2^{rj} \ , \qquad (5.8)$$

where $y_{-1} = 0$ and $r \in \mathrm{N}^*$. For simplicity purposes and without loss of generality, we assume that r is a divider of N. In Eq. 5.8, the two's complement representation of the multiplier Y is split into N/r two's complement slices (Q_j), each of r+1 bit length. Each pair of two contiguous slices has one overlapping bit.

In literature, Eq. 5.8 is referred to by radix-2^r equation, to which corresponds a digit set $D(2^r)$ such as $Q_j \in D(2^r) = \{-2^{r-1}, \cdots, 0, \cdots, 2^{r-1}\}$. Thus, the signed multiplication between X and Y becomes:

$$X.Y = \sum_{j=0}^{\frac{N}{r}-1} X.Q_j.2^{rj} \ , \qquad (5.9)$$

Where each partial product can be expressed as follows: $X.Q_j.2^{rj} = (-1)^e.2^f.(m.X)$, with $m \in O(2^r) = \{ 1, 3, \cdots, 2^{r-1}-1 \}$ such as $|O(2^r)| = 2^{r-2}$. $O(2^r)$ represents the required set of odd-multiples of the multiplicand ($m.X$) for radix-2^r. Hence, the partial-product generation-process consists first in selecting one odd-multiple ($m.X$) among the whole set of pre-computed odd-multiples, which is then submitted to a hardwired shift of f positions, and finally conditionally negated $(-1)^e$ depending on the bit sign e of Q_j term. Table 5.2 provides a picture on how the number of odd-multiples grows when the radix becomes higher. While lower $m.X$ can be obtained using just one addition ($3X$=$2X$+$1X$), the calculation of higher ones may require a number of computation steps ($11X$= $8X$+$2X$+$1X$).

TABLE 5.2 – Main features of the multibit recoding multiplication algorithm.

Radix	Number of Partial Products	Odd Multiples ($m.X$)
2^1	N	$1X$
2^2	$N/2$	$1X$
2^3	$N/3$	$1X$, $3X$
2^4	$N/4$	$1X$, $3X$, $5X$, $7X$
2^5	$N/5$	$1X$, $3X$, $5X$, $7X$, $9X$, $11X$, $13X$, $15X$

$O(2^{r+1})|=2\times|O(2^r)|$. In radix-$2^r$, the multiplier Y is divided into N/r slices, each of r+1 bit length. Each pair of two contiguous slices has one overlapping bit.

To bypass the hard problem of odd-multiples, we exploit the fact that the N+1 bit-length *two's complement* multiplier Y on which Eq. 5.8 is applied, is composed of a series (N/r) of r+1 bit-length *two's complement* slices (Q_j digits) on which Eq. 5.8 can be recursively applied again. Based on this

observation, we have already announced Th. 4.3 and 4.4 accompanied with their respective proofs, inserted in Appendix A. When Th. 4.3 is applied to Eq. 5.8, it gives:

$$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{s}-1} P_{ji}\, 2^{si} \right] 2^{rj} \ , \tag{5.10}$$

where $P_{ji} \in D(2^s) = \{-2^{s-1}, \ldots, 0, \ldots, 2^{s-1}\}$ with $O(2^s) = \{ 1, 3, \cdots, 2^{s-1}-1 \}$ such as $\frac{|O(2^r)|}{|O(2^s)|} = 2^{ks}$

and

$$X.Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{s}-1} X.P_{ji}\, 2^{si} \right] 2^{rj} . \tag{5.11}$$

Likewise, when Th. 4.4 is applied to Eq. 5.8, we obtain:

$$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{s+t}-1} \left[P_{ji} + T_{ji}\, 2^{s} \right] 2^{(s+t)i} \right] 2^{rj} , \tag{5.12}$$

where $P_{ji} \in D(2^s) = \{-2^{s-1}, \cdots, 0, \cdots, 2^{s-1}\}$ with $O(2^s) = \{ 1, 3, \cdots, 2^{s-1}-1 \}$ and $T_{ji} \in D(2^t) = \{-2^{t-1}, \cdots, 0, \cdots, 2^{t-1}\}$ with $O(2^t) = \{ 1, 3, \cdots, 2^{t-1}-1 \}$ such as $\frac{|O(2^r)|}{|O(2^{s+t})|} = 2^{k(s+t)}$

and

$$X.Y = \sum_{j=0}^{\frac{n}{r}-1} \left[\sum_{i=0}^{\frac{r}{s+t}-1} \left[X.P_{ji} + X.T_{ji}\, 2^{s} \right] 2^{(s+t)i} \right] 2^{rj} . \tag{5.13}$$

Th. 4.3 and 4.4 allow an exponential reduction ($1/2^{ks}$ and $1/2^{k(s+t)}$, resp.) of the number of odd-multiples in Eq. 5.11 and 5.13 in comparison to Eq. 5.9, but at the expense of a linear increase ($ks-1$ and $k(s+t)-1$, resp.) in the number of additions. The advantage by far outweighs the cost, as practically shown in the next section.

The translation of Eq. 5.11 into architecture is depicted by Fig. 5.1b, where each PPG_j (Q_j) is built up using r/s identical PPG_{ji} (P_{ji}). This is not the case for Eq. 5.13 which requires two different PPG_{ji} (P_{ji} and T_{ji}) . Th. 4.3 and 4.4 can be merged together to produce PPG_j made of a number of different PPG_{ji} ($P_{ji}, T_{ji}, U_{ji}, V_{ji}, \ldots$). This is the general case that is thoroughly studied in next sections in order to determine the optimal multiplier.

5.4.1 Two New High Radix (2^8 and 2^{16}) Illustrative Examples

Th. 4.3 and 4.4 permit to build up any high radix-2^r multiplication algorithm based on lower sub-radices, employing much less odd-multiples. The objective hereafter is to generate high radix-2^r multiplication *without* odd-multiples for a maximum reduction of multiplexer complexity inside PPG_j. To achieve such a goal, a number of odd-multiple free low-radix algorithms are used, such as Booth

algorithm (radix-2^1) [17], modified Booth algorithm (radix-2^2) [18], Seidel et *al.* algorithms (radix-2^5 and radix-2^8) [15][16]. Booth and modified Booth recoding (McSorley algorithm [18]) can be derived from Eq. 5.10 for $(r,s)=(1,1)$ and $(r,s)=(2,2)$, respectively. They are respectively summarized as follows:

$$Y = \sum_{j=0}^{N-1} \left(y_{j-1} - y_j \right) 2^j = \sum_{j=0}^{N-1} Q_j \; 2^j \,, \tag{5.14}$$

with $D\left(2^1\right) = \{-1, 0, 1\}$ and $O\left(2^1\right) = \{1\}$;

$$Y = \sum_{j=0}^{(N/2)-1} \left(y_{2j-1} + y_{2j} - 2y_{2j+1} \right) 2^{2j} = \sum_{j=0}^{(N/2)-1} Q_j \; 2^{2j} \,, \tag{5.15}$$

with $D\left(2^2\right) = \{-2, -1, 0, 1, \; 2\}$ and $O\left(2^2\right) = \{1\}$.

Seidel radix-2^5 recoding [15][16] is described as follows:

$$Y = \sum_{j=0}^{(N/5)-1} \left[7.Q_j + P_j \right] 2^{5j} \,, \tag{5.16}$$

with $Q_j \in \{-2, -1, 0, 1, 2\}$; $P_j \in \{-4, -2, -1, 0, 1, 2, 4\}$ and $O\left(2^5\right) = \{1\}$.

And Seidel radix-2^8 recoding is given by the following equation:

$$Y = \sum_{j=0}^{(N/8)-1} \left[11^2.Q_j + 11.P_j + T_j \right] 2^{8j} \,, \tag{5.17}$$

with $Q_j \in \{-2, -1, 0, 1, 2\}$; P_j , $T_j \in \{-16, -8, -4, -2, -1, 0, 1, 2, 4, 8, 16\}$ and $O\left(2^8\right) = \{1\}$. Note that while Eq. 5.16 and 5.17 are odd-multiple free since all included digits are power of 2. They require a post-accumulation to deal with odd numbers (7, 11 and 121). Thus, a number of extra-adders are needed.

Optimized higher radices are obtained as follows.

5.4.1.1 New Radix-2^8 Recoding

Based on Th. 4.4, each 8+1 bit slice is split into 5+1, 2+1, and 1+1 overlapping slices using Seidel radix-2^5, McSorley radix-2^2, and Booth radix-2^1 algorithms, respectively. The new recoding is given by the following equation:

$$Y = \sum_{j=0}^{(N/8)-1} \left[\left(7.Q_j + P_j \right) + \left(R_j + S_j \, 2^2 \right).2^5 \right].2^{8j} \,, \tag{5.18}$$

with $Q_j \in \{-2, -1, 0, 1, 2\}$; $P_j \in \{-4, -2, -1, 0, 1, 2, 4\}$; $R_j \in \{-2, -1, 0, 1, 2\}$; $S_j \in \{-1, 0, 1\}$ and $O\left(2^8\right) = \{1\}$.

5.4.1.2 New Radix-2^{16} Recoding

Likewise, using Th. 4.4, each 16+1 bit slice is split into 8+1, 5+1, 2+1, and 1+1 overlapping slices using Seidel radix-2^8 and radix-2^5, McSorley radix-2^2, and Booth radix-2^1 algorithms, respectively. The new recoding is described by the following equation:

$$Y = \sum_{j=0}^{\frac{N}{16}-1} \left[\left(11^2.Q_j + 11.P_j + T_j\right) + \left(7.R_j + S_j\right).2^8 + \left(U_j + V_j 2^2\right).2^{13} \right].2^{16j}, \qquad (5.19)$$

with $Q_j \in \{-2,-1,0,1,2\}$; $P_j, T_j \in \{-16,-8,-4,-2,-1,0,1,2,4,8,16\}$; $R_j \in \{-2,-1,0,1,2\}$; $S_j \in \{-4,-2,-1,0,1,2,4\}$; $U_j \in \{-2,-1,0,1,2\}$; $V_j = \{-1, 0, 1\}$ and $O\left(2^{16}\right) = \{1\}$.

In [19][20], we have pursued this combination process farther and generated a series (Appendix B) of higher radix (2^{24}, 2^{32}, …) recoding schemes with $O\left(2^r\right) = \{1\}$. However, what still remains unknown is to determine, for a given value N, the proper radix (2^r) that leads to the *optimal* architecture.

The translation of Eq. 5.18 and 5.19 into architectures are depicted in Fig. 5.4a and 5.4b, respectively.

All Dimitrov algorithms developed in [5] are unsigned. For an equitable comparison, we had to develop a new two's complement radix-2^8 recoding version with $O\left(2^8\right) = \{1,3,5,7\}$ based on Dimitrov unsigned radix-2^7 recoding (mult_7b2d in [5]) with $O\left(2^7\right) = \{1,3,5,7\}$. The new recoding is:

$$Y = \sum_{j=0}^{(n/8)-1} \left(2^k . Q_j + (-1)^e 2^h . P_j\right) (-1)^{8j+7} 2^{8i} \qquad (5.20)$$

with $Q_j, P_j \in \{1,3,5,7\}; k,h \in \{0,1,2,3,4,5,6,7\} and\ e \in \{0,1\}$.

For the comparative study, our proposed algorithms (Eq. 5.18 and 5.19) as well as Seidel and Dimitrov algorithms (Eq. 5.17 and 5.20, resp.) are first analytically characterized and then physically implemented on FPGA.

5.4.1.3 Analytical Characterization of Area and Speed

Prior implementation, we need to develop a generalized theoretical model which predicts area and speed features of each recoding algorithm with respect to N and r values.

Area

Three basic components are necessary for the implementation of RTL multipliers:

- multiplexers ($Mux1$) to recode the digit terms (Q_j,P_j,…) included in the recoding expression;
- shifters ($Mux2$) for partial product generation;
- and adders for partial product summation.

Whereas the exact number of adders can be known in advance, we need to develop heuristics for the two others. The total multiplexer complexity ($Mux1$) of a radix-2^r multiplier depends on:

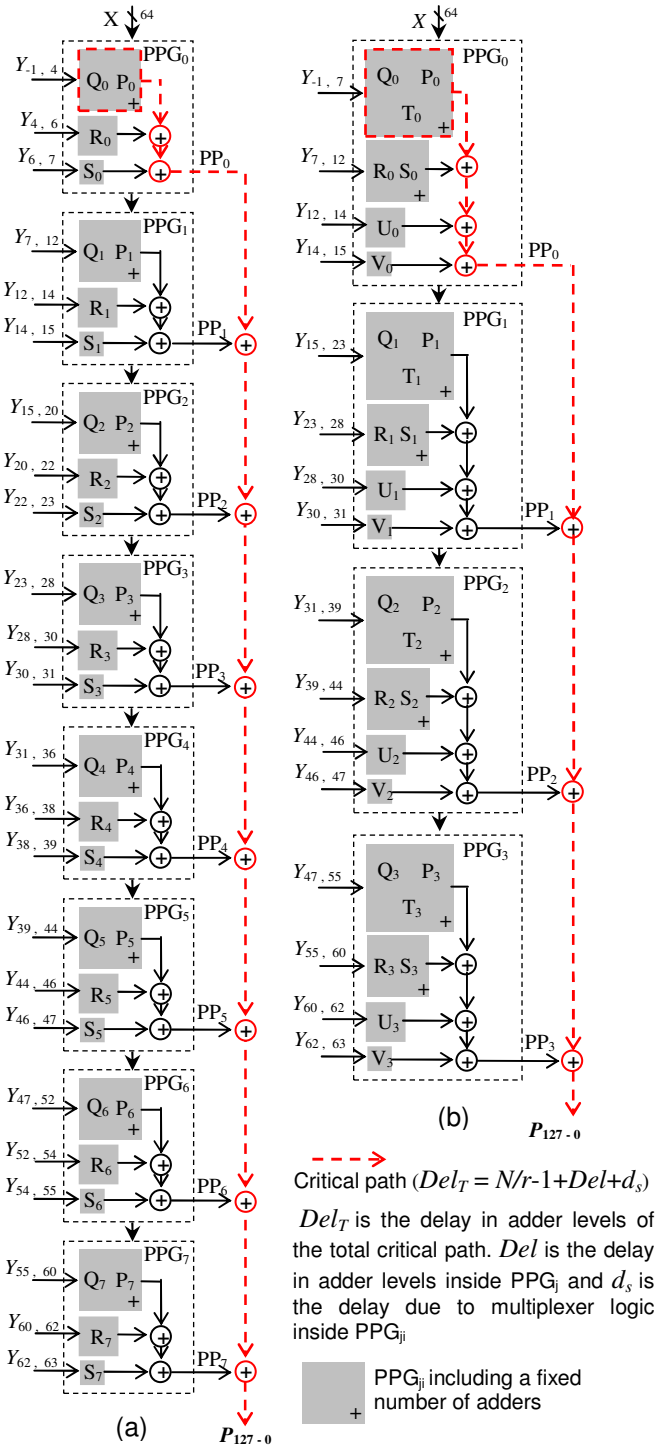


FIGURE 5.4 – Two's complement 64×64 bit multiplier.
(a) Radix-2^8 multiplier. Space partitioning according to Eq. 5.18
(b) Radix-2^{16} multiplier. Space partitioning according to Eq. 5.19

- the number (N/r) of PPG_j;
- the number (i) of lower sub-radices (2^1, 2^2, 2^5, and 2^8) used to build up the higher radix-2^r. To each sub-radix-2^s used (PPG_{ji}) corresponds an RTL "case statement" that recodes the digit terms (Q_{ji},P_{ji},T_{ji},…) present in the equation;

- the number of entries (e_s+1) in each "case statement" corresponding to each sub-radix-2^s;
- the number (d_s) of digit terms ($Q_{ji}, P_{ji}, T_{ji}, \ldots$) that figures in each "case statement";
- and on the number of necessary odd-multiples ($|O_s|$) used to calculate the digit terms.

Hence, we can announce that: $Mux1 = \frac{N}{r}.\sum_i \left(2^{e_s+1}.d_s.|O_s|\right)$. For Dimitrov algorithm (Eq. 5.20), this gives: r=8, i=1, e_s=8, d_s=2, and $|O_s|$=4. Thus, $Mux1$ = 512 N.

The synthesis of the RTL "shift statement" infers multiplexers whose complexity depends on the number (p_{sj}) of different shift positions for all odd-multiples involved in the calculation of each digit term (j). Thus, we can write: $Mux2 = \frac{N}{r}.\sum_i \sum_j \left(p_{sj}.|O_{sj}|\right)$. For Dimitrov algorithm (Eq. 5.20), this gives: r = 8, i=1, j=2, p_{s1} =p_{s2} =8, and $|O_{s1}|$ = $|O_{s2}|$ = 4. Thus, $Mux2$=8N. Hence, the total multiplexer complexity becomes: Mux_T = $Mux1$+$Mux2$=520N.

An N-bit radix-2^r multiplier generates N/r PP. Thus, the total number of adders comprises:

- $(N/r)-1$ adders to sum the N/r PP;
- plus the necessary adders inside each PPG_j to accumulate the intermediate PP issuing from PPG_{ji};
- plus a number of adders included inside each PPG_{ji} depending on the recoding scheme used.

For instance, in Seidel algorithm (Eq. 5.17), the term $11^2 Q_{ji} + 11 P_{ji} + T_{ji}$ is calculated as follows: $\left(2^7 Q_{ji} - 2^3 Q_{ji} + Q_{ji}\right) + \left(2^3 P_{ji} + 2^2 P_{ji} - P_{ji}\right) + T_{ji}$, which requires 6 adders for post-accumulation operation [15][16]. Hence, the total number of necessary adders is: $Add_T = (N/8) - 1 + 6(N/8) = (7/8)N - 1$.

Delay

The total delay (Del_T) along the critical path is the summation of PPG_j delay and reduction tree delay. Based on the total number of adders (Add_T), the critical path of the multiplier in terms of logic levels is: Del_T= N/r–1+Del+d_s, where Del is the delay due to adder stages inside PPG_j and d_s is the delay due to multiplexer logic inside PPG_{ji}. This latter depends on Mux factor of used PPG_{ji} (2^1, 2^2, 2^5, or 2^8). Therefore, $d_1 < d_2 < d_5 < d_8$. Note that d_s is fixed and Del depends on r and s values. For instance, according to Eq. 5.17, Seidel algorithm exhibits a critical path of: Del_T= $N/8$–1+6+d_8=$N/8$+5+d_8. Table 5.3 provides the area occupation and delay for each recoding algorithm.

Physical implementation

All recoding schemes mentioned in Table 5.3 underwent several verification steps. First all equations were validated with a random C-program. Then, they were implemented at RTL level in Verilog-2001 (IEEE 1364) as technology-independent reusable IP-cores [3], using exactly the same

optimized coding style for an equitable comparison. They are compile-time reconfigurable according to N and r. Reader is referred to [15], [16], and [5] for recoding tables used in Eq. 5.16, 5.17, and 5.20, respectively.

TABLE 5.3 – Main feature comparison.

Features	New Recoding Algorithms		McSorley [18]	Seidel [15][16]	Dimitrov [5]
	Eq. 5.18	Eq. 5.19	Eq. 5.15	Eq. 5.17	Eq. 5.20
Radix	2^8	2^{16}	2^2	2^8	2^8
Del_T	$\frac{N}{8}+3+d_5$	$\frac{N}{16}+8+d_8$	$\frac{N}{2}-1+d_2$	$\frac{N}{8}+5+d_8$	$\frac{N}{8}+d_8'$
Mux_T	$19N$	$106N$	$5N$	$194N$	$520N$
Add_T	$\frac{5N}{8}-1$	$\frac{6N}{8}-1$	$\frac{N}{2}-1$	$\frac{7N}{8}-1$	$\frac{N}{4}-1$

N is the operand size and 2^r is the radix used. Del_T is the total delay in terms of adder levels in the critical path of a linear reduction tree. d_s is the delay due to multiplexer logic inside PPG_{ji}. d_s depends on Mux factor ($d_1<d_2<d_5<d_8<d'_8$). $Mux_T=(N/r)Mux$, where Mux is an estimation of the multiplexer logic required by PPG_j. Add_T is the total number of adders required in the whole multiplier. Del_T is the delay in adder levels of the total critical path. Del is the delay in adder levels inside PPG_j and d_s is the delay due to multiplexer logic inside PPG_{ji}.

All RTL codes went through a severe cycle-accurate functional verification procedure using Modelsim SE-6.3f logic simulator. They were first challenged against a set of special and severe test cases, and then submitted to a random test for a very large number of vectors. After a successful functional verification, physical tests were performed. They were integrated into an FPGA evaluation board for an ultimate validation. Afterwards, all equations were synthesized and mapped to the same Virtex-6 FPGA circuit (xc6vsx475t-2ff1156) using Xilinx ISE 13.2 release version [21]. We used for comparison a two's complement 64×64 bit parallel multiplier. The implementation results are grouped in Table 5.4

TABLE 5.4 – Implementation results of a two's complement 64-bit parallel multiplier on Xilinx xc6vsx475t-2ff1156 circuit.

Results	New Recoding Algorithms		McSorley [18]	Seidel [15][16]	Dimitrov[5]
	Eq. 5.18	Eq. 5.19	Eq. 5.15	Eq. 5.17	Eq. 5.20
Area[1]	3219	4659	2103	5251	6599
Energy[2]	1.63	2.11	1.46	2.49	2.48
Speed[3]	52.4	49.34	30.04	48.62	43.17

Synthesis tool was forced to map RTL code to distributed slices of FPGA and avoid mapping to builtin 18x18 bit hardwired multipliers (DSP slices).
1: Area occupation in number of Virtex-6 slices. 2: Energy consumption per multiplication operation (pJ). 3: Million multiplications per second (MMPS).

Although Dimitrov recoding exhibits the shortest critical path in adder stages (N/8), the impact of multiplexer logic (d'_8) on the total performance is important (Table 5.4). Besides, it is the most area

consumer despite the fact that it employs the lowest number of adders (N/4-1). Adversely, Seidel algorithm is the most adder consumer (7N/8-1). To determine which factor, Mux_T or Add_T, exerts more influence on area occupation, let us compare their respective ratios for Seidel and Dimitrov algorithms: Mux_T(Eq. 5.20)/Mux_T(Eq. 5.17)=2.7 and Add_T(Eq. 5.17)/Add_T(Eq. 5.20)=3.5.

Significant conclusion: the area occupation is dominated by Mux_T factor, and becomes larger as Mux_T number becomes higher (Table 5.3 and 5.4). This correlation is advantageously used to minimize area occupation as will be shown in the next section.

McSorley algorithm (Eq. 5.15) is the least area consumer and the slowest recoding scheme for any value of N. The best area/speed compromise for N=64 is given by our recoding scheme based on Eq. 5.18. However, this latter will be outperformed by Eq. 5.19 for larger values of N (N>64) since a higher radix (2^{16}) is employed.

While energy consumption is function of the switched capacitance, Table 5.4 shows a direct correlation between area occupation and energy consumption. Making Mux_T indicator lower, will result in a less energy-consumer recoding algorithm.

Finally, based on theory and implementation results, we conclude that the best tradeoff related to our recoding schemes depends on N and r values. For larger N values (N>64), larger radices are necessary to reduce the critical path. But for larger radices (r>16) we need to duplicate some of the elementary PPG_{ji} (2^1,2^2,2^5,2^8) to build up the radix-2^r PPG_j. Therefore, at this level a relevant question arises: given N, what is the value of r and its corresponding elementary PPG_{ji} configuration (optimal partitioning of PPG_j) that leads to the shortest critical path (Del_{Tmin}) with minimum hardware resources (Mux_{Tmin})? The answer to this question is given in the next sections.

5.4.2 Preliminary Study to an Optimal Partitionning

We extend the recoding-space of our Eq. 5.18 and 5.19 to the general case as follows: each r+1 bit slice is recoded using a, b, c, d instances of radix 2^8, 2^5, 2^2, 2^1 algorithms, respectively, such that $8a+5b+2c+d=r$. To this recoding scheme corresponds the following equation:

$$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{a-1} A_i^j .2^{8i} + \sum_{i=0}^{b-1} B_i^j .2^{5i+8a} + \sum_{i=0}^{c-1} C_i^j .2^{2i+8a+5b} + \sum_{i=0}^{d-1} D_i^j .2^{i+8a+5b+2c} \right] .2^{rj}, \qquad (5.21)$$

where $A_i^j = 11^2 . Q_i^j + 11. P_i^j + T_i^j$ with $Q_i^j \in \{-2,-1,0,1,2\}$ and

$P_i^j , T_i^j \in \{-16,-8,-4,-2,-1,0,1,2,4,8,16\}$; $B_i^j = 7.R_i^j + S_i^j$ with $R_i^j \in \{-2,-1,0,1,2\}$ and

$S_i^j \in \{-4,-2,-1,0,1,2,4\}$; $C_i^j = y_{2i-1}^j + y_{2i}^j - 2y_{2i+1}^j$ with $C_i^j \in \{-2,-1,0,1,2\}$; finally

$D_i^j = y_{i-1}^j - y_i^j$ with $D_i^j \in \{-1, 0, 1\}$.

The translation of Eq. 5.21 into architecture is depicted in Fig. 5.1b (top view only), where each PPG_j is built up using a mixture of four different PPG_{ji} depending on the quadruplet (a,b,c,d) as illustrated by Fig. 5.5. For instance, Eq. 5.18 and 5.19 correspond (0,1,1,1) and (1,1,1,1), respectively.

Note that because of the general nature of Eq. 5.21, the d_s term of Del_T is equal to max(d_8,d_5,d_2,d_1) of used PPG_{ji}.

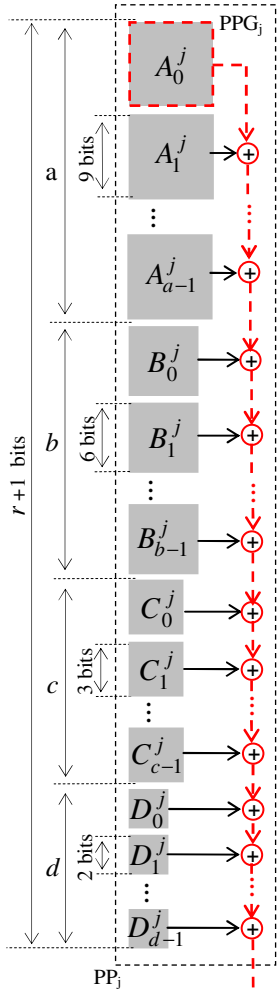


FIGURE 5.5 – Critical path ($Del+d_i$) inside a generalized PPG_j.

Given *N* and *r*, to determine the optimal partitioning of the whole multiplier (global optimum since PPG_j are identical), we need to find first the quadruplet (a,b,c,d) that satisfies the condition $8a+5b+2c+d=r$ and leads to the PPG_j with minimum hardware ressources (Mux_{min}) and the shortest critical path (Del_{min}). As it is not sure that such a solution exists, we are using composite metrics A^iT^j of area (A) and delay (T) for i and j varying from 0 to 5 [22]. A total of 11 metrics (A, A^5T, A^4T, A^3T, A^2T, AT, AT^2, AT^3, AT^4, AT^5, T) are used. The A metric alone delivers the best area solution (Mux_{min}), while T metric provides the best delay solution (Del_{min}). In between (A^iT^j), more-or-less balanced solutions are obtained. The implementation of this solution requires the (Mux, Del) couple (Table 5.5) corresponding to each basic recoding algorithm (2^8,2^5,2^2,2^1).

Because of an explosive number of possible combinations (N>>), the solution space is exhaustively explored using a deterministic C-program for r varying from 8 to 1024. The obtained results are reported in Table 5.6.

TABLE 5.5 – Delay and multiplexer complexity of basic radices: step #1

Algorithm	Del	Mux
2^1	0	5
2^2	0	10
2^5	2	133
2^8	6	1548

Mux values are extracted from the heuristic developed in Section 5.4.1.3
Ex: 1548=194 × 8.

TABLE 5.6 – Optimal PPG_j solution (a,b,c,d) leading to the optimal radix-2^r multiplier according to composite metrics A^iT^j : step #1.

r size (bits)	Criteria	Instance Number				Del	Mux	Del_{min}	Mux_{min}
		a	b	c	d				
8	A – T	0	0	4	0	3	40	3	40
16	A – AT^5	0	0	8	0	7	80	5	80
	T	0	3	0	1	5	404		
32	A – AT^3	0	0	16	0	15	160	8	160
	AT^4 – T	0	6	1	0	8	808		
64	A – AT^2	0	0	32	0	31	320	13	320
	AT^3 – AT^5	0	12	2	0	15	1616		
	T	8	0	0	0	13	12384		
128	A – AT^2	0	0	64	0	63	640	21	640
	AT^3 – AT^5	0	25	1	1	28	3340		
	T	16	0	0	0	21	24768		
256	A – AT	0	0	128	0	127	1280	37	1280
	AT^2 – AT^5	0	51	0	1	53	6788		
	T	32	0	0	0	37	49536		
512	A – AT	0	0	256	0	257	2560	69	2560
	AT^2 – AT^5	0	102	1	0	104	13576		
	T	64	0	0	0	69	99072		
1024	A – AT	0	0	512	0	512	5120	133	5120
	AT^2 – AT^5	0	204	2	0	207	27152		
	T	128	0	0	0	133	198144		

A – T: all the metric span A, A^5T, A^4T, A^3T, A^2T, AT, AT^2, AT^3, AT^4, AT^5, T. To A and T metrics correspond respectively the minimal values Mux_{min} and Del_{min} that serve as reference for the optimization process.

In conclusion, optimal area solutions (Mux=Mux_{min}) are exclusively based on radix-2^2 algorithm (0,0,c,0), but they are excessively slow (Del>>Del_{min}). While optimal speed solutions (Del=Del_{min}) are entirely composed of radix-2^8 algorithm (a,0,0,0), but they are exaggeratedly large (Mux>>Mux_{min}). Finally, balanced area/speed solutions are mainly based on radix-2^5 algorithm with at most one or two instances of radices 2^1 and 2^2 algorithms (0,b,c,d). However, even the "balanced" solution is not really balanced enough since the mean values of Del and Mux are 1.4×Del_{min} and 5.2×Mux_{min} , respectively. The reason is due to the large disparity between Mux values of the basic radices (Table 5.5).

To correct this disequilibrium, we replace respectively the two Seidel radix-2^8 and 2^5 expressions (A_i^j and B_i^j) included in Eq. 5.21 by their mathematically equivalent counterparts as follows: $A_i^j = \sum_{k=0}^{3} 2^{2k} C_k^{ji}$ and $B_i^j = D_0^{ji} + 2C_0^{ji} + 2^3 C_1^{ji}$. These new expressions are radix-2^8 and 2^5, respectively. They produce respectively the same intermediary partial products at PPG_{ji} output as their Seidel counterparts. In fact A_i^j is formed by a succession of four instances of McSorley algorithm, while B_i^j is composed of one instance of Booth algorithm followed by two instances of McSorley algorithm. Del and Mux values of the new basic radices are grouped in Table 5.7.

Results delivered by the deterministic C-program are reported in Table 5.8. All solutions are optimal since $Del=Del_{min}$ and $Mux=Mux_{min}$. They are all based on radix-2^8 algorithm (a,0,0,0). In case r is not a multiple of 8, optimal solutions are also obtained, composed mainly of radix-2^8 algorithm with at most one instance of radix-2^1, 2^2 or 2^5 algorithms, depending on the remainder of r by 8 division.

TABLE 5.7 – Delay and multiplexer complexity of the new basic radices: step #2.

Algorithm	Del	Mux
2^1	0	5
2^2	0	10
2^5	2	25
2^8	3	40

TABLE 5.8 – Optimal PPG_j solution (a,b,c,d) leading to the optimal radix-2^r multiplier according to composite metrics A^iT^j : step #2.

r size (bits)	Instance Number				Del	Mux	Del_{min}	Mux_{min}
	a	b	c	d				
8	1	0	0	0	3	40	3	40
16	2	0	0	0	4	80	4	80
32	4	0	0	0	6	160	6	160
64	8	0	0	0	10	320	10	320
128	16	0	0	0	18	640	18	640
256	32	0	0	0	34	1280	34	1280
512	64	0	0	0	66	2560	66	2560
1024	128	0	0	0	130	5120	130	5120

The new results are so interesting that we are encouraged to pursue further the optimization process using higher basic sub-radices (s>8) to reduce the total delay (Del_T) of the multiplier. Let us this time replace A_i^j and B_i^j as follows: $A_i^j = \sum_{k=0}^{7} 2^{2k} C_k^{ji}$ and $B_i^j = \sum_{k=0}^{3} 2^{2k} C_k^{ji}$. We eliminate radix-$2^5$ since it can be derived from radix-2^1 and 2^2. The new Del and Mux values of basic radices are grouped in Table 5.9. The C-program shows up even more interesting results since starting from r≥64 (Table 5.10), lower delays are obtained with the same multiplexer complexities as the ones reported in Table 5.8. Based on the obtained results, we pushed farther the optimization process using even higher basic sub-radices (s=16..32).

TABLE 5.9 – Delay and multiplexer complexity of the new basic radices: step #3.

Algorithm	Del	Mux
2^1	0	5
2^2	0	10
2^5	3	40
2^8	7	80

TABLE 5.10 – Optimal PPG_j solution (a,b,c,d) leading to the optimal radix-2^r multiplier according to composite metrics A^iT^j : step #3.

r size (bits)	Instance Number				Del	Mux	Del_{min}	Mux_{min}
	a	b	c	d				
8	0	1	0	0	3	40	3	40
16	0	2	0	0	4	80	4	80
32	0	4	0	0	6	160	6	160
64	4	0	0	0	10	320	10	320
128	8	0	0	0	14	640	14	640
256	16	0	0	0	22	1280	22	1280
512	32	0	0	0	38	2560	38	2560
1024	64	0	0	0	70	5120	70	5120

→ : Optimal solution moved from $(0,b,0,0)$ to $(a,0,0,0)$

All optimal solutions come either on the form $(a,0,0,0)$ or $(0,b,0,0)$. At this level we can draw a significant conclusion: since the optimal solution is always in the form $(a,0,0,0)$ or $(0,b,0,0)$ with a=2k and b=2k', there exists an integer s=2k'' such as either $(s,0,0,0)$ or $(0,s,0,0)$ is the optimal solution.

Consequently, Eq. 5.21 is rewritten accordingly, as follows:

$$Y=\sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{s}-1}\left[\sum_{k=0}^{\frac{s}{2}-1}C_k^{ji}.2^{2k}\right]2^{si}\right]2^{rj}, \qquad (5.22)$$

with $C_k^{ji}=y_{2k-1}^{ji}+y_{2k}^{ji}-2y_{2k+1}^{ji}$ and $C_k^{ji}\in\{-2,-1,0,1,2\}$.

Based on heuristic developed in Section 5.4.1.3, multiplexer complexity of Eq. 5.22 for the whole multiplier is always equal to Mux_T=10×N/2=5N for any value of r and s. As for the multiplier delay (Del_T), we need to determine the couple (r,s) that leads to the shortest critical path in terms of adder levels. This is what is achieved in the next section.

5.4.3 The Optimal Space/Time Partitioning

The total delay (Del_T) of the whole multiplier related to Eq. 5.22 is: Del_T= N/r–1+Del+d_2 where Del is the PPG_j delay equal to (r/s–1)+(s/2–1), and d_2 is the multiplexer delay corresponding to the recoding logic of radix-2^2. Thus, Del_T= N/r+r/s+s/2–3+d_2.

The optimal delay with regard to r is obtained for (r,s) couples satisfying $\partial(Del_T)/\partial r = 0$, which gives $r = \sqrt{s.N}$. When r is substituted by $\sqrt{s.N}$ into Del_T expression, we obtain: $Del_T = 2\sqrt{N/s} + s/2 - 3 + d_2$. Likewise, the optimal delay with regard to s is obtained for s value satisfying $\partial(Del_T)/\partial s = 0$. We obtain $s = 2\sqrt[3]{N/2}$. Hence, the optimal delay becomes:

$$Del_T = 3\sqrt[3]{N/2} - 3 + d_2. \quad (5.23)$$

Finally, we conclude that the optimal N-bit multiplier, in comparison to Eq. 5.15 [18], relies on the new triple recursive Eq. 5.22 with

$$(r,s) = (\sqrt[3]{2.N^2},\ 2\sqrt[3]{N/2}). \quad (5.24)$$

Table 5.11 provides the s and r values that lead to the optimal partitioning with respect to the operand size N. The values s and r correspond to the number of multiplier bits that are treated simultaneously inside each PPG_{ji} and each PPG_j, respectively. For N=64, the optimal partitioning is obtained with (r,s)=(32,8) as illustrated by Fig. 5.6. Whereas Eq. 5.22 and 5.15 require the same amount of hardware resources (Mux_T, Add_T)=(320,31), they exhibit different critical paths: 7 and 31 in terms of adder levels, respectively.

TABLE 5.11 – The optimal partitioning versus operand size N.

N (bits)	New recoding Eq. 5.22			McSorley [18] Eq. 5.15	Seidel [15][16] Eq. 5.17	Dimitrov [5] Eq. 5.20
	s	r	Del_T			
8	4	8	2	3	6	1
16	4	8	3	7	7	2
32	8	16	5	15	9	4
64	8	32	7	31	13	8
128	8	32	9	63	21	16
256	16	64	13	127	37	32
512	16	128	17	255	69	64
1024	16	128	21	511	133	128
2048	32	256	28	1023	261	256
4096	32	512	35	2047	517	512
8192	32	512	45	4095	1029	1024

s value corresponds to the number of bits that are treated simultaneously inside each PPG_{ji}, while r value indicates the number of bits that are processed simultaneously inside each PPG_j. d_s is not included in Del_T since $d_2 < d_8 < d'_8$.

5.4.4 Discussion of the Implementation Results

We proved via FPGA implementation (Table 5.4) how much accurate are the area heuristics developed in Section 5.4.1.3 (Table 5.3). Based on this, we have undertaken a gradual theoretical optimization process that yielded to Eq. 5.22. This latter is implemented on FPGA with N=64, and the results in terms of multiply-time, energy consumption per multiply-operation, and total gate count, are as follows: 78.98 MMPS, 1.45pJ and 1987 slices, respectively.

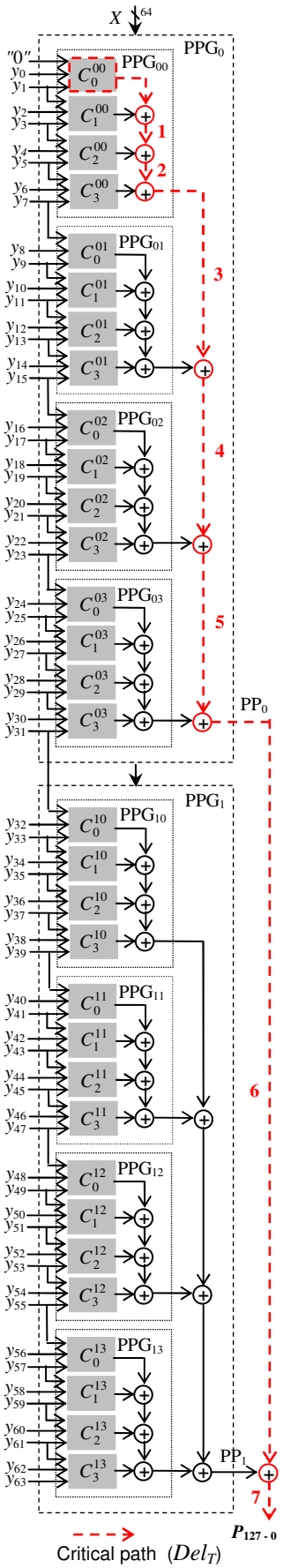


FIGURE 5.6 – Optimal partitioning of a two's complement 64×64 bit radix-2^{32} parallel multiplier based on Eq. 5.22 with (r,s)=(32,8).

Compared to implementation results of Seidel and Dimitrov algorithms (Table 5.4), gain ratios of 1.62, 1.71, 2.64 and 1.83, 1.71, 3.32 are obtained, respectively. A 64-bit multiplier generated by Xilinx Coregen exhibits 75.86 MMPS and consumes twelve 18×18 bit DSP-slice multipliers.

The real reasons behind these important results are cleared up as follows.

5.4.4.1 Area Occupation

For operand size N=64, Eq. 5.22 is a composite radix-2^{32} algorithm (Table 5.11), where each PPG_j processes simultaneously 32+1 inputs that are split on four sub-radix- 2^8 PPG_{ji} made of four instances (C_k^{ji}) of McSorley algorithm (Fig. 5.6). Seidel and Dimitrov algorithms are rather radix-2^8 algorithms, based on mono-bloc PPG_j.

In fact, although radix-2^8 PPG_{ji} of Eq. 5.22 and radix-2^8 PPG_j of Seidel and Dimitrov are based on different recoding schemes, they are mathematically equivalent since they produce the same partial product PP_{ji}/PP_j. Based on theory (Table 5.3) and implementation results (Table 5.4), Dimitrov recoding is the most space consuming due to the use of odd-multiples of the multiplicand. On the other hand, Seidel recoding does not require odd-multiples, but since 9 inputs are treated simultaneously in a mono-bloc PPG_j; a large amount of multiplexer resources is needed to recode the 2^9=512 input combinations. Finally, radix-2^8 PPG_{ji} of Eq. 5.22 is the least area consumer because it does not employ odd-multiples and requires a small amount of multiplexers as the total number of input combinations in each radix-2^8 PPG_{ji} is equal to 8+8+8+8=32. Note that the three recoding schemes are incorporating a number of adders in their PPG_{ji}/PPG_j which is 3, 6, and 1 for Eq. 5.22, Seidel and Dimitrov algorithms, respectively.

Significant conclusion: the area occupation is dominated by the Mux factor, and becomes larger as Mux value becomes higher.

5.4.4.2 Delay

Using higher radices (r>>) will certainly shortens the critical path. However, for high r values, mono-bloc PPG_j recoding induces an important delay (d_s) due to the high density of multiplexer logic that significantly degrades the whole performance of the multiplier. This is clearly illustrated by Dimitrov radix-2^8 recoding whose critical-path totalizes 8 adder levels but exhibits a lower multiply rate (43.17 MMPS) compared to Seidel recoding that have a critical-path composed of 13 adder levels but shows a more interesting rate (48.62 MMPS) due to lower multiplexer complexity (Table 5.3 and 5.4). As for Eq. 5.22, since a composite PPG_j is used, d_s is equal to d_2 (C_k^{ji} delay) which is the smallest delay ($d_2 < d_5 < d_8$). Besides, the critical path goes through the smallest number (7) of adder stages, exploiting maximum parallelism that can be provided by the triple-recursive Eq. 5.22. Thus, it is not surprising that Eq. 5.22 achieves the best performance (78.98 MHz), even when compared to Xilinx Coregen multiplier based on DSP-slices (75.86 MHz). A double-recursive (s=2) version of Eq. 5.22 served to design a scalable 16-bit setpoint Finite-Word-Length PID controller, employing five multiplication cores. The implementation results outperformed the published ones at all levels as will be shown in the next Chapter.

Significant conclusion: using composite recoding in conjunction with an optimal partitioning (r and s values) provides the shortest critical path.

Eq. 5.22 shows high aptitude for pipelining. Two finely and coarsely grained systolic architectures for 64-bit multiplier are depicted in Fig. 5.7a and Fig. 5.7b, respectively. Fig. 5.7a architecture is more suitable for high throughput applications, with 7 clock-cycle latency.

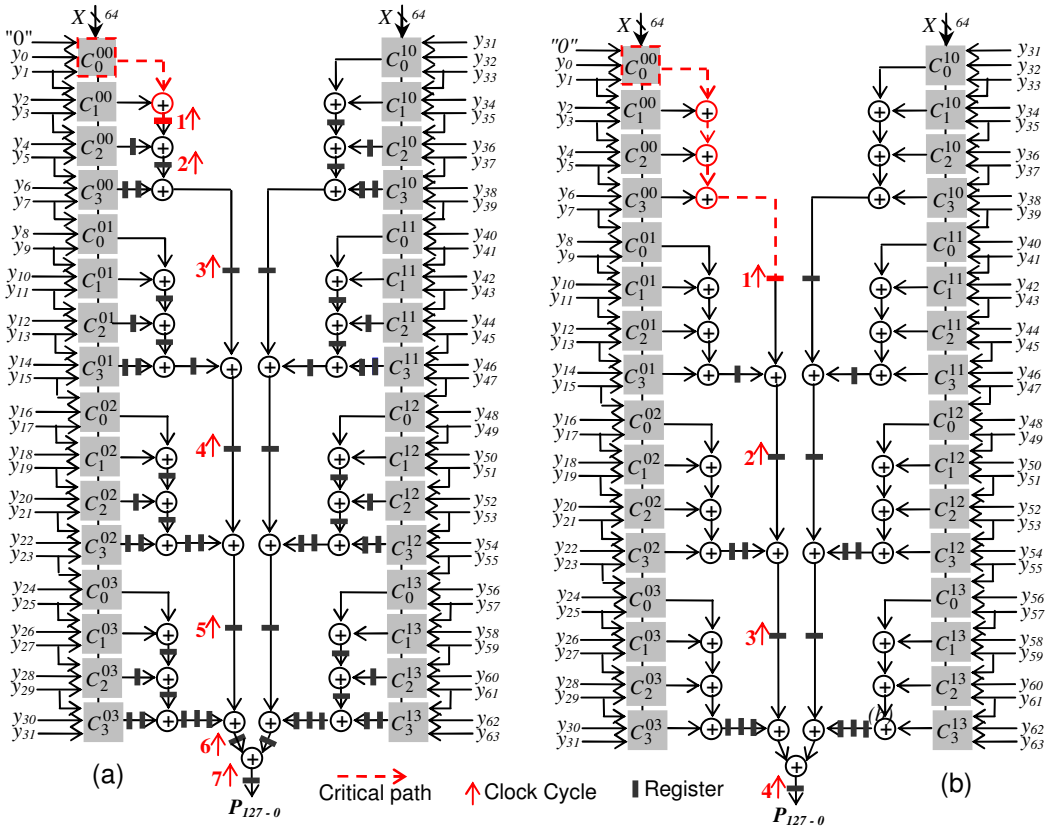


FIGURE 5.7 – Space/Time partitioning of a two's complement 64×64 bit radix-2^{32} parallel multiplier based on Eq. 5.22. (a) High-throughput finely-grained systolic architecture; (b) Low-latency coarsely-grained systolic architecture.

5.5 New Radix-2^r Multip-recision Multiplication Algorithms

Prior to develop a highly-scalable radix-2^r multi-precision multiplier, the need for a flexible and low-power sign-extension technique is mandatory.

5.5.1 New Radix-2^r Sign Extension Technique

Though many low-power sign extension techniques exist in the literature, they are not adapted to reconfigurability. The reason for this shortcoming is that the correction bits must be calculated for each value of operand-size N [23][24]. Besides, to our knowledge, no sign-extension solution exists for radix based multiplication (r). In what follows, we propose a generic low-power solution that circumvents these two obstacles. It is illustrated by Fig. 5.8 for N=8 and r=2, but can be systematically extended to any N and r values.

Intuitively, we are not simultaneously performing the sum of the partial products, but each partial product of current step j is added to the sum of the preceding ones (from 0 to j-1). The rationale for

the number of sign-bits to the left can be done locally, step by step, row by row. In other words, we have to take advantage of the fact that the partial sum already contains the sum of the sign bits of previous partial products. We must simply ensure that the sum output of the sign bit of current step j is added to the two most-significant bits of the next step (j+1). To generalize to radix-2^r multiplication, the sign-bit (N^{th} position bit) of each partial product is extended with r bits to the left (r-1 for a maximum shift, plus one bit for the sign), and the sum output of the sign bit of step j is added to the r most-significant bits of the next step (j+1).

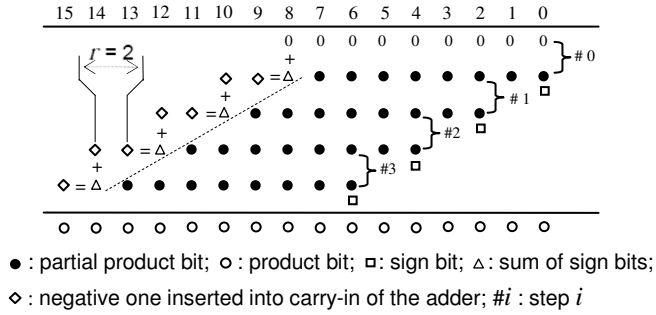


FIGURE 5.8 – Low-power sign-extension technique for the particular case (N, r)=(8, 2).

5.5.2 New Radix-2^r Multi-Precision Multiplication Techniques

In traditional N×N bit multi-precision multipliers, there is possibility to perform either a single N×N double precision, or a single N/2×N/2 simple precision, or a twin parallel N/2×N/2 simple precision multiplication. This is made possible by partitioning the two operands X and Y into respectively most and less significant sub-operands (X_H,Y_H) , and (X_L,Y_L). A number of solutions exist and are summarized in [23][25]. Unfortunately, they are either restricted to unsigned multiplication, or they do not take power consumption into consideration, or they are not flexible enough.

We propose hereafter a new technique that not only overcomes all above-mentioned shortcomings, but also allows a customized partitioning of the operands into any number of slices as well as in any slice sizes. Besides, this new technique is well adapted to radix based multiplication. Its features are compared to the technique presented in [23] (Fig 5.9).

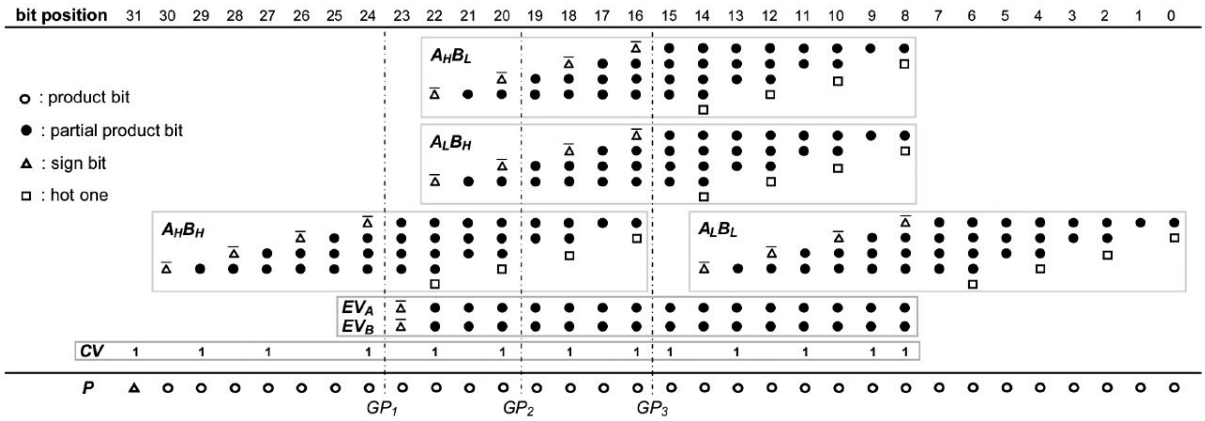


FIGURE 5.9 – Multiplication matrix of partial-products bits for 16-b multiplication with one level recursion [23].

Let us take Eq. 5.8 and apply it to X and Y for r=N/2, we obtain:

$$X = \sum_{j=0}^{1} Q_j 2^{\frac{N}{2}j} = Q_1 2^{\frac{N}{2}} + Q_0 = X_H + X_L . \quad (5.25)$$

Hence,

$$X.Y = Q_1 P_1 2^N + Q_1 P_0 2^{\frac{N}{2}} + Q_0 P_1 2^{\frac{N}{2}} + Q_0 P_0$$

$$= X_H.Y_H + X_H.Y_L + X_L.Y_H + X_L.Y_L . \quad (5.26)$$

Note that Q_1 and Q_0 are (N/2)+1 bit size, but x_{-1} can be omitted from Q_0 since it is stuck at zero. Thus, we obtain four independent signed multipliers: $X_H.Y_H$, $X_H.Y_L$, $X_L.Y_H$, $X_L.Y_L$ which are respectively (N/2)+1×(N/2), (N/2)+1×N/2, N/2×(N/2), N/2×N/2 bit size. Fig. 5.10 illustrates the implementation of Eq. 5.26 for a signed 16x16 bit multiplier based on recoding Eq. 5.15 with r=2.

Eq. 5.26 eliminates the cumbersome term (EV×$2^{N/2}$) in equation (6) of [23] as well as the necessary logic for its generation. More importantly, in Fig. 5.10, four 8x8 bit multiplications can be performed simultaneously, whereas in [23] only two are allowed because of the shared terms (EV×$2^{N/2}$) and CV required for the sign extension. Without counting the necessary EV generation logic and the use of inverters for the negation of the sign bits, the partitioning proposed in [23] consumes a total bit count of 205 for a 16x16 bit multiplier, while ours requires 198 bits.

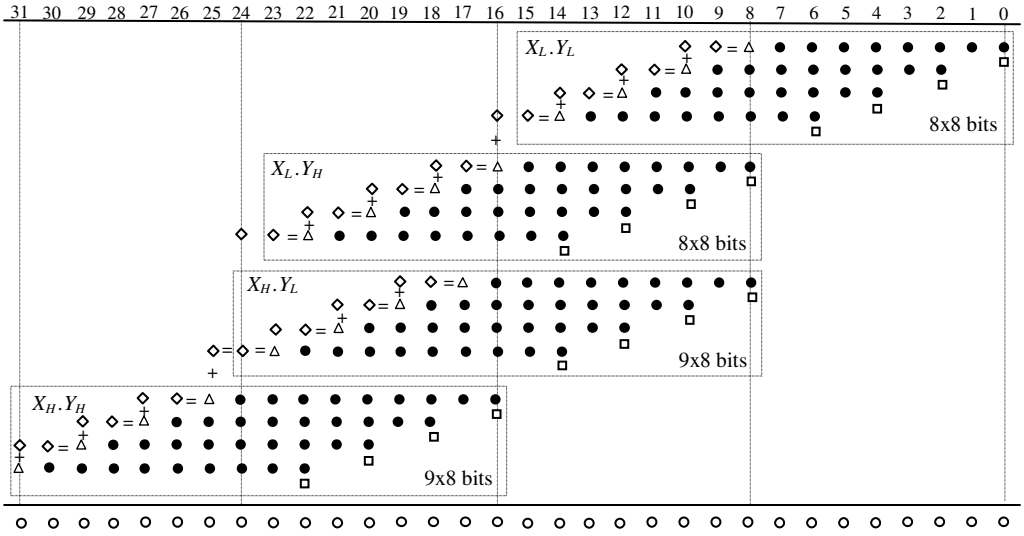


FIGURE 5.10 – Low-power multi-precision multiplier for the particular case (N,r)=(16,2) with 8-bit sub-operand size.

Note that Eq. 5.12 can be advantageously used to partition X_H and X_L with different bit lengths. For instance, with r=N, s=3N/4 and t=N/4, we obtain:

$$X = P + T\,2^{\frac{3N}{4}} \quad (5.27)$$

Hence,

$$X.Y = P\,P' + P\,T'\,2^{\frac{3N}{4}} + T\,P'\,2^{\frac{3N}{4}} + T\,T'\,2^{\frac{3N}{2}}$$

$$= X_L.Y_L + X_L.Y_H + X_H.Y_L + X_H.Y_H \quad (5.28)$$

Four independent signed multipliers are generated: $X_H.Y_H$, $X_H.Y_L$, $X_L.Y_H$, $X_L.Y_L$, which are respectively $(N/4)+1\times(N/4)$, $(N/4)+1\times(3N/4)$, $(3N/4)+1\times(N/4)$, and $(3N/4)\times(3N/4)$ bit size. The translation of Eq. 5.28 into architecture is depicted by Fig. 5.11. Both partitioning schemes (Fig. 5.10 and 5.11) needs the same amount of bits (198).

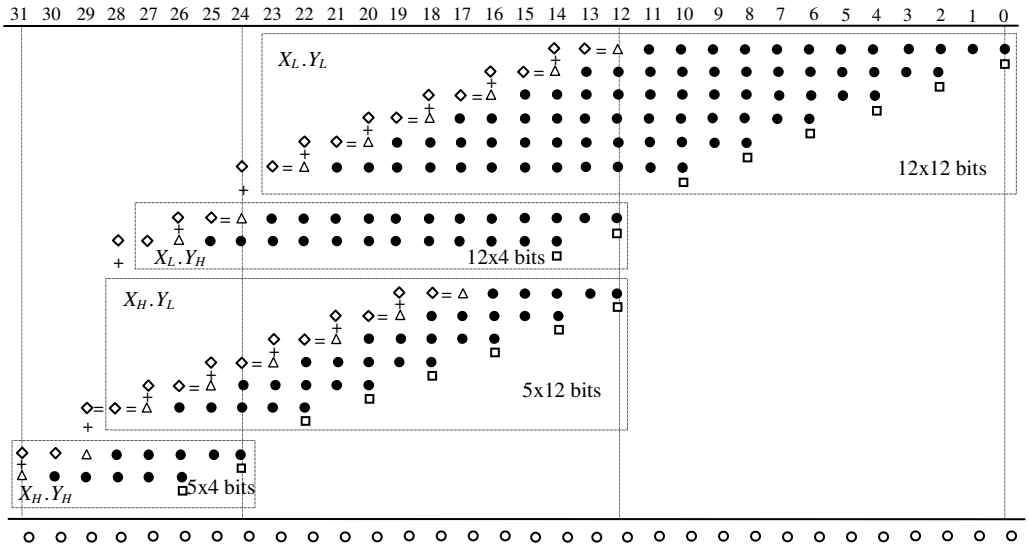


FIGURE 5.11 – Low-power multi-precision multiplier for the particular case (N,r)=(16, 2) with 12 and 4 bit sub-operand sizes.

More efficiently, Eq. 5.28 can be combined with Eq. 5.18 for the recoding of Y_H and Y_L sub-multiplicands to produce a faster partitioning (Fig. 5.12) for operand sizes larger than 16 bits.

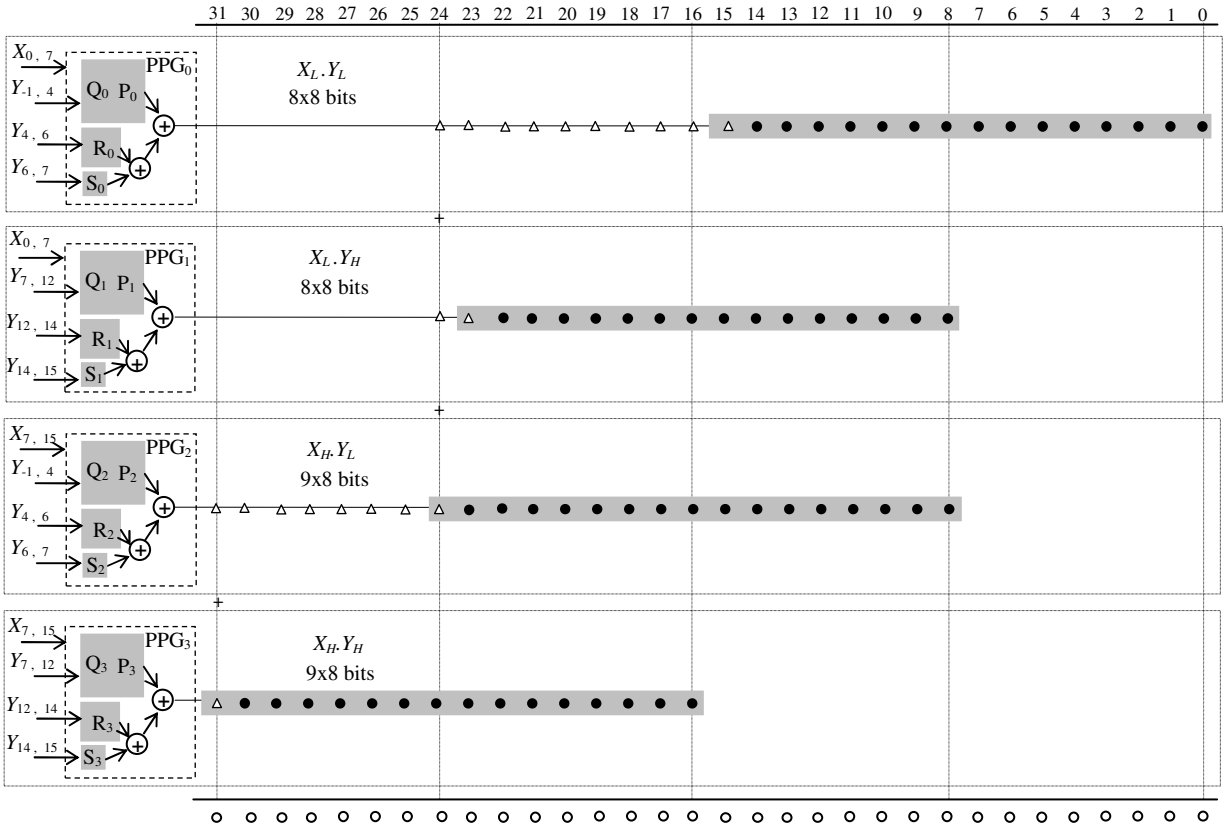


FIGURE 5.12 – Low-power multi-precision multiplier for the particular case (N,r)=(16,8) with 8-bit sub-operand size.

More importantly, Eq. 5.8 can be used to partition the X and Y operands into any desired number of slices depending on r value. Choosing for instance r=N/4 results into the following partitioning:

$$X = \sum_{j=0}^{3} Q_j 2^{\frac{N}{4} j} = Q_3 2^{\frac{3N}{4}} + Q_2 2^{\frac{N}{2}} + Q_1 2^{\frac{N}{4}} + Q_0 . \tag{5.29}$$

Hence,

$$\begin{aligned} X.Y = {} & Q_3P_3 2^{\frac{3N}{2}} + Q_3P_2 2^{\frac{5N}{4}} + Q_3P_1 2^{n} + Q_3P_0 2^{\frac{3N}{4}} \\ & + Q_2P_3 2^{\frac{5N}{4}} + Q_2P_2 2^{N} + Q_2P_1 2^{\frac{3N}{4}} + Q_2P_0 2^{\frac{N}{2}} \\ & + Q_1P_3 2^{N} + Q_1P_2 2^{\frac{3N}{4}} + Q_1P_1 2^{\frac{N}{2}} + Q_1P_0 2^{\frac{N}{4}} \\ & + Q_0P_3 2^{\frac{3N}{4}} + Q_0P_2 2^{\frac{N}{2}} + Q_0P_1 2^{\frac{N}{4}} + Q_0P_0 \\ = {} & X_{HH}.Y_{HH} + X_{HH}.Y_{HL} + X_{HH}.Y_{LH} + X_{HH}.Y_{LL} \\ & + X_{HL}.Y_{HH} + X_{HL}.Y_{HL} + X_{HL}.Y_{LH} + X_{HL}.Y_{LL} \\ & + X_{LH}.Y_{HH} + X_{LH}.Y_{HL} + X_{LH}.Y_{LH} + X_{LH}.Y_{LL} \\ & + X_{LL}.Y_{HH} + X_{LL}.Y_{HL} + X_{LL}.Y_{LH} + X_{LL}.Y_{LL} \end{aligned} \tag{5.30}$$

Eq. 5.30 generates sixteen independent signed multipliers. All are (N/4)+1×(N/4) bit size, except $X_{LL}.Y_{HH}$, $X_{LL}.Y_{HL}$, $X_{LL}.Y_{LH}$, $X_{LL}.Y_{LL}$ which are (N/4)×(N/4) bit size. The implementation details of Eq. 5.30 for N=16 based on Eq. 5.15 with r=2 are described in Fig. 5.13. Eq. 5.30 requires a total bit count of 254 which induces an overhead of 28% compared to Eq. 5.26.

Finally, Eq. 5.8 and Eq. 5.12 can be combined with any proposed radix-2^r recoding algorithm to produce any desired multi-precision multiplication scheme.

5.6 Conclusion

From the basis of the new multibit recoding multiplication algorithm, we have developed optimal multipliers with shortest critical paths and minimum hardware resources for any value of the operand size N. We have demonstrated by theory and FPGA implementation the superiority of our high-radix algorithms over their existing counterparts. Because exploiting the maximum parallelism inherent in the multiply operation, our look-up-table based multiplier (Eq. 5.22) is even speed-competitive with Xilinx's hardwired multiplier employing DSP-Slices (18×18 bit full-custom multipliers).

More importantly, we have also demonstrated that the current trend relying upon minimal number-bases for the development of high radix-2^r recoding (r≥8) with mono-bloc PPG requires an excessive amount of multiplexer resources, which offsets speed and power benefits of the compressor factor N/r. On the other hand, we have proved that composite PPG based on the new recursive multibit recoding algorithm is the best realistic alternative.

The topology of our proposed recoding schemes shows high capabilities for pipelining which can be finely or coarsely grained to satisfy both high throughput and low latency applications. A radix-2^{32} 64-bit parallel multiplier has been finely pipelined, resulting in a systolic architecture with seven clock-cycle latency.

While the theoretical concept has been validated using FPGA as a preliminary step, an ASIC implementation based on a standard-cell library is necessary for an ultimate validation of the whole optimization work.

As for the multi-precision solution, this latter would not have been possible without the development of a flexible sign-extension technique. Based on the new recursive algorithm, we have proposed a generic partitioning scheme that can be adapted to any size combination of the operands in order to reduce the power consumption while increasing the computational throughput. This new solution will be deeply explored for further optimizations using the proposed radix 2^r algorithms.

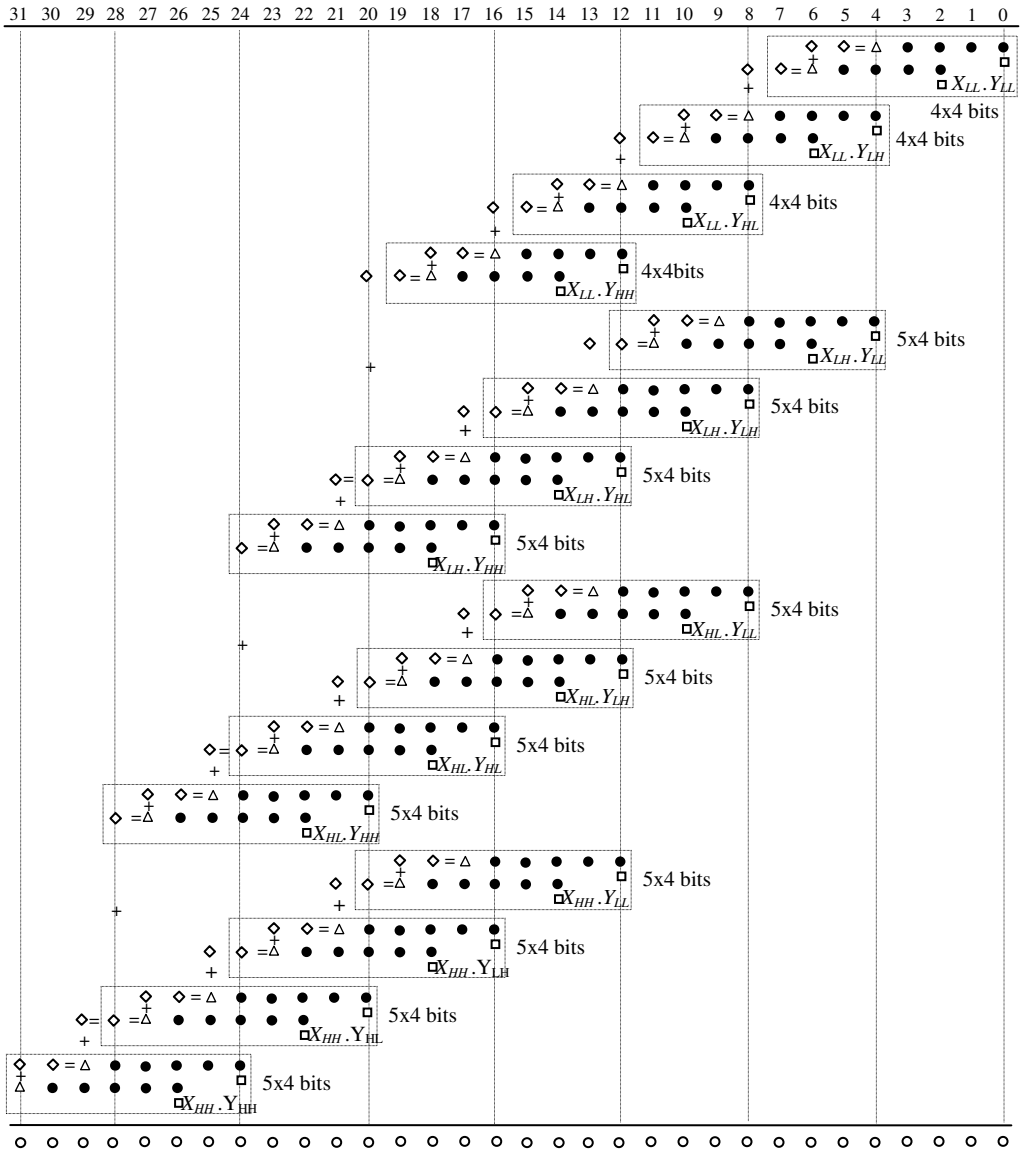


FIGURE 5.13 – Low-power multi-precision multiplier for the particular case (N,r)=(16,2) with 4-bit sub-operand size.

Bibliography

[1] A.K. Oudjida et al, "N Latency 2N I/O-Dandwidth 2D Array Matrix Multiplication Algorithm," The International Journal for Computation and Mathematics in Electrical & Electronics Engineering "COMPEL", Vol. 21, Issue 3, pp. 377-392, ISSN 0332-1649, UK, 2002.

[2] A.K. Oudjida et al, "Mapping Full-Systolic Arrays For Matrix Product On Xilinx's XC4000(E,EX) FPGAs," The International Journal for Computation and Mathematics in Electrical & Electronics Engineering "COMPEL", Vol. 21, Issue 1, pp. 69-81, ISSN 0332-1649, UK, 2002.

[3] Reports on System Drivers of the International Technology Roadmap for Semiconductors (ITRS), 2009 and 2010. Available: www.itrs.net/reports.html

[4] P.M. Seidel, L. D. McFearin, and D.W. Matula, "Secondary Radix Recodings for Higher Radix Multipliers," IEEE Trans. on Computers, vol. 54, N°2, February 2005.

[5] V.S. Dimitrov, K.U. Järvinen, and J. adikari, "Area Efficient Multipliers Based on Multiple-Radix Representations," IEEE Trans. on Computers, vol. 60, N° 2, pp 189-201, February 2011.

[6] H. Sam, and A. Gupta, "A Generalized Multibit Recoding of Two's Complement Binary Numbers and its Proof with Application in Multiplier Implementation," IEEE Trans. on Computers, vol. 39, N° 8, August 1990.

[7] B.J. Benschneider et al, "A Pipelined 50MHz CMOS 64-Bit Floating-Point Arithmetic Processor," IEEE Journal of Solid-State Circuits, vol. (24) 5, pp. 1317-1323, October 1989.

[8] C.F. Webb et al, "A 400-MHz s/390 Microprocessor," IEEE Journal of Solid-State Circuits, vol. (32) 11, pp. 1665-1675, November 1997.

[9] J. Clouser et al, "A 600-MHz Superscalar Floating-point Processor," IEEE Journal of Solid-State Circuits, vol. (34) 7, pp. 1026-1029,July 1999.

[10] R. Senthinathan et al, "A 650-MHz, IA-32 Microprocessor with Enhanced Data Streaming for Graphics and Video," IEEE Journal of Solid-State Circuits, vol. (34) 11, pp. 1454-1465, November 1999.

[11] A. Scherer et al, "An Out-of-Order Tree-Way Superscalar Multimedia Floating Point Unit," Proceeding of IEEE International Solid-State Circuits Conference (ISSCC), pp. 94-95, 1999.

[12] G. Kim et *al.*, "A Low-Energy Hybrid Radix-4/-8 Multiplier for Portable Multimedia Applications," Proceedings of IEEE International Symposium on Circuits and Systems, (ISCAS), pp. 1171-1174, Rio de Janeiro, Brazil, May 15-18, 2011.

[13] Intel Corp., "Intel 64 and IA-32 Architectures Software Developers Manual," volume 1, order number 253668, Copyright May 2011.

[14] R.J. Rieldlinger, "A 32 nm 3.1 Billion Transistor 12-Wide-Issue Itanium Processor for Mission-Critical Servers," Proceedings of IEEE International Solid-State Circuits Conference (ISSCC), pp. 84-86, San Francisco, CA ,USA, February 20-24, 2011.

[15] P.M. Seidel, L. D. McFearin, and D.W. Matula, "Secondary Radix Recodings for Higher Radix Multipliers," IEEE Trans. on Computers, vol. 54, N°2, February 2005.

[16] P.M. Seidel, L. D. McFearin, and D.W. Matula, "Binary Multiplication Radix-32 and Radix-256," Proceedings of the IEEE Symposium on Computer Arithmetic (ARITH-15), ISBN: 0-7695-1150-3, pp. 23-32, USA, June 2001.

[17] A. D. Booth, "A Signed Binary Multiplication Technique," Quarterly J. Mech. Appl. Math., Vol. 4, part 2, pp. 236-240,1951.

[18] O.L. McSorley, “High-Speed Arithmetic in Binary Computers,” Proceedings of the IRE, Vol. 49(1), pp. 67-91, January 1961.

[19] A.K. Oudjida, N. Chaillet, A. Liacha, and M.L. Berrandjia, “A New Recursive Multibit Recoding Algorithm for High-Speed and Low-Power Multiplier, ” Journal of Low Power Electronics (JOLPE), vol. 8, N° 5, pp. 579-594, ISSN: 1546-1998/2012/8/579/594, American Scientific Publishers (ASP), December 2012.

[20] A.K. Oudjida, N. Chaillet, A. Liacha, and M.L. Berrandjia “New High-Speed and Low-Power Radix-2^r Multiplication Algorithms,” Proceedings of the 11th edition of IEEE-FTFC Low-Voltage Low-Power Conference, ISSN: 978-1-4673-0821-2/12, Paris, June 2012.

[21] E. Manmasson et *al*., “FPGA in Industrial Control Applications,” IEEE Trans. on Industrial Informatics, vol. 7, N° 2, May 2011.

[22] M. Alioto, Elio Consoli, and Gaetano Palumbo, “Metrics and Design Consideration on the Energy-Delay Tradoff of Digital Circuits,” Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS’09), pp. 3150-3153, Taiwan, May 24-27 2009.

[23] S.R. Kuang, J.P. Wang, “Design of Power-Efficient Configurable Booth Multiplier,” IEEE Trans. on Circuit and Systems I, vol. 57, N° 3, March 2010.

[24] M. Annaratone, “Digital CMOS Circuit Design,” Kluwer Academic Publisher, 1986.

[25] M. Själander and P. Larsson-Edefors, “Multiplication Acceleration Through Twin Precision,” IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 17, N° 9, September 2009.

Chapter 6

Applications

Chapter 6

Applications

In this chapter we apply the results obtained in the previous chapters on the PID and LQG controllers with a Kalman filter. The new algorithms of multiplication by a variable (MV) and a constant (SCM/MCM) are applied to the PID and LQG controllers, respectively. We show through a practical implementation based on the design-reuse methodology, how these new algorithms contribute to designing high-speed, low-power, very compact, and highly scalable FWL controllers, which is the main objective of this thesis.

6.1 PID Controllers

The PID is by far the most commonly used closed-loop controller due to its simple structure and robust performance [1]. An important feature of this controller is that it does not require a precise analytical model of the system that is being controlled, which makes it very attractive for a large class of dynamic systems. While PID is well adapted for linear-time-invariant (LTI) systems [2], it stands powerless for non-LTI ones. Nevertheless some solutions exist, such as partitioning the non-LTI control algorithm into a linear part and a non-linear part [3][4][5]. The linear part represents the major control loop and is computed using an integrated PID, while the non-linear part that acts as dynamic compensation to the linear one is performed in software using a general-purpose-processor or a DSP.

In embedded control applications, such as in small-scale mobile robots, the control-loop-cycle is very tight and the power budget is very limited. A low sample rate leads to poor and degraded control-performance. And high power consumption shortens the battery lifetime. To cope with these two severe and *antagonistic* constraints, the need for both a *high-speed* and *low-power* PID structure is of utmost importance.

Today, design-reuse [6] is a well established design standard that allows grasping with rapid technology changes and increasing design complexity. It consists in the use of predesigned technology-independent, generic and reconfigurable IP-cores [7], most generally implemented at register-transfer-level (RTL).

However, at RTL abstraction level, no significant optimization results can be achieved if not undertaken at architectural and especially at algorithmic level. To achieve such a goal, a deep insight into PID *arithmetic* is necessary. At this stage, a choice of a numeric representation format is a crucial issue. Compared to floating-point, fixed-point format is the best candidate for optimized designs as it is much simpler to implement, faster, power-efficient and requires far much less hardware resources. However, the limited dynamic range can be source of control *instability*. This problem, referred to as finite-word-length (FWL) effect is an active research area that aims to shorten the floating-to-fixed point conversion time while preserving control performances [8][9].

The digital implementation of PID controllers went through several stages of evolution, initially dominated by the use of commercial-of-the-shelf (COTS) components and DSP. But over the past few years, FPGAs have brought a key advantage to digital control: the inherent parallelism of FPGA architecture allows many independent control loops to run at different deterministic rates without relying on shared resources that might slow down their responsiveness as in the case of COTS and DSP [10][11].

A survey of recent PID related works can be classified into three categories. The biggest one includes works that are straightforward FPGA implementations targeting specific applications: DC-DC converter [12], temperature control [13], motor multi-axis control [14], liquid level control [15], and Xilinx versus Altera FPGA implementation for result comparison [16]. The second category proposes methodologies that analyze the FWL effect on PID controller in order to reduce the number of hardware resources [17][18]. And finally the third category, paradoxically the smallest one despite the large popularity of PID, comprises architecture-optimization works. In [19] low-power serial and parallel multiple-channel PID architectures are proposed for small mobile robots. In this work, the optimization was carried out at macro-level considering several PIDs, rather than at micro-level (optimization of the PID itself). Nevertheless, the whole architecture will deliver much more interesting results if combined with an optimized PID. The second work [20] proposes serial, parallel, and mixed PID architectures incorporating different number (1-3) of multiplication cores. High power consumption, even with the serial architecture, and complex control-part are the two major shortcomings of this proposal. Finally, in [21] an attractive optimized PID structure based on distributed arithmetic (DA) is presented. Although this latter exhibits interesting results in terms of resource utilization and power consumption, it suffers from three serious drawbacks: high latency (n+1 clock-cycles for n bit set-point word-length), FPGA technology-dependent as it's essentially based upon FPGA look-up-tables (LUTs), and inability to handle time-varying PID parameters since they are precomputed and stored into LUTs. Nevertheless, it's considered as a reference design against which the obtained results are confronted into the same conditions.

The objective is to design optimized FWL-PID structures that overcome all above-mentioned shortcomings, and which are especially dedicated to embedded control applications. The PID cores are described at RTL level. They are highly reconfigurable and technology-independent, offering the possibility to be mapped both on FPGA and ASIC.

To reach such a goal, a special focus was put on the optimization of the *inner arithmetic* of PID. For that, we considered two discrete forms of PID algorithm: the commercial form [22], called also the standard or ISA form, and the incremental form. These two forms went through three successive types of FPGA implementations, using: Booth multiplication algorithm (BMA) [23], modified Booth multiplication algorithm (MBMA) [24], and a new developed version called recursive multibit recoding multiplication algorithm (RMRMA) [25].

Our previous paper [26] introduced a limited design-space of PID. In this chapter, we extended the design-space to accommodate different application cases and provided all necessary implementation details to make the design easily reproducible.

6.1.1 The Two Mostly-Used Discrete Versions of PID

A typical closed-loop system using a PID controller is shown in Fig. 6.1, where $u_c(k)$, $y(k)$, and $u(k)$ are the discrete signal quantities at the k^{th} sampling instant of the reference set-point, the process-

feedback measured output, and the PID controller output, respectively.

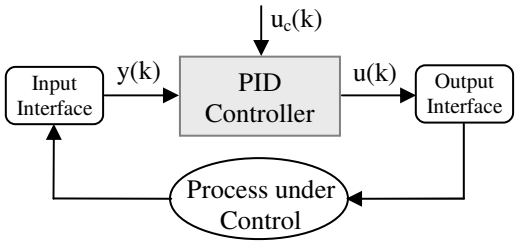


FIGURE 6.1 – Typical closed-loop control system using a PID.

In digital control, commercial and incremental forms are the two mostly-used discrete PID versions [1][22]. They are denoted by recurrent Eq. 6.1 and 6.2, respectively, and their corresponding coefficients are grouped in Table 6.1. Eq. 6.1 and 6.2 are fully detailed in the Appendix C.

$$u(k) = P(k) + I(k) + D(k)\ , \tag{6.1}$$

where $P(k) = A \cdot u_c(k) + B \cdot y(k)$; $I(k) = I(k-1) + C \cdot e(k-1)$; $D(k) = H \cdot D(k-1) + L \cdot f(k)$; with $e(k-1) = u_c(k-1) - y(k-1)$ and $f(k) = y(k) - y(k-1)$.

And $$u(k) = u(k-1) + A \cdot e(k) + B \cdot e(k-1) + C \cdot e(k-2), \tag{6.2}$$

where $e(k) = u_c(k) - y(k)$.

TABLE 6.1 – Coefficients of discrete recurrent equations.

Coefficients	Commercial PID	Incremental PID
A	$K_p\, b$	$K_p\left(1 + \frac{T_s}{T_i} + \frac{T_d}{T_s}\right)$
B	$-K_p$	$-K_p\left(1 + 2\frac{T_d}{T_s}\right)$
C	$-K_p \frac{T_s}{T_i}$	$K_p \frac{T_d}{T_s}$
H	$\frac{T_d}{T_d + NT_s}$	–
L	$-\frac{K_p T_d N}{T_d + NT_s}$	–

K_p is the proportional gain; T_i and T_d are the integral and derivative times, respectively; N is the maximum derivative gain; b is the fraction of set-point in proportional term; and T_s is the sampling period.

To satisfy different application cases, two IP versions are developed for each equation with constant coefficients and with varying coefficients (Fig. 6.2). This latter requires a host side interface (HSI) to handle the runtime change of the coefficients.

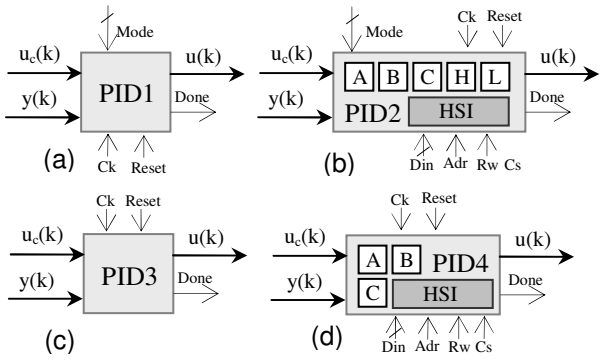


FIGURE 6.2 – Various PID IP-cores. (a) commercial PID with constant coefficients; (b) commercial PID with time varying coefficients; (c) incremental PID with constant coefficients; (d) incremental PID with time varying coefficients;

The commercial version allows the three standard PID functioning modes (P, PI, PID) according to Mode input value. At the end of u(k) computation, the Done output signal toggles during one clock cycle, and the PID enters into sleep mode (whole internal activity stopped except for clocking and HSI) for maximum energy conservation.

6.1.2 BMA Based PID

A straightforward parallel implementation of PID requires an amount of 7 adders/substractors and 5 multiplication cores for Eq. 6.1, and 4 adders/substractors and 3 multiplication cores for Eq. 6.2. In digital hardware, the total gate count scales linearly with word length for an adder core, while it scales quadratically for a multiplier core. Thus, any effort for a low-power optimization of PID must be focused on the implementation of the multiply-and-accumulate (MAC) function ($X \times Y$) [27]. In this work, the optimization effort is rather concentrated on the double MAC function ($X \times Y + T \times Z$) called DMAC, considered as the main building block of our PID structures. Eq. 6.1 and 6.2 are *partitioned* accordingly.

For FWL-PID, two's complement fixed-point representation is used, which is habitually expressed in Q notation as $Q_{ni.nf}$. The values are coded in n_i bits before the point (integer word length including 1 sign bit), and n_f bits after the point (fractional word length). The total word length is $n=n_i+n_f$.

Booth multiplication algorithm [23] belongs to the class of recoding algorithm, i.e. algorithms that recode one of the two operands to cope with signed two's complement multiplication. Let Y be the multiplier: Let Y be the multiplier:

$$Y = -y_{n-1}\,2^{n-1} + \sum_{j=0}^{n-2} y_j\,2^j\,; \tag{6.3}$$

$$Y = \sum_{j=0}^{n-1} \left(y_{j-1} - y_j\right) 2^j = \sum_{j=0}^{n-1} Q_j\,2^j\,, \tag{6.4}$$

where $y_{-1} = 0$ and $Q_j \in \{-1, 0, 1\}$.

Consequently, the multiplier Y is divided into n slices, each of 2 bits. Each pair of two contiguous slices has one bit in common. Thus, the DMAC becomes:

$$X.Y+T.Z=\sum_{j=0}^{n-1}\left(Q_j.X\right)2^j+\sum_{j=0}^{n-1}\left(P_j.T\right)2^j \quad (6.5)$$

$$=\sum_{j=0}^{n-1}\left[Q_j.X+P_j.T\right]2^j \quad (6.6)$$

According to Eq. 6.5, Booth algorithm consists in recoding the multiplier Y into a set of ternary numbers $\{-1,0,1\}$ in order to generate n *simple* partial products which are summed subsequently. Table 6.2 summarizes the 4 possibilities that may occur. The $-X$ can be easily formed by adding 1 to the complement of X. A direct translation of DMAC Eq. 6.5 into architecture (Fig. 6.3) requires one extra adder and two registers in comparison with the optimized version (Fig. 6.4) based on Eq. 6.6, called ODMAC. Additionally, one clock cycle latency is also needed in Fig. 6.3. The control-part responsible of producing the successive couples (y_{j-1} , y_j) is insignificant: just one multiplexer driven by a counter.

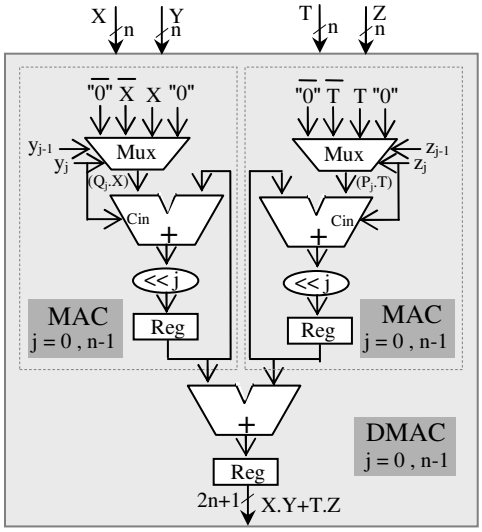


FIGURE 6.3 – Straightforward DMAC implementation.

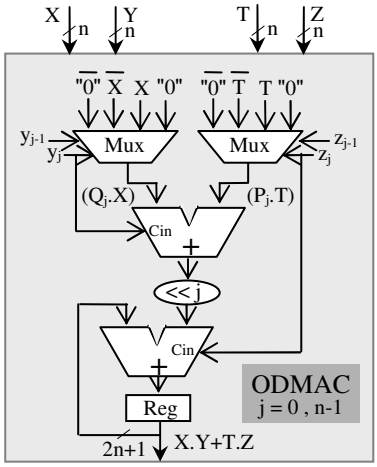


FIGURE 6.4 – Optimized DMAC implementation.

TABLE 6.2 – Booth algorithm.

Y_j	Y_{j-1}	Operation
0	0	+ 0
0	1	+ X
1	0	– X
1	1	– 0

Based upon ODMAC as the main building block, PID architectures are constructed for both incremental (Fig. 6.5) and commercial (Fig. 6.6) forms, and their implementation results (Table 6.3) are respectively compared to those of [21]. Comparison was made into identical conditions using the same FPGA device (Spartan XC2S50E-7FT256), although relatively old, as well as the same synthesis-tool version (Xilinx ISE 9.1i). In [21], only a 16-bit word-length commercial version with constant coefficients (without HSI) is implemented. PID1 and PID3 exhibits interesting results: 44%, 25%, and 32% savings and 62%, 35%, and 38% savings in terms of gate count, power, and speed, respectively. PID3 exhibits higher savings but at the expense of control-quality. Latency is rather the same (17), which is n+1 clock cycles for all designs (PIDX).

Optimizing latency without sacrificing the three other issues is the main objective of the two next sections.

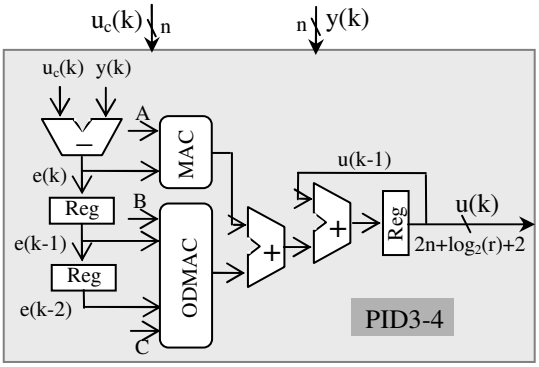


FIGURE 6.5 – Incremental PID architecture.

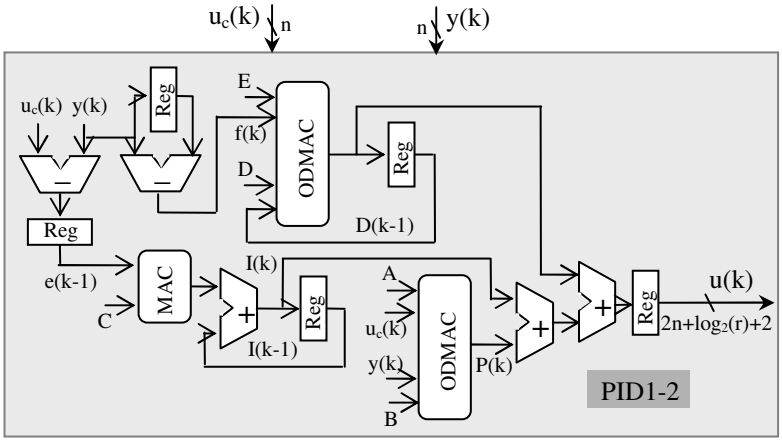


FIGURE 6.6 – Commercial PID architecture.

TABLE 6.3 – Implementation result comparison of MBA-based PID.

PID Core	Total Gate Count	Power* (mW)	Max. Clock Freq. (MHz)	Latency
PID [21]	16728	456	47	17
PID1	9286 (44%)	342 (25%)	62 (32%)	
PID2	10661 (36%)	359 (21%)	61 (30%)	
PID3	6337 (62%)	297 (35%)	65 (38%)	
PID4	7168 (57%)	308 (32%)	62 (32%)	

* : Dynamic power consumption at 47MHz; (XX%): saving.

6.1.3 MBMA Based PID

Eq. 6.3 can also be rewritten as follows [24]:

$$Y = \sum_{j=0}^{(n/2)-1} \left(y_{2j-1} + y_{2j} - 2y_{2j+1} \right) 2^{2j} = \sum_{j=0}^{(n/2)-1} Q_j \; 2^{2j} \;, \tag{6.7}$$

where $y_{-1} = 0$ and $Q_j \in \{-2, -1, 0, 1, 2\}$.

In this case, the multiplier Y is divided into $n/2$ slices, each of 3 bits, with one bit overlapping between adjacent slices. The proof of Eq. 6.7 is given in [28]. Thus, the DMAC equation becomes:

$$X.Y+T.Z=\sum_{j=0}^{(n/2)-1}\left[Q_j.X+P_j.T\right]2^{2j}. \qquad (6.8)$$

Likewise, n/2 *simple* partial products are generated (Table 6.4). Since ODMAC is a reconfigurable RTL block, it is parameterized to suit Eq. 6.8. The new adapted ODMAC architecture is depicted in Fig. 6.7. The only difference is that Mux(8:1) are used instead of Mux(4:1), and (<<2.j) hardwired shifter instead (<<1.j). Compared to BMA based PID (Table 6.5), MBMA based one (PID1) shows much more interesting results, since latency is divided by 2 while maintaining stable power consumption and speed. Control rate is drastically improved as its equal to maximum clock frequency divided by latency. As the discrete commercial form (Eq. 6.1) can accommodate the three functioning modes, implementation of PID2 produced the following power consumption values at 47 MHz: 268 mW, 313 mW, and 366 mW for P, PI, and PID functioning modes, respectively.

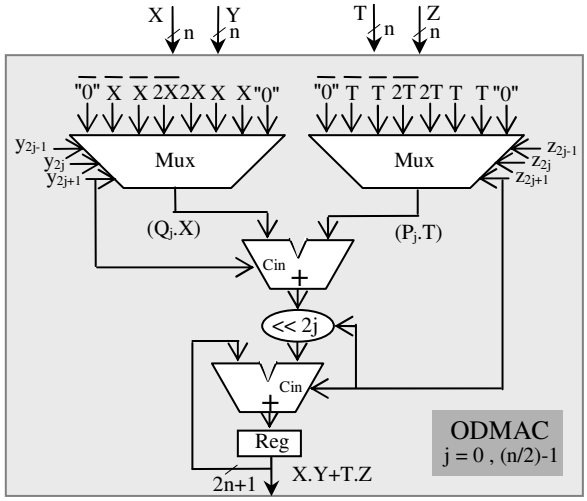


FIGURE 6.7 – Optimized DMAC architecture for *r*=2

TABLE 6.4 – Modified Booth algorithm.

Y_{2j+1}	Y_{2j}	Y_{2j-1}	Operation
0	0	0	+ 0
0	0	1	+ X
0	1	0	+ X
0	1	1	+ 2X
1	0	0	- 2X
1	0	1	- X
1	1	0	- X
1	1	1	- 0

TABLE 6.5 – Implementation result comparison of MBMA-based PID.

PID Core	Total Gate Count	Power* (mW)	Max. Clock Freq. (MHz)	Latency
PID [21]	16728	456	47	17
PID1	10642 (36%)	350 (23%)	62 (32%)	9 (47%)
PID2	11923 (29%)	366 (20%)	61 (30%)	
PID3	7042 (58%)	303 (33%)	64 (38%)	
PID4	7795 (53%)	315 (31%)	62 (32%)	

* : Dynamic power consumption at 47MHz; (XX%): saving.

With regard of these improvements, one is encouraged to pursue farther [24] in reducing latency by considering larger slices, such as:

$$Y=\sum_{j=0}^{(n/3)-1}\left(y_{3j-1}+y_{3j}+2.y_{3j+1}-2^2\,y_{3j+2}\right)2^{3j}=\sum_{j=0}^{(n/3)-1}Q_j\,2^{3j}, \qquad (6.9)$$

where $y_{-1}=0$ and $Q_j\in\{-4,\ldots,0,\ldots,4\}$.

But in this case, some *hard* partial products are required such as 3X and –3X which are not easy to generate. How to circumvent this obstacle is the purpose of the next section.

6.1.4 RMRMA Based PID

Multiplication is a fundamental operation in digital design. Its speed and power requirements are two critical factors limiting the whole system performances (PID in our case). Since the publication of Booth's algorithm in 1951, a huge number of improvement attempts were proposed, especially after the publication of a generalized version of MBA algorithm accompanied with its proof [29]. Most of the proposals aimed to reduce the number of partial products either by employing digital optimization techniques [30][31][32] or by using larger slices (higher radices) [33]. However, experience showed [34] that beyond 4-bit slices (radix 8), the complexity to generate hard partial products can not be managed in a realistic way. In [34], three metrics are provided for comparing the tradoffs when employing higher radix Booth recodings: partial product compression factor (gain), the number of hard multiples that must be precomputed (computation complexity), and partial product generation fanin (routing complexity).

To circumvent the problem of hard partial products in higher radices, the idea proposed in [35] is to apply a recursive Booth recoding on the *r*-bit slice. While the idea is interesting, it relies upon a complicated mathematical formulation, leading to a complex control circuitry and especially to an exaggerated latency (2n/r).

According to the multibit recoding algorithm presented in [29], an n-bit two's complement operand Y can be written as:

$$Y = \sum_{j=0}^{(n/r)-1} \left(y_{rj-1} + 2^0 y_{rj} + 2^1 . y_{rj+1} + 2^2 y_{rj+2} + \cdots + 2^{r-2} y_{rj+r-2} - 2^{r-1} y_{rj+r-1} \right) 2^{rj} = \sum_{j=0}^{(n/r)-1} Q_j 2^{rj} \ , \quad (6.10)$$

where $y_{-1} = 0$; $r \in \mathrm{N}^*$; and $Q_j \in \left\{ -2^{r-1}, \ldots, 0, \ldots, 2^{r-1} \right\}$.

In this general case, the multiplier Y is divided into n/r slices, each of r+1 bits. Each pair of two contiguous slices has one overlapping bit. To bypass the problem of *hard* partial products, MBMA (Eq. 6.7) is applied to the Q_j terms. Thus, Eq. 6.10 takes the new simpler recursive form:

$$Y = \sum_{j=0}^{(n/r)-1} \left[\left(y_{rj-1} + y_{rj} - 2.y_{rj+1} \right) 2^0 + \left(y_{rj+1} + y_{rj+2} - 2.y_{rj+3} \right) 2^2 + \ldots + \left(y_{rj+r-5} + y_{rj+r-4} - 2.y_{rj+r-3} \right) 2^{2(\frac{r}{2}-2)} + \left(y_{rj+r-3} + y_{rj+r-2} - 2.y_{rj+r-1} \right) 2^{2(\frac{r}{2}-1)} \right] 2^{rj} \quad (6.11)$$

$$= \sum_{j=0}^{(n/r)-1} \left[\sum_{i=0}^{(r/2)-1} \left(y_{rj-1+2i} + y_{rj+2i} - 2.y_{rj+1+2i} \right) 2^{2i} \right] 2^{rj} \quad (6.12)$$

$$= \sum_{j=0}^{(n/r)-1} \left[\sum_{i=0}^{(r/2)-1} Q_{ji} \, 2^{2i} \right] 2^{rj}, \quad \text{with } Q_{ji} \in \left\{ -2, -1, 0, 1, 2 \right\}. \quad (6.13)$$

There is no need to prove Eq. 6.11 since it is a combination of Eq. 6.10 and 6.7 which were already proven in [29] and [28], respectively. The partitioning of operand Y according to Eq. 6.13 is illustrated by Fig. 6.8. To avoid dealing with special cases, n and r must be chosen as even numbers, with r as a divider of n. Thus, the DMAC equation becomes:

$$X.Y + T.Z = \sum_{j=0}^{(n/r)-1} \left[\sum_{i=0}^{(r/2)-1} \left(Q_{ji}.X + P_{ji}.T \right) 2^{2i} \right] 2^{rj} \ . \quad (6.14)$$

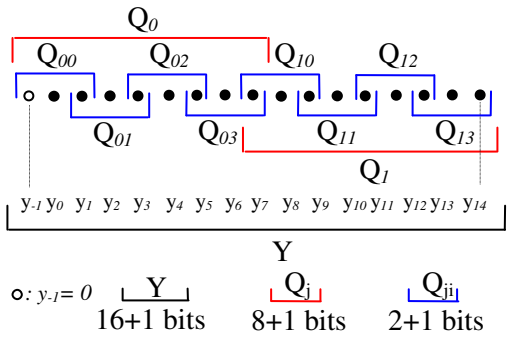


FIGURE 6.8 – Partitioning of a 16-bit Y operand with r=8.

Depending on r value ranging from 2 to n, PIDs with various levels of parallelism and latencies ($n/r+1$) can be automatically generated with slight control complexity. The special cases of $r=n$ and $r=2$ correspond to fully-parallel and fully-sequential PID, respectively. In between ($r=4,n/2$), partially-parallel PIDs are obtained. The outstanding advantage of this algorithm (Eq. 6.13) is that *hard* partial products are generated using *simple* ones ($2X$, X) only. For a simplified hardware and lower power consumption, the step-by-step *sign-propagate* technique is employed [36].

Obviously, Eq. 6.13 does not reduce the number of partial products, but allows a modulable space-time partitioning of the multibit recoding algorithm (Eq. 6.10), where n/r sets comprising each $r/2$ partial products can be generated and summed either simultaneously or iteratively. Whilst the parallel implementation of Eq. 6.13 allows an important reduction of the critical path (using a carry-save adder CSA), it requires too much space. Therefore, only the serial implementation is retained. In this case, latency drops from ($n/2+1$) to ($n/r+1$), whereas the overhead on the total critical path, which goes through $\log_2(r/2)$ adder levels and which is equal to D in the case of MBMA, is slightly increased $D+\log_2(r/2)$. Note that we are using a logarithmic summation tree and not a linear one (CSA like).

An illustrative serial example with $r=4$ is described as follows:

$$Y = \sum_{j=0}^{(n/4)-1} \left(y_{4j-1} + y_{4j} + 2\,y_{4j+1} + 2^2\,y_{4j+2} - 2^3\,y_{4j+3} \right) 2^{4j} \tag{6.15}$$

$$= \sum_{j=0}^{(n/4)-1} \left[\sum_{i=0}^{1} \left(y_{4j-1+2i} + y_{4j+2i} - 2\,y_{4j+1+2i} \right) 2^{2i} \right] 2^{4j} \tag{6.16}$$

$$= \sum_{j=0}^{(n/4)-1} \left[Q_{j0} + Q_{j1}\,2^2 \right] 2^{4j} \tag{6.17}$$

$$X.Y + T.Z = \sum_{j=0}^{(n/4)-1} \left[\left(Q_{j0}X + P_{j0}T \right) + \left(Q_{j1}X + P_{j1}T \right) 2^2 \right] 2^{4j} \tag{6.18}$$

The mapping of Eq. 6.18 into a serial architecture is shown in Fig. 6.9. Such a case ($r=4$) would have required the computation of hard partial products ($7X$, $5X$, $3X$) if the simple form of Eq. 6.15 was used. Notice that MBMA is a special case of RMRMA for $r=2$. For $r=1$, Eq. 6.10 corresponds to BMA (Eq. 6.4).

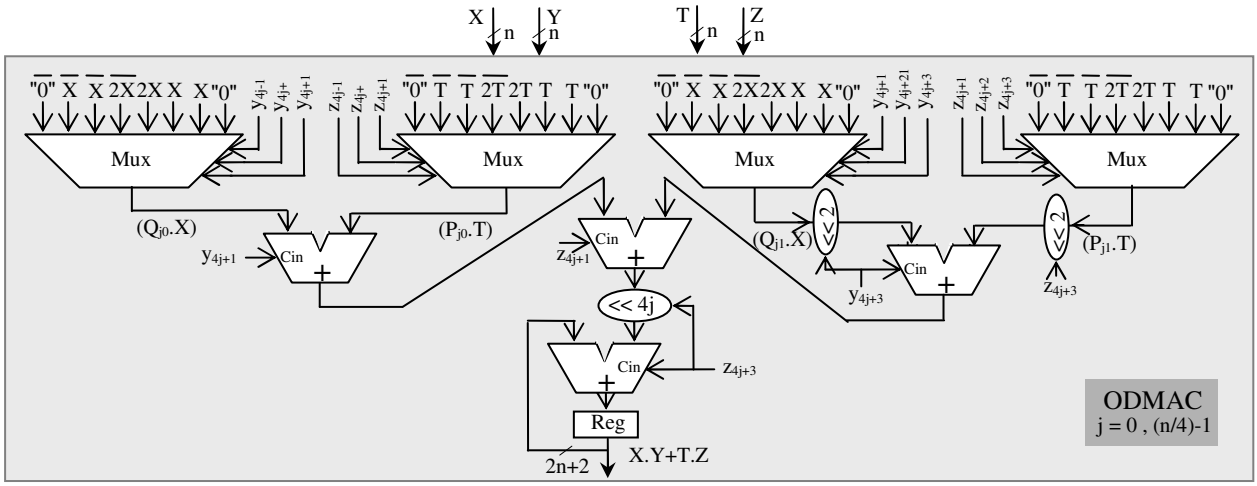


FIGURE 6.9 – Optimized DMAC architecture for r=4.

Table 6.6 comprises the implementation results of PIDs with n=16 and r=4,8,16. For instance, PID1 with r=4 not only achieves high improvement in latency (71%), but also maintains positive savings in power (14%) and speed (13%). These important achievements are partially due to logic-trimming performed by the synthesis tool on the constant coefficients. Such an operation is impossible in the case of PID [21] since the coefficients are stored into LUTs.

TABLE 6.6 – Implementation result comparison of RMRMA-based PID.

PID Core	Total Gate Count	Power* (mW)	Max. Clock Freq. (MHz)	Latency
PID [21]	16728	223	47	17
PID1_4	12443 (+26%)	191 (+14%)	53 (+13%)	5 (+71%)
PID1_8	15688 (+06%)	194 (+13%)	44 (-06%)	3 (+82%)
PID1_16	23545 (-41%)	217 (+03%)	26 (-45%)	2 (+88%)
PID2_4	22962 (-37%)	256 (-15%)	43 (-08%)	5 (+71%)
PID2_8	26073 (-56%)	204 (+08%)	37 (-21%)	3 (+82%)
PID2_16	40327 (-141%)	488 (-119%)	23 (-51%)	2 (+88%)

*: Dynamic power consumption at 23MHz; PIDY_X: X = r;
(+AB%): saving; (-AB%): overhead.

At this stage, a key question arises: among this panoply of PIDs, which one fits the best one's application case? The answer to this question is given in the next section.

6.1.5 Discussion

In embedded control, satisfactory control-rate (without performance degradation) at minimum power consumption is the main requirement. To select the most adequate PID for a given application, it's necessary to investigate how speed, power and hardware resources scales versus r factor for a fixed word length n. Referring to Eq. 6.14 and aided by Fig. 6.9, the ODMAC architecture scales as a binary tree with one stage of r mux(8:1) followed by $Log_2(r)+1$ stages of adders with a total of r adders too. Thus, the total delay cumulated by the critical path which goes through $Log_2(r)+2$ stages increases with $O(Log(r))$ complexity, whilst latency $(n/r+1)$ decreases linearly $O(r)$, which makes the maximum control-rate increases as r increases. This is confirmed by implementation results shown in Table 6.7

and 6.8 corresponding to PID1 and PID2, respectively. The sole exception to this general rule is PIDX_n/2 which always yields to the highest control-rate compared to PIDX_n despite the numerous tests with various n values. This is justified since they exhibit very close latencies (3 and 2, respectively) and one stage difference in the critical path (n-1 and n, respectively), but an important multiplexer fanin difference (n/4 and n/2, respectively).

TABLE 6.7 – Maximum power-consumption and control-loop-cycle of PID1.

PID Core	Power* (mW)	Max. Clock Freq. (MHz)	Latency	Max. Control Loop Cycle (MHz)
PID [21]	456	47	17	2.76
PID1_1	342 (+25%)	62	17	3.65 (+32%)
PID1_2	350 (+23%)	62	9	7.66 (+177%)
PID1_4	431 (+05%)	53	5	10.60 (+284%)
PID1_8	365 (+20%)	44	3	14.67 (+431%)
PID1_16	244 (+46%)	26	2	13.00 (+371%)

*: Dynamic power consumption at maximum clock frequency; PID1_X: X=r;
Max. control loop cycle = Max. clock frequency / Latency.

TABLE 6.8 – Maximum power-consumption and control-loop-cycle of PID2.

PID Core	Power* (mW)	Max. Clock Freq. (MHz)	Latency	Max. Control Loop Cycle (MHz)
PID [21]	456	47	17	2.76
PID2_1	466 (-02%)	61	17	3.59 (+30%)
PID2_2	475 (-04%)	61	9	6.78 (+146%)
PID2_4	479 (-05%)	43	5	8.60 (+211%)
PID2_8	328 (+28%)	37	3	12.33 (+347%)
PID2_16	488 (-07%)	23	2	11.50 (+317%)

*: Dynamic power consumption at maximum clock frequency; PID2_X: X = r;
Max. control loop cycle = Max. clock frequency / Latency.

In terms of resource occupation, the total complexity grows linearly $O(r)$ as r multiplexers and r adders are required by ODMAC which is the most resource consuming block of PID architecture. This is also confirmed by the implementation results shown in Table 6.6. Note that each adder of each level of MAC and ODMAC as well as the two ones at the output of the PID (Fig. 6.5 and 6.6) are successively extended by one bit so that the total bit size of the control output u(k) becomes $2n+\log_2(r)+2$. It's necessary to do so to prevent the apparition of a possible overflow in the data-path which can cause signal clipping and instabilities in the closed loop response [37].

As for power consumption, intuitively, one would expect to see PID1_16 of Table 6.7 as being the most rapid and the most power consumer too, for the reason that it exhibits the smallest latency and the biggest total gate count! While it is almost true for the latter (13 MHz, before the first), it is quite the opposite for the former (244 mW, the smallest one). The explanation is that power consumption ($P = 0.5\, V_{dd}^2\, C_{sw}\, F_{clk}$) depends linearly on the frequency (F_{clk}), which is in this case 26 MHz (the smallest one) and also on the switched capacitance (C_{sw}) which describes the average capacitance charged during each clock period ($1/F_{clk}$). In fact, C_{sw} depends on a number of parameter (circuit structure, logic function, input pattern dependence…) and not only on the total gate count (more precisely, not only on the total physical capacitance of the circuit). Furthermore, a study [38] that analyzed the dynamic power consumption in Xilinx's FPGA revealed the following share: 60% by

routing, 16% by logic, and 14% by clocking. The reason is that routing is intensively segmented, using pass logic and buffers.

When both high control-rate close to 13MHz and low power are required, PID1_16 (244 mW at 13MHz) stands as the best candidate compared to PID1_8 (323 mW at 13MHz). However, it's noteworthy to mention that this comparison stands valid only for the special case of 16-bit word-length PID, for a given set of coefficients, mapped on XC2S150E-7FT256 FPGA circuit and using Xilinx's XST synthesis tool, version 9.2. Results could significantly change under other conditions, especially when considering the logic trimming process which is essentially dependant on the bit-arrangement of the coefficients. For a minimum influence of the trimming operation on the synthesized results, appropriate coefficients were used such as all Q_j terms are represented except the null one to avoid generating null partial products that greatly simplify the circuit logic. In fact, constant coefficients PIDs (PID1) are somehow unpredictable with regard to r. They are coefficient dependant. Adversely, PID2 is not involved with the trimming process since coefficients are time varying. Implementation results comprised in Table 6.8 show that PID2_8 is the best at all aspects for the same reasons cited above. In sum, when high control-rate is the ultimate objective, PIDX_n/2 is the best candidate whatever n value. But in the case where both high speed and low power are required, timing and power evaluations are necessary to decide which PID to select: either PIDX_n/2 or PIDX_n.

Finally, when only low power is targeted, PIDX_1 is the best candidate. We dealt here with extreme situations only, but for a given couple (cr, pc) of control-rate and power consumption, several candidates are possible. Yet, the best PID is the one which requires the smallest gate count.

So far, speed and power have been considered in isolation to area which becomes critical, and sometimes prohibitive, for large word-length n due to the fact that PID is basically built of a set of multipliers (three or five) that scale quadratically with word length. The bigger is the area, the higher is the cost. Consequently, another advantage of RMRMA algorithm is to cope also with the cost issue as an additional constraint to speed and power.

We have deliberately chosen Spartan2e FPGA to compare our results with those provided in [21]. A mapping on a recent FPGA circuit (Virtex6) using XST 12.1 version of extreme PID2 delivered state-of-the-art results grouped in Table 6.9. Note that control-rate scaled with an average factor of 2, while power dissipation scaled with an average factor of 45.

TABLE 6.9 – Maximum power-consumption and control-loop-cycle of PID2 mapped on Virtex6.

PID Core	Number of Slices	Power* (mW)	Max. Clock Freq. (MHz)	Latency	Max. Control Loop Cycle (MHz)
PID2_1	231	23	122	17	07.17
PID2_8	1060	04	90.5	3	30.16
PID2_16	1963	13	50.4	2	25.19

*: Dynamic power consumption at maximum clock frequency; PID2_X: X=r; Max. control loop cycle=Max.clock frequency / Latency.

This is not surprising, since Spartan2e and Virtex6 were fabricated with two differently scaled technology processes: 150 nm and 40 nm, respectively. Therefore, the physical capacitances of the circuit in Virtex6 are relatively too much smaller. Additionally, the supply-voltages (V_{dd}) used for internal core (Vccint) and for output blocks (Vcco) are respectively 1.8V and 3.3V for Spartan2e, 1V and 2.5V for Virtex6. Furthermore, the efficient advances made in CAD tools (from Xilinx ISE 9.1 to

12.1 versions) as well as in FPGA architecture, such as advanced segmented-routing, much contributed to lower the power consumption [39]. Power consumption evaluation studies [38][39] based on simulation and measurements, targeting Virtex2 and Virtex6 families revealed the following results: 5.9µW per CLB per MHz, and 1.09 mW per 100 MHz at 38% toggle rate, respectively. These studies roughly confirm our power results as proximate values are obtained.

Timing and power evaluations were performed in the following conditions. Delays were calculated for two types of paths: Clock-To-Setup and all paths together (Pad-To-Setup, Clock-To-Pad and Pad-To-Pad.) The Clock-To-Setup gives more precise information on the delays than other remaining paths, which depend in fact on I/O Block (IOB) configuration (low/high fanout, CMOS, TTL, LVDS…). Thus, all delays (frequencies) presented so far are clock-to-setup delays with the highest speed grade of the FPGA circuit. As for power, we chose the highest Vcco voltage value (3.3 for Spartan2e and 2.5 for Virex6) with a maximum toggle activity of 50%, which means that Flip-Flops (FFs) toggle one time during each clock cycle. The reason is that only simple-edge triggered FFs are used for synthesis (no double-edge FFs).

6.1.6 Verification Method

The PID design verification process went through several steps. First Eq. 6.12 and 6.14 were tested with a random C-program. Then, a severe cycle-accurate functional verification procedure using Modelsim simulator was applied to MAC and ODMAC as they are the main building blocks of PID architecture. They were challenged against a set of special test cases (visual simulation), and then submitted to a random test for a very large number of vectors. Once tested successfully, the RTL PID module written in Verilog-2001 (IEEE 1364) was integrated into Modelsim/Simulink environment for a co-simulation. At this stage, a ZOH discrete time invariant model of a third order continuous process ($G(s)=1/(s+1)^3$) was chosen from the test set used by Åström and Hägglund [1] as examples of representative plants for the dynamics of typical industrial processes (Fig 6.10).

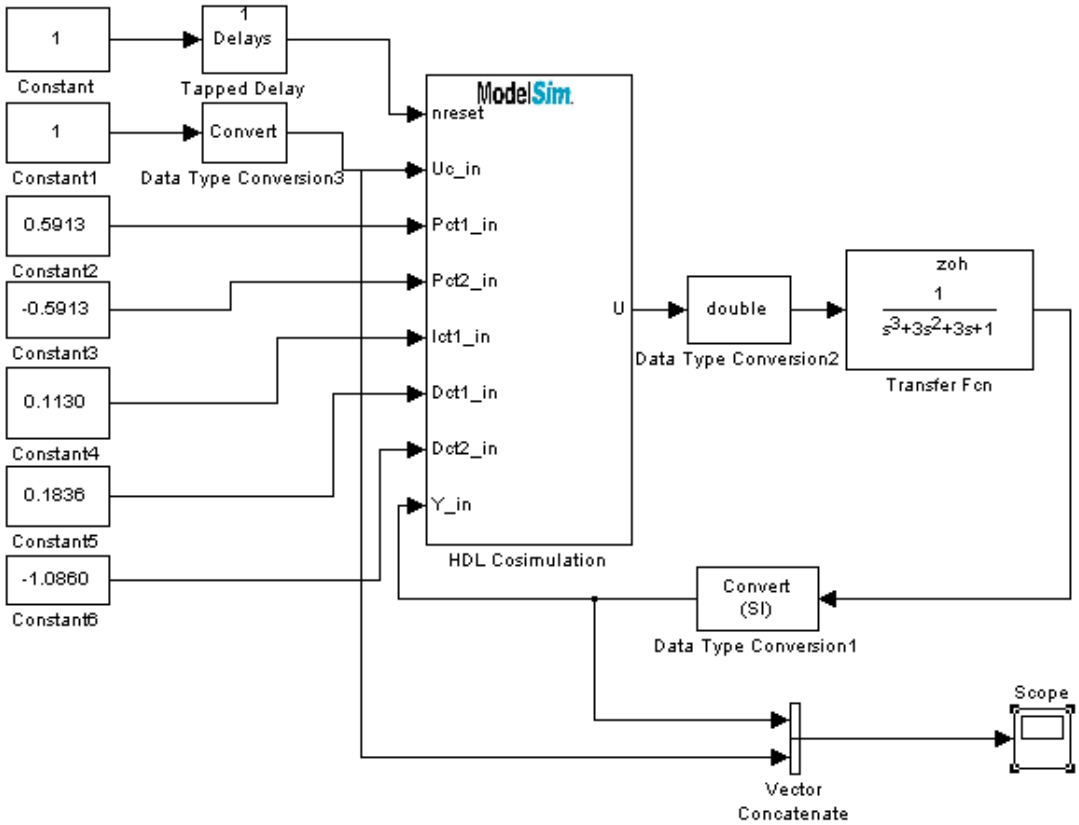


FIGURE 6.10 – The co-simulation of the PID in the Simulink/Modelsim environment.

To derive the PID parameters, a theoretical PID taken from Matlab component-library was tuned using floating-point numerical representation (IEEE 754 double format). The sampling period T_s was chosen based on the magnitude of the pole time constants. For this case T_s=10 ms. The following parameters were obtained:

$$K_p = 0.5913 \;;\; T_i = 0.0523 \;;\; T_d = 0.0225 \text{ for N=10 and b=1.}$$

Calculations give the following floating-point values for the coefficients of commercial PID:

$$A=0.5913;\ B=-0.5913;\ C=0.1130;\ D=0.1836;\ E=-1.0860$$

To co-simulate the RTL PID, a conversion of the coefficients to 16-bit ($Q_{4.12}$) fixed-point representation was necessary. Variations were obtained:

$$A=0.5911;\ B=-0.5911;\ C=0.1130;\ D=0.1836;\ E=-1.0860$$

Note that to represent the original parameters with full-precision, 44 bits are needed for the fractional part. Varied simulations were performed to verify the correctness of the PID RTL code. First, to explore the precision effect on control quality, the control output of PIDs with various fractional-part sizes ($Q_{4.4}$, $Q_{4.12}$, $Q_{4.20}$) were compared to that of the Matlab floating-point PID component (Fig. 6.11). Simulation shows different rise-times for different precisions. The higher is the precision; the closer is the control output from the ideal model. The second simulation tests the behavior of the PID after having reached the steady state (Fig. 6.12). For that, two perturbations are successively exerted on control output and on the plant measure. Each time the system recovers as expected. And finally, the third simulation investigates the PID capabilities to track set-points of arbitrary amplitudes and durations (Fig. 6.13).

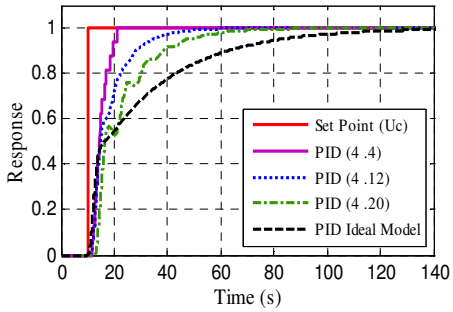


FIGURE 6.11 – Fixed-point versus floating-point.

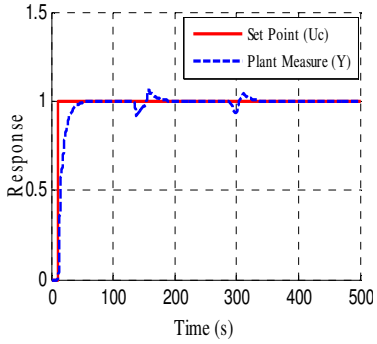


FIGURE 6.12 – Perturbation after steady-state on control-output and on plant measure, successively.

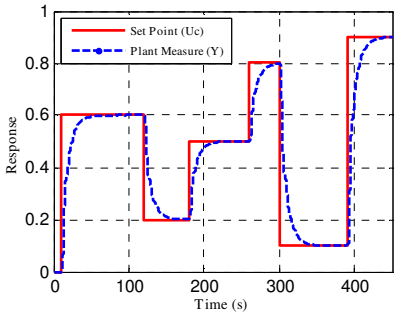


FIGURE 6.13 – Set-point tracking of arbitrary amplitudes and durations.

After a successful functional verification, the RTL code of PID was synthesized, placed, and routed on Xilinx's FPGA (Virtex-2). The three preceding co-simulations but with timing backannotation were performed again as a last necessary software verification step before hardware integration of the PID into an FPGA evaluation board (MEMEC V2MB1000).

Finally, as an ultimate validation step, a physical test of our PIDs is performed. We built up a classical temperature control setup (Fig. 6.14 and 6.15), which consists in a tube comprising a halogen lamp (12 V, 21 W), a temperature sensor (LM35), and a DC Fan (12 V, 1.68 W). Temperature regulation inside the tube is achieved by controlling either the intensity of the lamp, or the rotation speed of the fan. This is carried out by the use of two PWMs, whose duty-cycle ratios represent the PID controller output (u(k)). These two PWMs do not act directly on the fan or on the lamp but rather on transistors (IRF540) that control the power consumed by the lamp and fan.

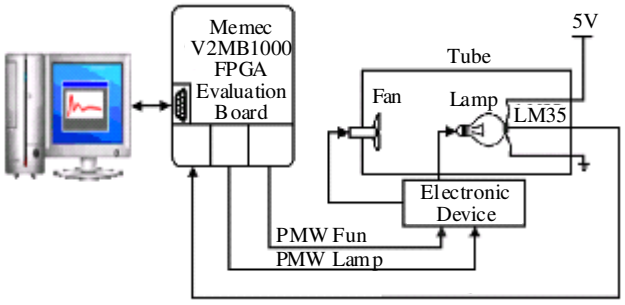


FIGURE 6.14 – Synoptic scheme of the setup.

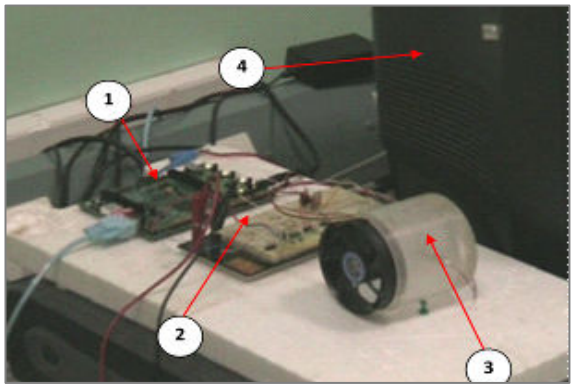


FIGURE 6.15 – Setup of temperature regulation. 1: FPGA evaluation board; 2: Electronic device; 3: Tube containing a fan and a lamp, 4: PC display screen.

The sensing of the actual temperature of the tube is assured by LM35 component which delivers a voltage value that grows linearly with temperature (1.5 volts corresponds to 150 °C). As the maximum voltage allowed by FPGA evaluation board (V2MB1000) is 3.3 Volts, the calculation of the real temperature (T) is done as follows: T = [(val_opb_ADC * 3.3)/1023] * 100. This allows a temperature control with a minimum step of 0.32 °C.

The V2MB1000 board is connected through RS232 port to a PC running a .net application which allows a real-time display of the temperature as well as an instantaneous tuning of the set-point.

6.1.7 The Finite-Word-Lenght (FWL) Effect

Fixed-point arithmetic is employed as an approximation of real numbers (floating-point), with a fixed bit-length of the word used to represent data (Finite Word-Length). This limitation leads to performance degradation (FWL effect) mainly due to quantization of coefficients (parametric errors) and roundoff errors subsequently cumulated during the computation process (numeric noise). In fact, the FWL effect is more-or-less exaggerated depending on the control algorithm used (I/O relationship, levels of parallelism, etc) as well as on the way the computations are performed (number of bits, different/unique fixed point position, round/truncation, etc). Compared to the reference floating-point implementation, the FWL effect can be assessed using some indicators such as transfer function sensitivity, or pole sensitivity [40][41][42].

In fact, the objective is twofold: we need to provide an optimal ASIC/FPGA implementation of FWL PID without degrading control performances. To achieve such a goal, a double expertise is required in hardware design and control system. But usually, hardware designers do not master control system design, and control system experts do not have the required skills to implement and evaluate the controllers using ASIC/FPGAs [17][43]. This is why we propose a highly reconfigurable (n, r) and technology-independent FWL PID that can systematically respond to control-engineer demands after having modelled, simulated, and evaluated the performances provided by different bit-width fixed-point representations using Matlab/Simulink environment, and finally opted for an appropriate word-length (n) of the setpoint. As for latency value (r), it depends on the application domain and intended objectives. Precise guidelines on how to choose r value were given in Section 6.1.5.

Now that (n, r) couple is known, the FWL problem is tackled from hardware side by simply adjusting

in the RTL code the two compile-time constants: setpoint bit-size (n) and latency (r). The synthesis of such a PID generates an optimal structure that not only meets the performances specified by control-engineers, but also consumes minimum power and hardware resources. This would not have been possible without the use of the new highly serialisable multi-bit multiplication algorithm (Eq. 6.13). The incorporation of Eq. 6.13 [25] into Eq. 6.1 and 6.2 as an efficient PID engine, allows the generation of PID architectures classified as regular iterative architectures (RIA) [44], known for their high conformity with the principles of regularity and locality. In addition to Eq. 6.13, we highly recommend to use the optimal multiplier (Eq. 5.22) for high values of n (n≥16). As for low values of n, we have proposed in [25] several new highly serialisable multiplication algorithms, offering different features in power, space and delay, depending on the operand size (n). Reader is encouraged to explore these algorithms [25] to select the appropriate one that leads to best performances of its controller with regard to the size (n) of the setpoint.

Regularity and locality are two important features, highly sought in hardware design as they lead to an important gain in space and delay. Regularity is a general space feature, where the repetitiveness of just one or few elementary building-blocks (mux, adders and shifters of ODMAC, Fig. 6.9) and their interconnection scheme (predefined netlist) suffice to draw the whole architecture (MAC/ODMAC and then PID). In the other hand, locality is both space and time feature, in the sense where each building-block can only interact with its nearest surrounding neighbours, and any transaction from one building-block to the next is completed in one and only one unit time delay (clock period). Because of these two important features, our PIDs can be finely grained at bit level in space (setpoint bit-size n, latency r) and unit delay in time (latency r).

Experimental results depicted in Fig. 6.16 illustrate the FWL effects on temperature regulation. Reducing the fractional-part size of the set-point beyond a certain limit (4 bits) yields to a continuous fluctuation of the temperature inside the tube (Fig. 6.16d). The best compromise is a 6-bit fractional-part (Fig. 6.16c) which ensures a correct regulation while consuming less power and hardware

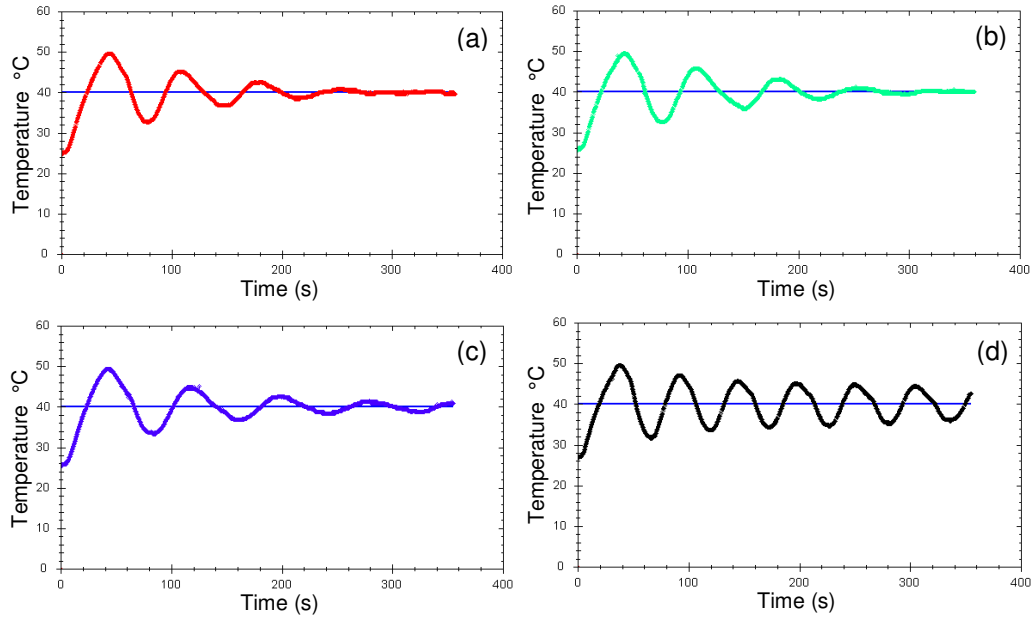


FIGURE 6.16 – Effect of the setpoint fractional length on temperature regulation.
(a) Floating point PID; (b) Our PID with $Q_{ni.nf} = Q_{8.8}$; (c) Our PID with $Q_{ni.nf} = Q_{8.6}$;
(d) Our PID with $Q_{ni.nf} = Q_{8.4}$

resources. As temperature regulation system has a very slow dynamic, speed is not a concern. Therefore, the most appropriate PID in this case is PIDX_1 as it is the least power consumer. Adversely, for very fast dynamic systems, such as MEMS [45] or microrobotics applications [46], PIDX_n/2 is the most adequate option as it leads to the highest control rate.

6.2 LQG Controller with Kalman Filter

Accurate and dexterous micromanipulation tasks are very important in a wide range of microrobotics applications such as microassembly tasks, minimally invasive surgery, genetics, in vitro fertilization, and cell mechanical characterization. In this case, the use of microgrippers and controlling gripping forces applied on manipulated samples in the micrometer range (i.e. between *1µm* and *1mm*) without destroying and damaging is still a great scientific and technological challenge [47].

The AS2M department of FEMTO-ST (Besançon) has been working for many years on problems related to the modelling and control of micro-systems for micro-assembly/micromanipulation applications. One of its numerous works is the modelling [47] and control [48] of the FT-G100 micro-gripper (Fig. 6.17) in order to enable dexterous micromanipulation tasks through gripping force sensing and control. The developed control model is based on the Linear Quadratic Gaussian (LQG) controller with Kalman filter.

We show hereafter how the Matlab model of the controller (LQG+Kalman) is gradually translated into a synthesisable Verilog code through the application of the new SCM/MCM heuristics introduced in Chapter 4.

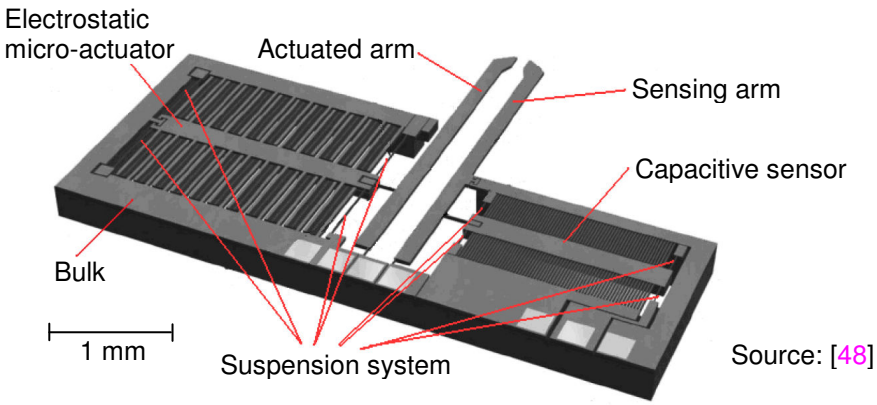


FIGURE 6.17 – Structure of the FT-G100 microgripper (Femto Tools GmbH).

6.2.1 Dynamic Model of the FT-G100 Micro-Gripper

The gripping force appears when the gripper arms are in contact with the manipulated object. The use of a coupled model of the gripper is necessary in order to take into account both dynamics of the actuated subsystem (electrostatic actuator + actuated arm) and the sensing one (sensing arm + capacitive sensor) (Fig. 6.18). The actuated and sensing subsystems are first modelled when an external force is applied at the tip of the gripper arms. Thereafter a simple modelling of the manipulated object is used to couple previous subsystems in the state space representation. Internal dimensions of the FTG100 microgripper provided by femtotools technical support

"support@femtotools.com" are used for the modelling approach. Higher dynamics of each subsystem are neglected as their effects are not significant. Thus, only second order models are considered [47].

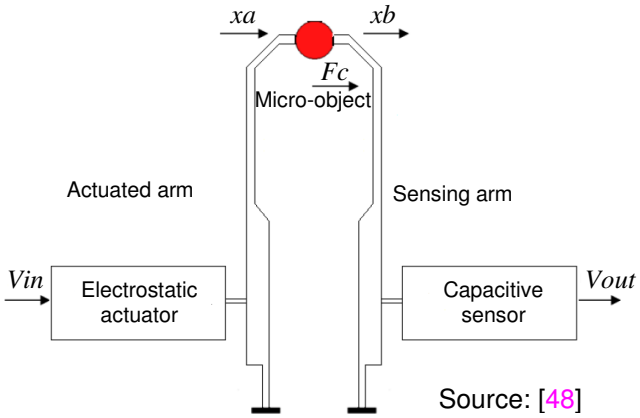


FIGURE 6.18 – System modelling.

Actuated System Modelling

Around an excitation voltage of 70V, the model of the discrete state of the actuated system identified during the gripping of a glass-ball of 80µm diameter is the following:

$$\begin{cases} X_1(k+1) = A_a \cdot X_1(k) + B_a \cdot V_{in}(k) \\ x_a(k) = C_a \cdot X_1(k) \end{cases}, \tag{6.18}$$

with $A_a = \begin{pmatrix} 1.68515803958439 & -0.963064501560099 \\ 1 & 0 \end{pmatrix}$ $B_a = \begin{pmatrix} 0.5 \\ 0 \end{pmatrix}$.
$C_a = \begin{pmatrix} 0.176864041916997 & -0.0773147100025847 \end{pmatrix}$ $D_a = 0$

Sensing System Modelling

Likewise, the state-space representation of the sensing system is defined as follows:

$$\begin{cases} X_2(k+1) = A_c \cdot X_2(k) + B_c \cdot F_c(k) \\ x_b(k) = C_c \cdot X_2(k) \end{cases}, \tag{6.19}$$

with $A_c = \begin{pmatrix} 1.42560829239571 & -0.967281866841811 \\ 1 & 0 \end{pmatrix}$ $B_c = \begin{pmatrix} 0.25 \\ 0 \end{pmatrix}$.
$C_c = \begin{pmatrix} 0.0605315717144615 & -0.299232450813051 \end{pmatrix}$ $D_c = 0$

Coupled System Modeling

Considering the assumption that the objects being manipulated by the micro-gripper behave like a spring having a stiffness k_0, it is possible to characterize the effort of the gripping force F_c by the following equation: $F_c = k_0 \cdot (x_a - x_b) = k_2 \cdot x_b \Rightarrow x_b = k_0 / (k_0 + k_2) \cdot x_a$, with k_2 being the stiffness of the sensing system.

Now we can group both models (Eq. 6.18 and 6.19) into a generalized model defining the FT-G100

micro-gripper in the gripping phase (actuation and sensing), as follows:

$$\begin{cases} \begin{bmatrix} X_1(k+1) \\ X_2(k+1) \end{bmatrix} = \begin{bmatrix} A_a & 0 \\ B_c \cdot k_k \cdot C_a & A_c \end{bmatrix} \cdot \begin{bmatrix} X_1(k) \\ X_2(k) \end{bmatrix} + \begin{bmatrix} B_a \\ 0 \end{bmatrix} \cdot V_{in}(k) \\ V_{out}(k) = \begin{bmatrix} 0 & r \cdot C_c \end{bmatrix} \cdot \begin{bmatrix} X_1(k) \\ X_2(k) \end{bmatrix} \end{cases} \equiv \begin{cases} X(k+1) = A \cdot X(k) + B.V_{in}(k) \\ V_{out}(k) = C \cdot X(k) \end{cases}, \tag{6.20}$$

with $k_k = (k_0 \cdot k_2)/(k_0 + k_2)$.

6.2.2 Kalman Filtering

Kalman filtering [49] uses a state-space representation comprising a stochastic part modeled in the form of a state noise $w(k)$ and a measure noise $v(k)$. Considering Eq. 6.20, the Kalman model of the microgripper FT-G100 is:

$$\begin{cases} X(k+1) = A \cdot X(k) + B.V_{in}(k) + M \cdot w(k) \\ V_{out}(k) = C \cdot X(k) + v(k) \end{cases}, \tag{6.21}$$

where $X(k) \in \Re^{4\times1}$: system state vector at time $t = k \cdot T_e$, T_e is the sampling period.

$A \in \Re^{4\times4}$: transition state matrix.

$B \in \Re^{4\times1}$: control input matrix.

$V_{in} \in \Re^{1\times1}$: known and deterministic input vector.

$w(k) \in \Re^{4\times1}$: vector of random unknown signals coming disrupting the equation state of the system through an entry matrix $M_{1\times4}$.

$V_{out} \in \Re^{1\times1}$: vector of measures (output).

$C \in \Re^{1\times4}$: observation output matrix.

$v(k) \in \Re^{1\times1}$: vector of random signal disrupting the system measures.

Fig. 6.19 summarizes the different equations governing the evolution of the Kalman algorithm, implemented on the micro-gripper FT-G100 (Eq. 6.21). The algorithm requires at the starting step the determination of the initial states $P(0/0)$ and $\hat{X}(0/0)$. A zero value has been assigned to these parameters due to the initial state of the system.

Time Update ("Predict"): time=k	Measurement Update ("Correct"): time=k+1
(1) Project the state ahead for time k+1: $\hat{X}(k+1/k) = A \cdot \hat{X}(k/k) + B \cdot V_{in}(k)$	**(5)** Update the error covariance at time k+1: $P(k+1/k+1) = [I_4 - K_e(k+1) \cdot C].P(k+1/k)$
(2) Project the error covariance ahead for time k+1: $P(k+1/k) = A \cdot P(k/k) \cdot A^T + M \cdot W \cdot M^T$	**(4)** Update state estimate with the innovation at time k+1: $\hat{X}(k+1/k+1) = \hat{X}(k+1/k) + K_e(k+1).\tilde{V}_{out}(k+1)$
	(3) Compute the Kalman gain at time k+1: $K_e(k+1) = P(k+1/k) \cdot C^T \cdot [C \cdot P(k+1/k) \cdot C^T + V]^{-1}$

FIGURE 6.19 – Kalman recursive algorithm.

It is important to note that the gain of Kalman K_e rapidly converges to a constant value. It is therefore possible to make an offline calculation of the gain and exploit its permanent regime throughout the filtering algorithm. In this case, only equations (1) and (4) of the Fig. 6.19 are used.

6.2.3 Force Control of the FT-G100 Micro-Gripper

The LQG belongs to the category of controllers called "optimal". It is based on the minimization of an energy criterion in order to achieve a compromise between the performance of the controlled system and the energy consumed. The LQG controller and the Kalman filter can be independently calculated according to a principle of separation. The synthesis of the LQG for controlling the FT-G100 micro-gripper is beyond the scope of this thesis. We limit ourselves to the utilization of the final equation provided in [48]:

$$\begin{cases} \begin{bmatrix} X(k+1) \\ \varepsilon_x(k+1) \end{bmatrix} = \begin{bmatrix} A - B \cdot K_c & B \cdot K_c \\ 0 & A - K_e \cdot C \end{bmatrix} \cdot \begin{bmatrix} X(k) \\ \varepsilon_x(k) \end{bmatrix} + \begin{bmatrix} B \cdot L \\ 0 \end{bmatrix} \cdot F_{cc}(k) \\ \varepsilon_y(k) = \begin{bmatrix} 0 & C \end{bmatrix} \cdot \begin{bmatrix} X(k) \\ \varepsilon_x(k) \end{bmatrix} \end{cases}, \tag{6.22}$$

where ε_y is the estimation error of the output, F_{cc} is the force setpoint, and $\varepsilon_x(k) = X(k) - \hat{X}(k/k)$.

The general control scheme grouping the LQG controller and Kalman filter is depicted in Fig. 6.20.

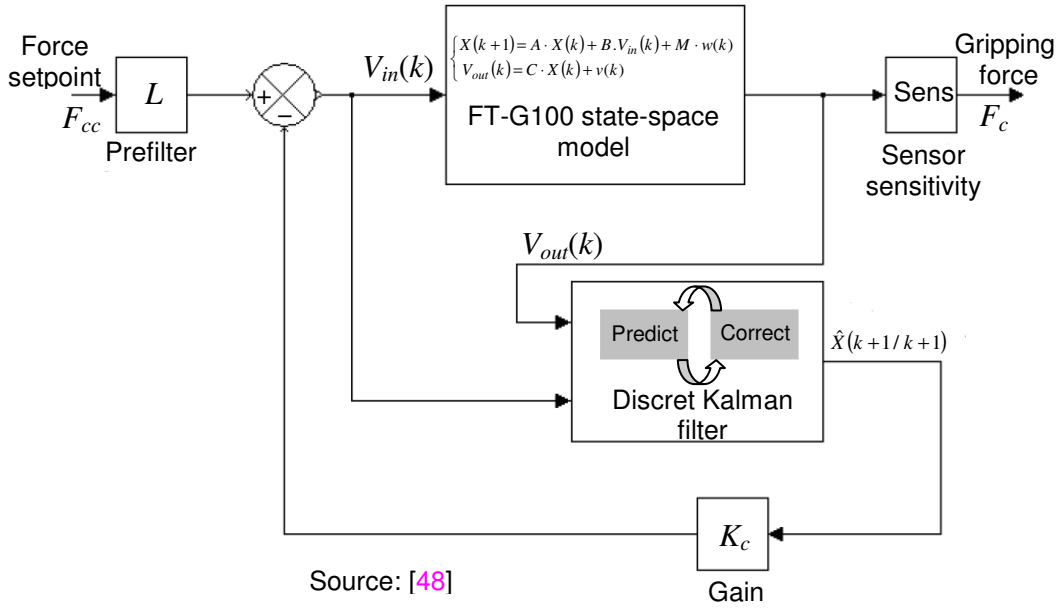


FIGURE 6.20 – General scheme of the LQG controller with Kalman filter.

6.2.4 Hardware integration of the LQG controller with Kalman Filter

The Matlab model corresponding to Fig. 6.20 was kindly provided by Boudaoud [48] (Appendix D). It runs on dSPACE with a 200 KHz sampling frequency. We have translated it to a synthesizable Verilog code (Appendix D) through the methodology described in Fig. 6.21. The latter can be perceived as a standard methodology allowing the translation of LTI Matlab models to synthesizable HDL (Verilog/VHDL) code. The steps of Fig. 6.21 are successively commented as follows.

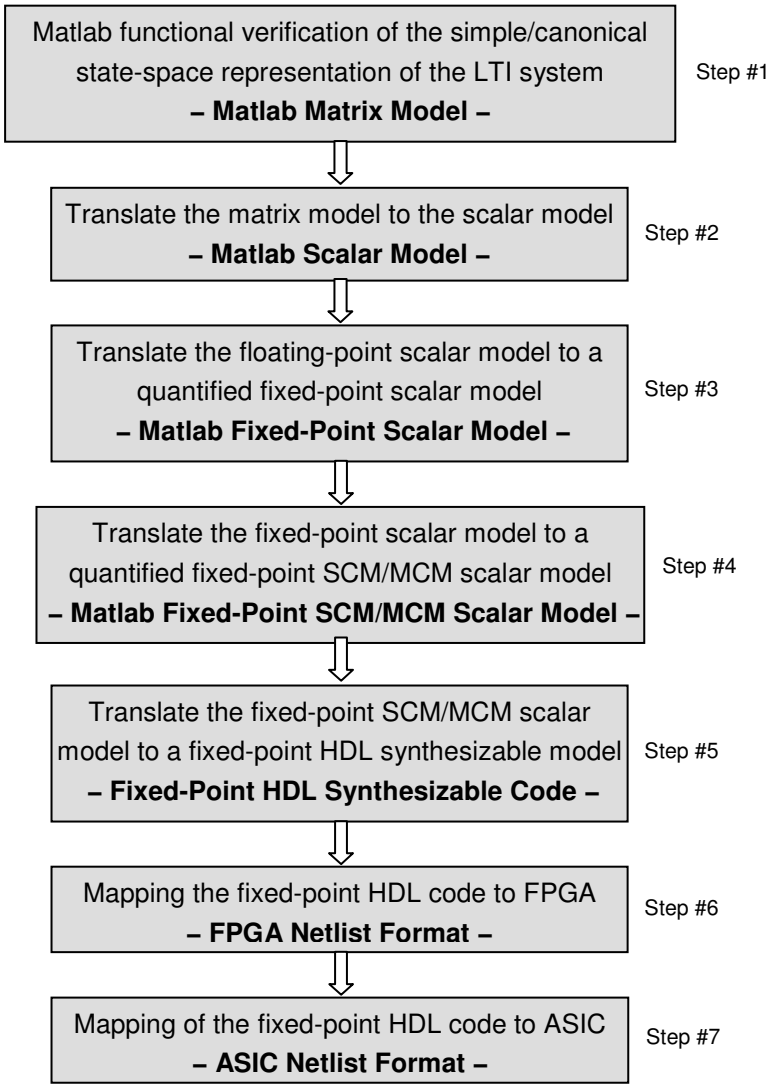


FIGURE 6.21 – Standard methodology for an optimized hardware integration of LTI systems: from Matlab functional model to HDL synthesizable code.

Step #1

Before performing the functional verification, we must determine which state-space form, canonical or simple, is more optimized (contains more zeros and ones elements). Because using the Matlab "canon" function, we have noticed that for low order systems (≤4) the canonical form is not necessarily better than the simple form. To decide which form to use, we have to count the number of zeros and ones elements in each representation. In our case, we opted for the canonical form though it does not offer a clear advantage over the simple form (to be in total conformity with the initial Matlab model for ulterior comparisons). Then, before pursuing the process, we must be sure that the Matlab description is 100% functionally correct to avoid very time-consuming backward verifications. In our case, the Matlab model of Fig. 6.20 has undergone severe verifications. For instance, Fig. 6.22 describes the noisy and filtered force (F_c) of the actuated arm.

Step #2

In this step, the Matlab code is decompressed: the linear matrix operations (+, –, ×) are replaced by

scalar operations. The higher the order of the system, the more difficult the translation is. In our case, the order of the system is 4. The decompression process results in 29 multiplications and 23 additions. The decompression operation is error prone; therefore it should be carefully performed or automated if possible. A functional verification of the scalar model is necessary to move to the next step.

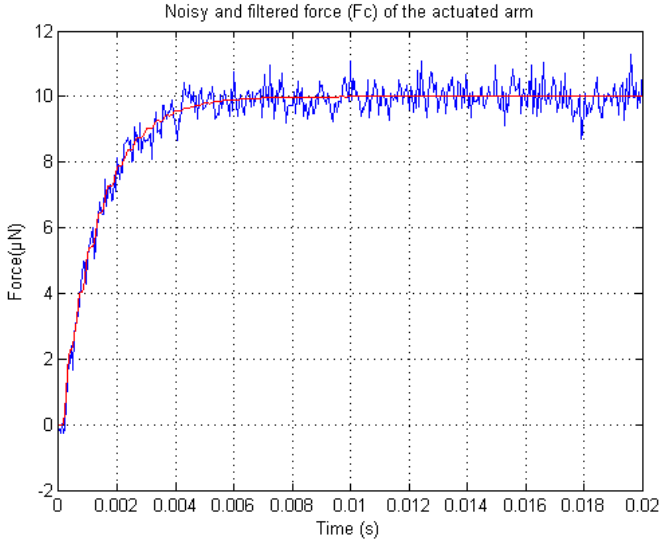


FIGURE 6.22 – The noisy and filtered force (F_c) of the actuated arm.

Step #3

This is a crucial step. Up to now, we have been using the floating-point representation. To move to the fixed-point representation, we need to determine the dynamic range of all the variables involved. In our case, this operation has led to an integer part of 5 bits and a fractional part of 16 bits ($Q_{5.16}$). With 21-bit word length, the difference between the floating-point and the fixed-point filtered force (F_c) is less than 7×10^{-5} µN as indicated by Fig. 6.23. Improving the precision ($<7\times10^{-5}$ µN), will result in a larger controller (from a hardware point of view).

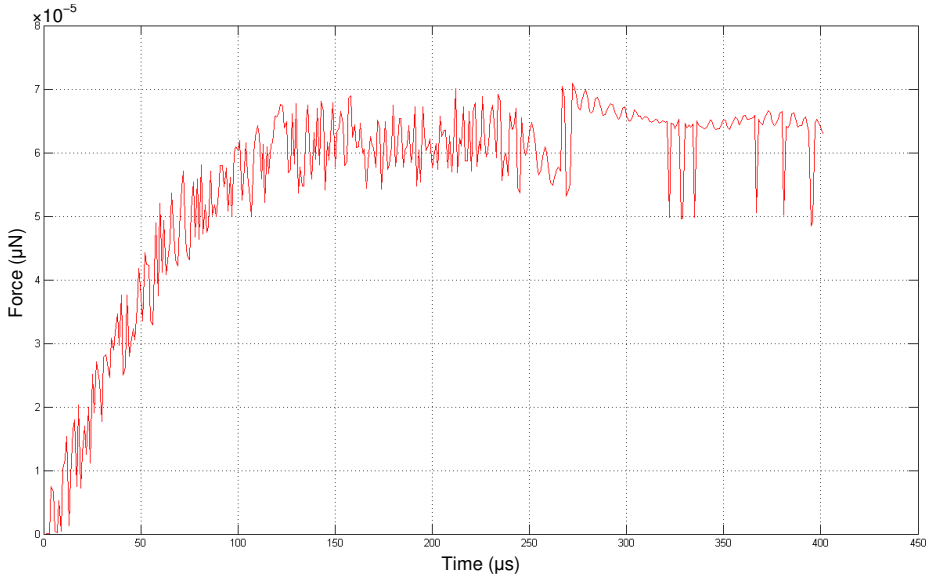


FIGURE 6.23 – The difference between the floating-point and the fixed-point filtered force (F_c).

Step #4

In this step, the scalar multiplication (×) is replaced by SCM/MCM (additions, subtractions, and shifts) and the common sub-expressions are eliminated using MCM. In our case, the replacement of the scalar multiplication by SCM/MCM results in 101 additions (Appendix D). The 29 multiplications of step #3 would have required $\lceil(21/2)-1\rceil\times 29 = 290$ additions in case the optimal multiplier (Eq. 5.22) was used. Adding the 29 additions, the total number of additions corresponding to step #3 is 290+23=313. A saving of (313-101)/313=68% is therefore induced. In fact the saving in logic resources is more important since the 29 multipliers require a recoding logic (PPG).

The Matlab simulation results of this step must be exactly identical to the ones of step #3.

Step #5

In this step, the Matlab model is simply translated into a synthesizable HDL code (Verilog in our case). The code should be compliant to the standard design-reuse methodology [6] to make it technology-independant. This allows the code to be mapped on FPGA and ASIC as well. Matlab/Simulink simulations are necessary to validate the code. Simulation results should be exactly identical to the ones of step #3.

For the time being, this step is under progress. The Matlab code of step #4 has been systematically translated into a Verilog code, and the necessary verifications are being performed.

Step #6

The synthesizable HDL code is mapped to a FPGA circuit and physical tests are performed to validate the HDL code.

Step #7

Since the HDL code is technology-independent (design-reuse methodology), it can also be mapped to an ASIC using a standard-cell-library of a given foundry (e.g. TSMC 0.18 µm CMOS technology).

The hardware integration flow of LTI FWL controllers (Fig. 6.21) is very time consuming and error-prone, particularly the crucial steps #3 and #4. Consequently, the automation of the design flow becomes necessary, especially for high order systems.

6.3 Conclusion

Despite the large popularity of PID controller, little attention has been paid to its optimization, either for ASIC or for FPGA integration. To break down this paradoxical situation, a series of high-speed and low-power PIDs, dedicated to embedded applications were proposed. They are based on two discrete forms of PID algorithm: the incremental form and the commercial form, both with constant and time-varying coefficients. The work focused more particularly on the commercial form with varying coefficients (LTV) as it is the most used in industry due to the higher control-quality provided. Two types of optimizations were carried out: architectural and algorithmic optimizations. The former is a macro-level optimization, based on an efficient partitioning of PID discrete-equations, considering the double MAC (DMAC) as the main building block of PID architecture. An optimized version of DMAC was developed (ODMAC) for less hardware resource occupation. As for the micro-level optimization (inner optimization of ODMAC), three multiplication algorithms were experienced: BMA, MBMA, and a

new general and recursive version of MBMA called RMRMA. In addition, some low-power design techniques were incorporated, such as: sleep mode, and step-by-step sign-propagation technique.

The implementation results of PID based upon these three algorithms yielded to gradual improvements with a clear superiority over results presented in [21]. For instance, concerning PID1_2 and PID1_4, savings of 177%, 23%, and 36%, and savings of 284%, 14%, and 26% are obtained in control-rate, power consumption, and total gate count, respectively. Additionally, analytical scaling-complexity evaluations with respect to the couple (n,r), confirmed also by software simulations, revealed useful information which is summarized as follows:

- PIDX_n/2 is the fastest PID that yields to the highest control-rate (30 MHz for PID2_8 mapped on Virtex6, with (n,r)=(16,8));
- PIDX_1 is the most power efficient PID when speed is not a concern;
- PIDX_n and PIDX_n/2 are the most efficient PIDs when both high control-rate and low-power dissipation are required.

RMRMA is a double recursive algorithm (Eq. 6.13), which is a particular case of Eq. 5.22. The use of the triple recursive version (Eq. 5.22) instead Eq. 6.13 will produce more efficient PIDs.

Further extension to the present work is to apply the same (or appropriate) partitioning in conjunction with RMRMA algorithm (or Eq. 5.22) to the set of recurrent equations of an arbitrary number of multi-loop PID controllers taken as a whole.

The LTI option has been addressed through the implementation of an LQG controller with Kalman filtering. A methodology for converting a Matlab code to a synthtizable HDL code has been proposed and applied. Though it has contributed to a drastic reduction of logic resources (68% saving) due to the utilization of our SCM/MCM algorithms, the methodology is very time-consuming and error-prone, especially for high-order systems. The automation stands therefore as the sole practical solution, however, it requires a considerable effort (code translations).

Bibliography

[1] K. Åström, T. Hägglund, "PID Controllers: Theory, Design, and Tuning," by the Instrument Society of America, Research Triangle Park, NC, USA, 2nd Edition, ISBN: 1-55617-516-7, Copyright 1995.

[2] D. Xue et al, "Linear Feedback Control," by the Society for Industrial and Applied mathematics, Copyright 2007. Available: http://www.siam.org/books/dc14/DC14Sample.pdf

[3] S. Xiaoyin et al, "A New Motion Control Hardware Architecture with FPGA based IC-Design for Robotic Manipulators," Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 3520-3525, Orlando, Florida, May 2006.

[4] J.S. Kim, H.W. Jeon, and S. Jeung, "Hardware Implementation of Nonlinear PID Controller with FPGA based on Floating Point Operation for 6-DOF Manipulator Robot Arm," Proceedings of the IEEE International Conference on Control Automation and Systems (ICROS), pp. 1066-1071, Seoul, Korea, October 2007.

[5] L. Qu, Y. Huang, and L. Ling, "Design and Implementation of Intelligent PID Controller based on FPGA," Proceedings of the IEEE International Conference on Natural Computation (ICNC), pp. 511-515, 2008.

[6] M. Keating & P. Bricaud, "Reuse Methodology Manual for System on a Chip Designs," by the Kluwer Academic Publishers, NY, USA, 3rd Edition, ISBN: 1-4020-7141-8, Copyright 2002.

[7] Reports of the International Technology Roadmap for Semiconductors (ITRS), 2007 & 2008. Available: www.itrs.net/reports.html

[8] T. Hilaire, P. Chevrel, and J.F. Whidborne, "A Unifying Framework for Finite Word Length Realizations," IEEE Trans. on Circuits and Systems, Vol. 54, N° 8,, August 2007.

[9] T. Hilaire, D. Ménard, and O. Sentieys, "Bit Accurate Roundoff Noise Analysis of Fixed-Point Linear Controllers," Proceedings of the IEEE International Conference on Computer-Aided Control Systems (CACSD), pp. 607-612, 2008.

[10] S. Gretlein et al, "DSPs, Microprocessors and FPGAs in Control," the Magazine of Record for the Embedded Computing Industry (RTC Magazine), March 2006.

[11] E. Manmasson et *al.*, "FPGA in Industrial Control Applications," IEEE Trans. on Industrial Informatics, vol. 7, N° 2, May 2011.

[12] S. Chander, P. Agarwal, and I. Gupta, " FPGA-based PID Controller for DC-DC Converter," Proceedings of the IEEE Joint International Conference on Power Electronics, Drives and Energy Systems (PEDES), India, 2010.

[13] S. Yang et al, "The IP Core Design of PID Controller based on SOPC," Proceedings of the IEEE International Conference on Intelligent Control and Information Processing, pp. 363-366, Dalian, China, August 2010.

[14] J. Lazaro et al, "Simulink/Modelsim Simulable VHDL PID Core for Industrial SoPC Multiaxis Controllers," Proceedings of the IEEE 32nd Annual Conference on Industrial Electronics (IECON), pp. 3007-3011, 2006.

[15] F. Fons, M. Fons, and E. Canto, "Custom-Made Design of a Digital PID Control System," Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing(ICASSP), Vol. 3, pp. 1020-1023, 2006.

[16] B.V. Sreenivasappa and R.Y. Udaykumar, " Design and Implementation of FPGA based Low Power Digital PID Controllers," Proceedings of the IEEE International Conference on Industrial and Information Systems (ICIIS), pp. 568-573, 2009.

[17] J. Lima et al, "A Methodology to Design FPGA-based PID Controllers," Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pp. 2577-2583, Taipei, Taiwan, October 2006.

[18] I. Urriza et al, "Word Length Selection Method based on Mixed Simulation for Digital PID Controllers Implemented in FPGA," Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE), pp. 1965-1970, 2008.

[19] W. Zhao et al, "FPGA Implementation of Closed-Loop Control Systems for Small-Scale Robot," Proceedings of the IEEE 12th International Conference on Advanced Robotics (ICAR), pp. 70-77, 2005.

[20] L. Samet et al, "A Digital PID Controller for Real-Time and Multi-Loop Control: a Comparative Study," Proceedings of the IEEE International Conference on Electronics, Circuits, and Systems (ICECS), vol. 1, pp. 291-296, 1998.

[21] Y. Fong, M. Moallem, and W. Wang, "Design and Implementation of Modular FPGA-Based PID Controllers," IEEE Trans. on Industrial Electronics, Vol. 54, N° 4, pp. 1898-1906, August 2007.

[22] B. Wittenmark, K. J. Astrom, and K.-E. Arzenin "Computer control: An overview," Technical Report of Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden, Apr. 2003.

Available: www.control.lth.se/kursdr/ifac.pdf

[23] A. D. Booth, "A Signed Binary Multiplication Te:chnique," Quarterly J. Mech. Appl. Math., Vol. 4, part 2, pp. 236-240,1951.

[24] O.L. MacSorley, "High-Speed Arithmetic in Binary Computers," Proceedings of the IRE, Vol. 49(1), pp. 67-91, January 1961.

[25] A.K. Oudjida, N. Chaillet, A. Liacha, and M.L. Berrandjia, "A New Recursive Multibit Recoding Algorithm for High-Speed and Low-Power Multiplier," Journal of Low Power Electronics (JOLPE), vol. 8, N° 5, pp. 1-16, December 2012, American Scientific Publishers (ASP), USA.

[26] A.K. Oudjida et *al.*, "High-Speed and Low-Power PID Structures for Embedded Applications," Proceedings of the 21th edition of the International Workshop on Power and Timing Modeling, Optimization and Simulation PATMOS, LNCS 6951, pp. 257-266, Springer-Verlag Editor. Madrid, Spain, September 26-29, 2011.

[27] Y.H. Seo, and D.W. Kim, "A New VLSI Architecture of Parallel Multiplirer-Accumulator Based on Radix-2 Modified Booth Algorithm," IEEE Trans. on VLSI Systems, vol. 18, N° 2, Feb. 2010.

[28] L.P. Rubinfield, "A Proof of the Modified Booth Algorithm for Multiplication," IEEE Trans. On Computers, C-24, (10), pp. 1014-1015, 1975.

[29] H. Sam, and A. Gupta, "A Generalized Multibit Recoding of Two's Complement Binary Numbers and its Proof with Application in Multiplier Implementation," IEEE Trans. on Computers, vol. 39, N° 8, August 1990.

[30] F. Lamberti, "Reducing the Computation Time in (Short Bit-Width) Two's Complement Multiplier," IEEE Trans. on Computers, vol. 60, N° 2, pp. 148-156, February 2011.

[31] S.R. Kuang, J.P. Wang, and C.Y. Guo, "Modified Booth Multipliers with a Regular Partial Product Array," IEEE Trans. on Circuit and Systems II, Express Brief, vol. 56, N° 5, May 2009.

[32] J.Y. Kang, J.L. Gaudiot, “A Simple High-Speed Multiplier Design,” IEEE Trans. on Computers, vol. 55, N° 10, Oct. 2006.

[33] D. Crookes and M. Jiang, “Using Signed Digit Arithmetic for Low-Power Multiplication,” Electronics Letters, vol. 43, N° 11, may 2007.

[34] P.M. Seidel, L. D. McFearin, and D.W. Matula, “Secondary Radix Recodings for Higher Radix Multipliers,” IEEE Trans. on Computers, vol. 54, N°2, February 2005.

[35] R.C. North, and W.H. Ku, “β-Bit Serial/Parallel Multipliers,” Journal of VLSI Signal Processing, Kluwer Academic Publishers, Boston, vol. 2, pp. 219-233, 1991.

[36] D.A. Henlin, M.T. Fertsch, M. Mazin, and E.T. Lewis, “A 16 bit x 16 bit Pipelined Multiplier Marcrocell,” 1EEE Journal of Solid-State Circuits, vol. SC-20, no. 2, pp. 542-547, 1985.

[37] J.S. Kelly et al, “Design and Implementation of Digital Controllers for Smart Structures Using Field Programmable Gate Arrays,” Smart Material Structure Journal, PII: S0964-1726 (97) 87085-1, pp. 559-572, Printed in the UK, 1997.

[38] L. Shang, A.S. Kaviani, and K. Bathala, “Dynamic Power Consumption in Virtex-II FPGA Family,” Proceedings of FPGA Conference, pp. 157-164, Monterey, California, USA, February 2002.

[39] Xilinx Inc., “Virtex6 FPGA: Satisfying the Insatiable Demand for Higher Bandwidth,” PN 2403, Printed in the USA, Copyright 2009.

www.xilinx.com/publications/prod_mktg/Virtex6_Product_Brief.pdf

[40] M. Gevers and G. Li, “Parametrizations in Control, Estimation and Filtering Probems,” Springer-Verlag, 1993.

[41] T. Hilaire and P. Chevrel, “Sensitivity-based pole and input-output errors of linear filters as indicators of the implementation deterioration in fixed-point context,” EURASIP Journal on Advances in Signal Processing, vol. special issue on Quantization of VLSI Digital Signal Processing Systems, January 2011.

[42] B. Lopez, T. Hilaire and L.S. Didier, “Sum-of-products Evaluation Schemes with Fixed-Point arithmetic, and their application to IIR filter implementation,” Proceedings of the IEEE International Conference on Design and Architecture for Signal and Image Processing (DASIP), Karlsruhe, Germany, Oct. 2012.

[43] M. Petko and G. Karpiel, “Semi-automatic implementation of control algorithms in ASIC/FPGA,” Proceedings of Emerging Technologies and Factory Automation Conference (ETFA '03), vol. 1, pp. 427- 433. Sept. 2003.

[44] S.K. Rao and T. Kailath, “Regular Iterative Algorithms and their Implementation on Processor Arrays,” Proceeding of the IEEE, vol. 76, pp. 259-269, Mar. 1988.

[45] G. Hoover et al, “Towards Understanding Architectural Tradeoffs in Mems Closed-Loop Feedback Control,” Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES’07), pp. 95-102, Salzburg, Austria, Sep. 30-Oct. 3, 2007.

[46] R. Casanova et al, “Integration of the Control Electronics for a mm^3-sized Autonomous Microrobot into a Single Chip,” Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 3007-3012, Kobe, Japan, May 12-17, 2009.

[47] M. Boudaoud, Y. Haddab, and Y. Le Gorrec, "Modelling of a MEMS Micro-Gripper: Application to Dexterous Micromanipulation," Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5634-5639, Taipei, Taïwan, October 18-22, 2010.

[48] M. Boudaoud, "Commande en Effort d'une Micropince en Actionnenement Electrostatique," Master Thesis, AS2M Department, FEMTO-ST, Besançon, July 2009.

[49] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," Internal Report, Department of Computer science, UNC-Chapel Hill, TR 95-041, July 24, 2006.

Chapter 7

General Conclusion

Chapter 7

General Conclusion

Let us restart from the introduction:

The objective of the thesis is the development of a new binary arithmetic adapted to the FWL problem, enabling an easy generation of minimum word-size controller-structures with acceptable control performances.

In this thesis, a new radix-2^r arithmetic has been developed. It basically relies on the multibit recoding algorithm which was introduced in 1990 but abandoned due to its inability to cope with high-radix arithmetic. We have solved its radix lock and taken, as a result, full advantage of its high potential in producing efficient multiply arithmetic, which is the real engine of LTI/LTV controllers. In addition, because highly-scalable (r), the new radix-2^r arithmetic is well adapted to the FWL effect. Given a bit-size N of the word that ensures acceptable control performance, the new arithmetic is capable of generating the optimal realization of the FWL controller without degrading the control performances.

The "optimality" of the realization comes from the multipliers themselves. For each value of N, we provide the optimal space/time partitioning of the multiplier that leads ineluctably to the optimal implementation of the controller structure. This is valid for both types of multiplier: variable multipliers and constant multipliers.

The variable and the constant multipliers have been used to build up two FWL structures: an LTV (PID) and LTI (LQG + Kalman filter) controllers, respectively. The PID shows a high superiority over its existing counterparts, either in speed, power, or area. As for the LQG controller, a drastic reduction in logic resources is achieved, which means also much higher speed and less power consumption.

We have proved via the implementation of the PID and LQG controllers, the high capabilities of the new arithmetic to respond to the needs of MEMS applications in speed, power, and especially in scalability for an easy adjustment of the control performances.

7.1 Major Contributions

Breaking the high-radix deadlock of the multibit recoding algorithm has enabled to achieve some important results in binary arithmetic, notably:

- The development of a linear $O(N)$ and fully predictable heuristic for the multiplication by a constant (SCM/MCM). The major advantage over the existing heuristics is that no limitation exists on the word-size (N) of the LTI controller.
- The development of an optimal algorithm for the multiplication by a variable (MV). According to the word-size of the LTV controller, there exists an optimal multiplier that leads to an optimal implementation of the controller.

- The development of several multi-precision multipliers (MPM) allowing a number of LTV controllers with different word-sizes to share the same multiplication array. This contributes to increase the throughput while decreasing the power consumption.
- The determination of the currently best known analytically-proved bounds for the multiplication problem (SCM, MCM, and MV).

7.2 Current Limitations

Because of a limited time, a number of improvements could not be undertaken, namely:

- For the multiplication by a constant, we have proposed two heuristics: RADIX-2^r and R3. While RADIX-2^r is fully predictable (*Upb*, *Ath*, *Avg*), R3 is not predictable. Nevertheless, since we have proved that R3 is better than RADIX-2^r, R3 can be bounded by RADIX-2^r metrics.
- Lefèvre's algorithm for the multiplication by a constant relies on CSD recoding. Since R3 is much better than CSD, combining Lefèvre's algorithm with R3 will produce better results.
- To determine the optimal multiplier (Eq. 5.22), we have employed a mathematical formalism (Eq. 5.21) including only four low radix recodings (Seidel radix-2^8 and radix-2^5, McSorley radix-2^2, and Booth radix-2^1). These are the only low-radix algorithms that we are aware of. In case another low-radix recoding exists, it can be inserted into Eq. 5.21 to look for another optimum that *could be* better than Eq. 5.22.
- We still do not know whether RADIX-2^r is a canonical recoding (minimal and unique) even after the simplifications given by Eq. 3.45 and 3.46. A mathematical proof is missing. The proof, if any, will let to declare RADIX-2^r as a generalization of CSD (i.e. CSD=RADIX-2^1).
- While the theoretical concept for MV has been validated using FPGA as a preliminary step, an ASIC implementation based on a standard-cell library is necessary for an ultimate validation of the whole optimization work.

7.3 Perspectives

The new arithmetic is a complete package, ready for integration. While it is dedicated to MEMS applications in general, a direct utilization for the AS2M department would be in the control of microgrippers or the integration of smart sensors.

Translating FWL controllers from high-level specifications in C or Matlab code to a synthesizable HDL code requires a fully automated design flow. Such a flow is already under progress in a French ANR project called DEFIS (Design of Fixed-Point Embedded Systems) [1]. It would be useful to explore the possibility to integrate some of the results of this thesis into the DEFIS flow, especially the predictable SCM/MCM heuristic and the optimal MV algorithm.

The "predictability" is a highly sought feature for CAD synthesis tools. It enables the synthesis tools to rapidly satisfy designer requirements in speed and area, avoiding therefore unnecessary feedbacks. Hence, the new SCM/MCM, MV, and MPM algorithms can be incorporated into synthesis tools to produce predictable IPs.

Finally, the new arithmetic can be advantageously applied to other numeric areas, such as DSP, image processing, telecommunication, and encryption. An idea is to apply the new SCM/MCM and MV algorithms in RSA encryption to target long encryption keys (more than 4096 bits).

Bibliography

[1] D. Menard et al., “Design of Fixed-Point Embedded Systems (DEFIS) French ANR Project,” Proceedings of the IEEE International Conference on Design and Architecture for Signal and Image Processing (DASIP), Karlsruhe, Germany, Oct. 2012.

Appendices

Appendix A

Proofs of Theorems 4.3 and 4.4

Proof of Theorem 4.3

Initially, the multiplier Y is an N bit string. But to comply with the requirement of the multibit recoding algorithm, we need to add a zero bit (y_{-1}) to the less significant side of Y. Thus, the total size becomes N+1. Y is a two's complement number. It is written as follows:

$$Y = y_{-1}\ y_0\ y_1\ y_2 \cdots y_{N-2}\ y_{N-1} \text{ , with } y_{-1}=0$$

$$= y_{-1} + 2^0 y_0 + 2^1 y_1 + 2^2 y_2 + \cdots + 2^{N-2} y_{N-2} - 2^{N-1} y_{N-1}$$

$$= -y_{N-1}\, 2^{N-1} + \sum_{j=-1}^{N-2} y_j\, 2^j$$

In the multibit recoding algorithm, the multiplier Y is split into N/r two's complement slices (Q_j), each of r+1 bit length. Two contiguous slices (Q_j with Q_{j-1}, and Q_j with Q_{j+1}) have one overlapping bit in common. Thus Y becomes:

$$Y = \sum_{j=0}^{\frac{N}{r}-1} \left(y_{rj-1} + 2^0 y_{rj} + 2^1 y_{rj+1} + 2^2 y_{rj+2} + \cdots + 2^{r-2} y_{rj+r-2} - 2^{r-1} y_{rj+r-1} \right) 2^{rj} = \sum_{j=0}^{\frac{N}{r}-1} Q_j\, 2^{rj}$$

In fact the Q_j term is no more than a two's complement representation of r+1 bit string which can be split in its turn into r/s two's complement overlapping slices (P_{ji}), each of s+1 bit length. Thus Y becomes:

$$\begin{aligned} Y = \sum_{j=0}^{\frac{N}{r}-1} \Big[& \left(y_{rj-1} + 2^0 y_{rj} + 2^1 y_{rj+1} + 2^2 y_{rj+2} + \cdots + 2^{s-2} y_{rj+s-2} - 2^{s-1} y_{rj+s-1} \right) 2^0 \quad + \\ & \left(y_{rj+s-1} + 2^0 y_{rj+s} + 2^1 . y_{rj+s+1} + 2^2 y_{rj+s+2} + \cdots + 2^{s-2} y_{rj+2s-2} - 2^{s-1} y_{rj+2s-1} \right) 2^s \quad + \\ & \vdots \\ & \left(y_{rj+r-1-2s} + 2^0 y_{rj+r-2s} + 2^1 . y_{rj+r-2s+1} + 2^2 y_{rj+r-2s+2} + \cdots + 2^{s-2} y_{rj+r-s-2} - 2^{s-1} y_{rj+r-1-s} \right) 2^{s\left(\frac{r}{s}-2\right)} \quad + \\ & \left(y_{rj+r-1-s} + 2^0 y_{rj+r-s} + 2^1 . y_{rj+r+s+1} + 2^2 y_{rj+r+s+2} + \cdots + 2^{s-2} y_{rj+r-2} - 2^{s-1} y_{rj+r-1} \right) 2^{s\left(\frac{r}{s}-1\right)} \Big] \end{aligned}$$

$$= \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{s}-1} \left(y_{rj-1+si} + 2^0\, y_{rj+si} + 2^1\, y_{rj+1+si} + 2^2\, y_{rj+2+si} + \cdots + 2^{s-2}\, y_{rj+s-2+si} - 2^{s-1}.y_{rj+s-1+si} \right) 2^{si} \right] 2^{rj}$$

$$= \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{s}-1} P_{ji}\, 2^{si} \right] 2^{rj}$$

A synoptic scheme is depicted in Fig. A.1 to illustrate the use of Th 4.3 in the partitioning of a 16-bit Y operand.

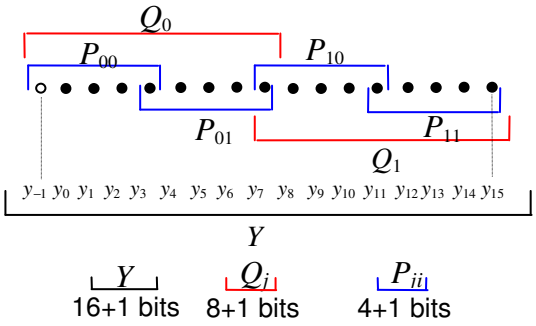


FIGURE A.1 Partitioning of a 16-bit Y operand with r=8 and s=4

Proof of Theorem 4.4

Likewise, Y can also be rewritten as follows:

$$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{s+t}-1} \left[\left(y_{rj-1+(s+t)i} + 2^0\, y_{rj+(s+t)i} + 2^1.y_{rj+1+(s+t)i} + \cdots + 2^{s-2}\, y_{rj+s-2+(s+t)i} - 2^{s-1}\, y_{rj+s-1+(s+t)i} \right) + \right.\right.$$

$$\left.\left. \left(y_{rj+s-1+(s+t)i} + 2^0\, y_{rj+s+(s+t)i} + 2^1.y_{rj+1+s+(s+t)i} + \cdots + 2^{t-2}\, y_{rj+r-2+(s+t)i} - 2^{t-1}\, y_{rj+r-1+(s+t)i} \right) 2^s \right] 2^{(s+t)i} \right] 2^{rj}$$

$$= \sum_{j}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{s+t}-1} \left[P_{ji} + T_{ji}\, 2^s \right] 2^{(s+t)i} \right] 2^{rj}$$

A synoptic scheme is depicted in Fig. A.2 to illustrate the use of Th. 4.4 in the partitioning of a 16-bit Y operand.

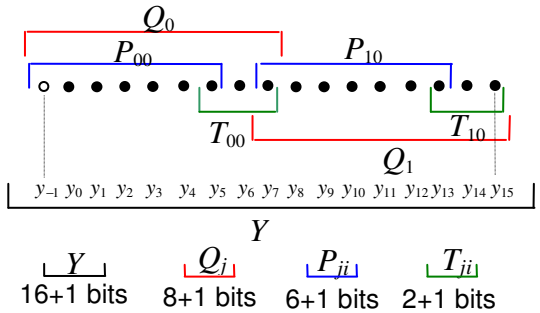


FIGURE A.2 Partitioning of a 16-bit Y operand with r=8, s=6 and t=2

Appendix B

A Series of New High-Radix Recodings

New High-Speed and Low-Power Radix-2^r Multiplication Algorithms

A.K. Oudjida, A. Liacha, M.L. Berrandjia
Microelectronics and Nanotechnology Division
Centre de Développement des Technologies Avancées
Algiers, Algeria
a_oudjida@cdta.dz

N. Chaillet
AS2M Department
FEMTO-ST Institute
Besançon, France
nicolas.chaillet@femto-st.fr

***Abstract*— In this paper, a new recursive multibit recoding multiplication algorithm is introduced. It provides a general space-time partitioning of the multiplication problem that not only enables a drastic reduction of the number of partial products (*N/r*), but also eliminates the need of pre-computing odd multiples of the multiplicand in higher radix (*r*≥3) multiplication. Based on a mathematical proof that any higher radix-2^r can be recursively derived from a combination of two or a number of lower radices, a series of generalized radix-2^r multipliers are generated by means of primary radices: 2^1 , 2^2, 2^5, and 2^8. A variety of higher-radix (2^3-2^{32}) two's complement 64x64 bit serial/parallel multipliers are implemented on Virtex-6 FPGA and characterized in terms of multiply-time, energy consumption per multiply-operation, and area occupation for *r* value varying from 2 to 64. Compared to a recent published algorithm, savings of 21%, 53%, 105% are respectively obtained in terms of speed, power, and area.**

Keywords—High-Radix Multiplication; Low-Power Multiplication; Multibit Recoding Multiplication; Partial Product Generator (PPG)

I. BACKGROUND AND MOTIVATION

The continuous refinement of the mostly-used design paradigm based on modified Booth algorithm [1] combined to a reduction tree (carry-save-adder array , Dadda,…) has reached saturation. In [2] only slight improvements are achieved. The proposal reduces the partial product number from *N*/2+1 to *N*/2 using different circuit optimization techniques of the critical path.

Theoretically, only the signed multibit recoding multiplication algorithm [3] is capable of a drastic reduction (*N/r*) of the partial product number, given that *r*+1 is the number of bits of the multiplier that are simultaneously treated (1≤*r*≤*N*). Unfortunately, this algorithm requires the pre-computation of a number of odd multiples of the multiplicand (until $(2^{r-1}-1).X$) that scales linearly with *r*. The large number of odd multiples not only requires a considerable amount of multiplexers to perform the necessary complex recoding into PPG, but dramatically increases the routing density as well. Therefore, a reverse effect occurs that offsets speed and power benefits of the compression factor (*N/r*). This is the main reason why the multibit recoding algorithm was abandoned. In practice, designs do not exceed *r*=3 (radix-8).

This work is supported by "Centre de Développement des Technologies Avancées" (CDTA), Algiers, Algeria, in collaboration with FEMTO-ST Institute, Besançon, France.

The current trend [4][5] relies upon advanced arithmetic to determine minimal number bases that are representatives of the digits resulting from larger multibit recoding. The objective is to eliminate information redundancy inside *r*+1 bit-length slices for a more compact PPG. This is achievable as long as no or just very few odd multiples are required.

In [4], Seidel et al. have introduced a secondary recoding of digits issued from an initial multibit recoding for 5≤*r*≤16. The recoding scheme is based on balanced complete residue system. Though it significantly reduces the number of partial products (*N/r* for 5≤*r*≤ 16), it requires some odd multiples for *r*≥8. While in [5], Dimitrov et al. have proposed a new recoding scheme based on double base number system for 6≤*r*≤11. The algorithm is limited to unsigned multiplication and requires a larger number of odd multiples.

Instead of looking for more effective number bases, which is a hard mathematical task, our approach consists in exploiting already existing odd-multiple *free* recoding algorithms (2^1, 2^2, 2^5, and 2^8) to recursively build up generalized odd-multiple *free* radix-2^r recoding schemes.

To achieve such a goal, the multibit recoding multiplication algorithm is revisited [3]. Its design space is extended by the introduction of a new recursive version that enables a hardware-friendly space-time partitioning of the multiplication problem. Depending on *r* value ranging from 2 to *N*, highly-scalable signed multipliers with various levels of parallelism and latencies can be systematically generated with insignificant control-complexity. The new algorithm has also the merit to recursively reduce the number of partial products (*N/r*) without any limit for the parameter *r* and any need for the odd multiples of the multiplicand. It also allows the combination of different recoding schemes proposed in the literature into the same architecture for better performances of the multiplier. Several higher radix (2^3-2^{32}) two's complement 64x64 bit serial/parallel multipliers based on combined recoding schemes are implemented on Virtex-6 FPGA and characterized in terms of speed, power, and area occupation for *r* value ranging from 2 to 64. Compared to a new signed version of Dimitrov et al. algorithm [5] and Seidel et al. algorithm [4], outstanding results are obtained with the new multibit recoding scheme for *r*=8 formed by the combination of Seidel algorithm (*r*=5), MacSorley algorithm (*r*=2) [1] and Booth algorithm (*r*=1) [6].

The respective savings are as follows: 21%, 53%, 105% and 8%, 52%, 63% are obtained in terms of multiply-time, energy consumption per multiply-operation, and total gate count, respectively.

The paper is organized as follows. Section I outlines the main requirement specifications for a generalized radix-2^r multiplication. Section II introduces the new recursive multibit recoding multiplication algorithm. A number of high-radix (2^3-2^{32}) variants of the new algorithm accompanied with their implementation results are presented in Section III.

II. THE NEW RECURSIVE MULTIBIT RECODING MULTIPLICATION ALGORITHM

The equation (2.1.2) of the original multibit recoding algorithm presented in [3] does not offer hardware visibility. Let us rewrite it in a simpler hardware-friendly form, as follows:

$$Y=\sum_{j=0}^{\frac{N}{r}-1}\left(y_{rj-1}+2^0\,y_{rj}+2^1\,y_{rj+1}+2^2\,y_{rj+2}+\cdots+2^{r-2}\,y_{rj+r-2}-2^{r-1}\,y_{rj+r-1}\right)2^{rj}=\sum_{j=0}^{\frac{N}{r}-1}Q_j\,2^{rj} \quad (1)$$

Where $y_{-1}=0$ and $r\in \mathrm{N}^*$. For simplicity purposes and without loss of generality, we assume that r is a divider of N.

In equation (1), the two's complement representation of the multiplier Y is split into N/r two's complement slices (Q_j), each of r+1 bit length. Each pair of two contiguous slices has one overlapping bit. In literature, equation (1) is referred to by radix-2^r equation, to which corresponds a digit set $D(2^r)$ such as $Q_j\in D(2^r)=\{-2^{r-1},\ldots,0,\ldots,2^{r-1}\}$. Thus, the multiplication between X and Y becomes: $X.Y=\sum_{j=0}^{\frac{N}{r}-1}X.Q_j.2^{rj}$ (2). Where each partial product can be expressed as follows: $X.Q_j.2^{rj}=(-1)^s.2^e.(m.X)$, with $m\in O_m(2^r)=\{1,3,\ldots,2^{r-1}-1\}$ such as $|O_m(2^r)|=2^{r-2}$. $O_m(2^r)$ represents the required set of odd-multiples of the multiplicand ($m.X$) for radix-2^r. Hence, the partial-product generation-process consists first in selecting one odd- multiple ($m.X$) among the whole set of pre-computed odd- multiples, which is then submitted to a hardwired shift of e positions, and finally conditionally complemented $(-1)^s$ depending on the bit sign s of Q_j term.. While lower $m.X$ can be obtained using just one addition ($3X$=$2X$+$1X$), the calculation of higher ones may require a number of computation steps ($11X$= $8X$+$2X$+$1X$).

To bypass the hard problem of odd-multiples, we exploit the fact that the two's complement multiplier Y on which equation (1) is applied, is composed of a series of two's complement digits (Q_j) on which equation (1) can be recursively applied again. Based on this observation, let us announce the two following theorems.

Theorem 1. *Any digit $Q_j\in D(2^r)$ can be represented in a combination of digits $P_i\in D(2^s)$, such as s is a divider of r.*

When theorem (1) is applied to equation (1), it gives:

$$Y=\sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{s}-1}P_{ji}\,2^{si}\right]2^{rj} \quad (3)\ ; \quad \text{where}$$

$P_{ji}\in D(2^s)=\{-2^{s-1},\ldots,0,\ldots,2^{s-1}\}$ with

$O_m(2^s)=\{1,3,\ldots,2^{s-1}-1\}$ such as $\frac{|O_m(2^r)|}{|O_m(2^s)|}=2^{ks}$ and

$$X.Y=\sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{s}-1}X.P_{ji}\,2^{si}\right]2^{rj} \quad (4)$$

Theorem 2. *Any digit $Q_j\in D(2^r)$ can be represented in a combination of digits P_i+T_i such as $P_i\in D(2^s)$ and $T_i\in D(2^t)$ with s+t a divider of r, and $t<s$.*

Likewise, when theorem (2) is applied to equation (1), we obtain: $Y=\sum_{j}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{s+t}-1}\left[P_{ji}+T_{ji}\,2^s\right]2^{(s+t)i}\right]2^{rj}$ (5). Where

$P_{ji}\in D(2^s)=\{-2^{s-1},\ldots,0,\ldots,2^{s-1}\}$ with

$O_m(2^s)=\{1,3,\ldots,2^{s-1}-1\}$ and

$T_{ji}\in D(2^t)=\{-2^{t-1},\ldots,0,\ldots,2^{t-1}\}$ with

$O_m(2^t)=\{1,3,\ldots,2^{t-1}-1\}$ such as $\frac{|O_m(2^r)|}{|O_m(2^{s+t})|}=2^{k(s+t)}$

and $X.Y=\sum_{j}^{\frac{n}{r}-1}\left[\sum_{i=0}^{\frac{r}{s+t}-1}\left[X.P_{ji}+X.T_{ji}\,2^s\right]2^{(s+t)i}\right]2^{rj}$ (6)

Theorem (1) and (2) allow an exponential reduction ($1/2^{ks}$ and $1/2^{k(s+t)}$, resp.) of the number of odd-multiples in equations (4) and (6) in comparison to equation (2), but at the expense of a linear augmentation (ks-1 and $k(s+t)$-1, resp.) in the number of additions. The advantage by far outweighs the cost, as practically shown in the next section.

The translation of equation (4) into architecture is depicted by Fig. 1, where each PPG_j (Q_j) is built up using identical PPG_{ji} (P_{ji}). This is not the case for equation (6) which requires two different PPG_{ji} (P_{ji} and T_{ji}). Theorem (1) and (2) can be merged together to produce PPG_j made of a number of different PPG_{ji} (P_{ji},T_{ji}, U_{ji}, V_{ji},...). This is the general case that is thoroughly studied in the next section in order to determine the optimal multiplier.

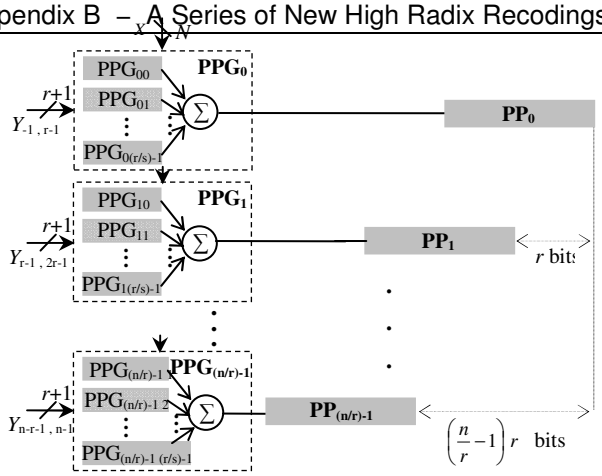


Figure. 1. Generalized $N\times N$ bit radix-2^r parallel multiplier based on sub-radix 2^s. Space partitioning according to r and s values.

III. SOME NEW HIGH RADIX (2^3-2^{32}) RECODING SCHEMES

Theorems (1) and (2) permit to build up any high radix-2^r multiplication algorithm based on lower sub-radices, employing much less odd-multiples. The objective is to generate high radix-2^r multiplication without odd-multiples for a maximum reduction of multiplexer complexity inside PPG_j. To achieve such a goal, a number of odd-multiple free low-radix algorithms are used, such as Booth algorithm [6] (radix-2^1), McSorley algorithm [1] (radix-2^2), Seidel et al. algorithms [4] (radix-2^5 and radix-2^8). The combination of these four algorithms enabled the generation of a series of higher radix recoding schemes (2^3-2^{32}) with minimum hardware resources (Table I). The generation process was manually guided by an heuristic (Table II) that evaluates the logic complexity (*Mux*) inside each PPG_j (Fig. 1).

The multipliers were mapped to Virtex-6 FPGA and characterized in terms of multiply-time, energy consumption per multiply-operation, and area occupation for r value varying from 2 to 64. The obtained results (Fig. 2, 3, and 4) showed an outstanding superiority of our algorithms over their recent counterparts [4][5]. When comparing our algorithms to each other, ß2^2 algorithm is the most area and energy efficient algorithm for any value of r (Table II). For r ranging from 8 to 64, ß"2^8 is the fastest algorithm, but it is outperformed by ß2^{32} for r values greater than 64. ß2^2 algorithm served to design a 16-bit set-point PID. The implementation results outperformed the published ones at all levels [7].

TABLE II
THEORETICAL ESTIMATION OF AREA OCCUPATION AND DELAY

Recoding Algorithm	Area Occupation		Delay (levels)		
	Mux	*Add*	*Mux Delay*	*PPG Adders*	*Linear Reduction Tree*
ß2^2	5r	r/2	d_2	0	r/2
ß2^3	5r	(2/3)r	d_2	1	r/3
ß2^5	27r	(3/5)r	d_5	2	r/5
ß2^8	194r	(7/8)r	d_8	6	r/8
ß'2^8	520r	r/4	d'_8	1	r/8
ß"2^8	19r	(5/8)r	d_5	4	r/8
ß2^{13}	130r	(10/13)r	d_8	9	r/13
ß2^{16}	100r	(11/16)r	d_8	10	r/16
ß2^{24}	74r	(16/24)r	d_8	15	r/24
ß2^{32}	60r	(21/32)r	d_8	20	r/32

Mux is an heuristic measure of the multiplexer logic inside PPG_i. *Add is* the exact umber of adders. d_i is the delay due to *Mux* logic ($d_2 < d_5 < d_8 < d'_8$)

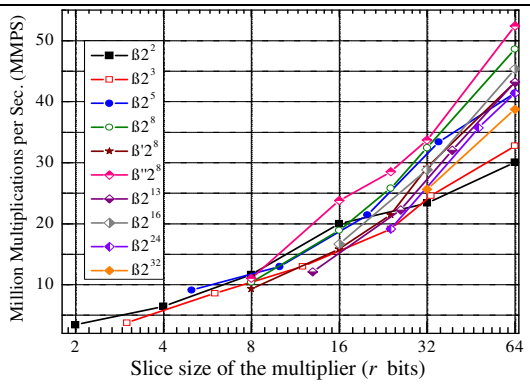


Figure 2. Max. multiply time versus r.

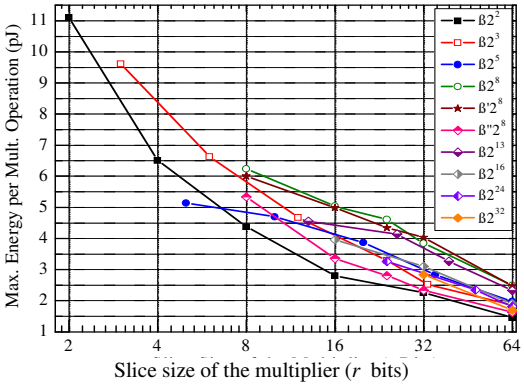


Figure 3. Max. energy consumption per mult. operation versus r.

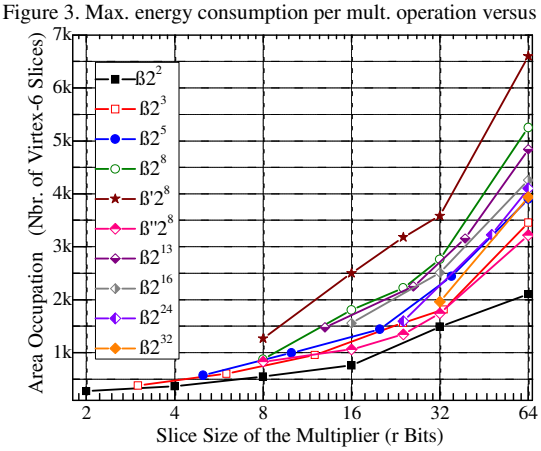


Figure. 4. Area occupation versus r

REFERENCES

[1] O.L. MacSorley, "High-Speed Arithmetic in Binary Computers," Proceedings of the IRE, Vol. 49(1), pp. 67-91, January 1961.

[2] F. Lamberti, "Reducing the Computation Time in (Short Bit-Width) Two's Complement Multiplier," IEEE Trans. on Computers, vol. 60, N° 2, pp. 148-156, February 2011.

[3] H. Sam, and A. Gupta, "A Generalized Multibit Recoding of Two's Complement Binary Numbers and its Proof with Application in Multiplier Implementation," IEEE Trans. on Computers, vol. 39, N° 8, August 1990.

[4] P.M. Seidel et al., "Secondary Radix Recodings for Higher Radix Multipliers," IEEE Trans. on Computers, vol. 54, N°2, February 2005.

[5] V.S. Dimitrov et al., "Area Efficient Multipliers Based on Multiple-Radix Representations," IEEE Trans. on Computers, vol. 60, N° 2, pp 189-201, February 2011.

[6] A. D. Booth, "A Signed Binary Multiplication Te:chnique," Quarterly J. Mech. Appl. Math., Vol. 4, part 2, pp. 236-240,1951.

[7] A.K. Oudjida et al., "High-Speed and Low-Power PID Structures for Embedded Applications" Proceedings of the 21th edition of the International Workshop on Power and Timing Modeling, Optimization and Simulation PATMOS, LNCS 6951, pp. 257-266, Springer-Verlag Editor. Madrid, Spain, Sep. 26-29 2011.

TABLE I
SUMMARY OF OUR NEW RADIX-2^r MULTIBIT RECODING ALGORITHMS

Recoding Algorithm	Recoding Equation and Main Features
$\beta 2^r$	$Y = \sum_{j=0}^{\frac{N}{r}-1} Q_j . 2^{rj}$; BR: 2^r ; OM: $\{1,3,...,2^{r-1}-1\}$; DV: $Q_j \in \{-2^{r-1},...,0,...,2^{r-1}\}$;
$\beta 2^2$	$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{2}-1} Q_{ji}\, 2^{2i} \right] . 2^{rj}$; BR: 2^2 ; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$;
$\beta 2^3$	$Y = \sum_{j}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{3}-1} \left(Q_{ji} + P_{ji}\, 2^2\right).2^{3i} \right] . 2^{rj}$; BR: $2^1, 2^2$; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji} \in \{-1,0,1\}$
$\beta 2^5$	$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{5}-1} \left(7 . Q_{ji} + P_{ji}\right) 2^{5i} \right] . 2^{rj}$; BR: 2^5 ; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji} \in \{-4,-2,-1,0,1,2,4\}$
$\beta 2^8$	$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{8}-1} \left(11^2 . Q_{ji} + 11 . P_{ji} + T_{ji}\right) 2^{8i} \right] . 2^{rj}$; BR: 2^8 ; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji}, T_{ji} \in \{-16,-8,-4,-2,-1,0,1,2,4,8,16\}$;
$\beta' 2^8$	$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{8}-1} \left(2^k . Q_{ji} + (-1)^e\, 2^h . P_{ji}\right) 2^{8i} \right] . 2^{rj}$ BR: 2^8 ; OM: $\{1,3,5,7\}$; DV: $Q_{ji}, P_{ji} \in \{1,3,5,7\}$ $k, h \in \{0,1,2,3,4,5,6,7\}$; $e \in \{0,1\}$
$\beta'' 2^8$	$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{8}-1} \left[\left(7 . Q_{ji} + P_{ji}\right) + \left(R_{ji} + S_{ji}\, 2^2\right).2^5 \right] . 2^{8i} \right] . 2^{rj}$ BR: $2^1, 2^2, 2^5$; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji} \in \{-4,-2,-1,0,1,2,4\}$, $R_{ji} \in \{-2,-1,0,1,2\}$, $S_{ji} \in \{-1,0,1\}$
$\beta 2^{13}$	$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{13}-1} \left[\left(11^2 . Q_{ji} + 11 . P_{ji} + T_{ji}\right) + \left(7 . R_{ji} + S_{ji}\right).2^8 \right] . 2^{13i} \right] . 2^{rj}$ BR: $2^5, 2^8$; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji}, T_{ji} \in \{-16,-8,-4,-2,-1,0,1,2,4,8,16\}$, $R_{ji} \in \{-2,-1,0,1,2\}$, $S_{ji} \in \{-4,-2,-1,0,1,2,4\}$
$\beta 2^{16}$	$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{16}-1} \left[\left(11^2 . Q_{ji} + 11 . P_{ji} + T_{ji}\right) + \left(\sum_{k=0}^{3} M_{kji} . 2^{2k} \right) . 2^8 \right] . 2^{16i} \right] . 2^{rj}$ BR: $2^2, 2^8$; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji}, T_{ji} \in \{-16,-8,-4,-2,-1,0,1,2,4,8,16\}$, $M_{kji} \in \{-2,-1,0,1,2\}$
$\beta 2^{24}$	$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{24}-1} \left[\left(11^2 . Q_{ji} + 11 . P_{ji} + T_{ji}\right) + \left(\sum_{k=0}^{3} M_{kji} . 2^{2k} \right) . 2^8 + \left(7 . R_{ji} + S_{ji}\right) 2^{16} + \left(U_{ji} + V_{ji}\, 2^2\right) . 2^{21} \right] . 2^{24i} \right] . 2^{rj}$ BR: $2^1, 2^2, 2^5, 2^8$; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji}, T_{ji} \in \{-16,-8,-4,-2,-1,0,1,2,4,8,16\}$, $M_{kji} \in \{-2,-1,0,1,2\}$, $R_{ji} \in \{-2,-1,0,1,2\}$, $S_{ji} \in \{-4,-2,-1,0,1,2,4\}$, $U_{ji} \in \{-2,-1,0,1,2\}$, $V_{ji} \in \{-1,0,1\}$
$\beta 2^{32}$	$Y = \sum_{j=0}^{\frac{N}{r}-1} \left[\sum_{i=0}^{\frac{r}{32}-1} \left[\left(11^2 . Q_{ji} + 11 . P_{ji} + T_{ji}\right) + \left(\sum_{k=0}^{3} M_{kji} . 2^{2k} \right) . 2^8 + \sum_{k=0}^{1} \left(\left(7 . R_{kji} + S_{kji}\right) 2^{16+8k} + \left(U_{kji} + V_{kji}\, 2^2\right) . 2^{21+8k} \right) \right] . 2^{32i} \right] . 2^{rj}$ BR: $2^1, 2^2, 2^5, 2^8$; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji}, T_{ji} \in \{-16,-8,-4,-2,-1,0,1,2,4,8,16\}$, $M_{kji} \in \{-2,-1,0,1,2\}$, $R_{kji} \in \{-2,-1,0,1,2\}$, $S_{kji} \in \{-4,-2,-1,0,1,2,4\}$, $U_{kji} \in \{-2,-1,0,1,2\}$, $V_{kji} \in \{-1,0,1\}$

BR: Based on Radix ; DV: Digit Variations ; OM: Odd-Multiples

Appendix C

PID Equations

Incremental Form

The standard version of the PID controller is described in a differential equation as:

$u(t) = K_p\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)\cdot d\tau + T_d \cdot \frac{de(t)}{dt}\right)$, where e is the system error ($e(t) = u_c(t) - y(t)$), u_c is the command signal (setpoint), y is the process variable (measured variable). K_p is the proportional gain, T_i the integration time constant, and T_d the derivative time constant of the controller. Using Laplace transform, $u(t)$ is expressed in s-domain by: $U(s) = K_p\left(E(s) + \frac{E(s)}{s\cdot T_i} + s\cdot T_d \cdot E(s)\right)$.

For a small sample interval T_s, the continuous time variable $u(t)$ can be discretized using the following approximations: $\int_0^{k\cdot T_s} e(t)\cdot dt \approx \sum_{j=0}^{k} e(j)\cdot T_s$; $\frac{de(t)}{dt} \approx \frac{e(k)-e(k-1)}{T_s}$. k denotes the k^{th} sampling instant ($k.T_s$). Thus, $u(t)$ can be rewritten as:

$$u(k) = K_p \cdot \left(e(k) + \frac{1}{T_i}\sum_{j=0}^{k} e(j)\cdot T_s + T_d \cdot \frac{e(k)-e(k-1)}{T_s}\right) \quad \text{with} \quad e(k) = u_c(k) - y(k) \quad \text{and}$$

$$u(k-1) = K_p\left(e(k-1) + \frac{1}{T_i}\sum_{j=0}^{k-1} e(j).T_s + T_d \cdot \frac{e(k-1)-e(k-2)}{T_s}\right).$$

We calculate the difference:

$$u(k) - u(k-1) = K_p \cdot (e(k) - e(k-1)) + \frac{K_p}{T_i}\left(\sum_{j=0}^{k} e(j)\cdot T_s - \sum_{j=0}^{k-1} e(j)\cdot T_s\right) + K_p \cdot T_d \cdot \left(\frac{e(k)-e(k-1)}{T_s} - \frac{e(k-1)-e(k-2)}{T_s}\right)$$

Developing separately each term of $u(k) - u(k-1)$, we obtain:

$$K_p \cdot (e(k) - e(k-1)) = K_p.e(k) - K_p \cdot e(k-1);$$

$$\frac{K_p}{T_i}\cdot\left(\sum_{j=0}^{k} e(j)\cdot T_s - \sum_{j=0}^{k-1} e(j)\cdot T_s\right) = K_p \cdot \frac{T_s}{T_i}\cdot e(k);$$

$$K_p \cdot T_d \cdot \left(\frac{e(k)-e(k-1)}{T_s} - \frac{e(k-1)-e(k-2)}{T_s} \right) = K_p \cdot \frac{T_d}{T_s} \cdot e(k);$$

$$- K_p \cdot \frac{2 \cdot T_d}{T_s} \cdot e(k-1) + K_p \cdot \frac{T_d}{T_s} \cdot e(k-2).$$

After simplifications, we get the following recurrent equation:

$$u(k) = u(k-1) + K_p \cdot \left(1 + \frac{T_s}{T_i} + \frac{T_d}{T_s} \right) \cdot e(k) - K_p \cdot \left(1 + 2.\frac{T_d}{T_s} \right) \cdot e(k-1) + K_p \cdot \frac{T_d}{T_s} \cdot e(k-2)$$

$$= u(k-1) + A \cdot e(k) + B \cdot e(k-1) + C \cdot e(k-2).$$

This latter equation is called the *incremental* form of the controller. A drawback with the incremental algorithm is that it cannot be used for P or PD controllers.

Commercial Form

For better performances of PID, two corrections are performed: limitation of the derivative gain and setpoint weighting. A pure derivative action will induce a very large amplification of measurement noise. The gain of the derivative must thus be limited. This can be done by approximating the transfer function $s.T_d$ as follows: $s \cdot T_d \approx \frac{s \cdot T_d}{1 + s \cdot T_d / N}$, where N is typically in the range of 3 to 20. In addition, to avoid sudden overshoots due to high variations of the setpoint, only a fraction b of u_c acts on the proportional part ($b.u_c - y$). Hence, the improved PID algorithm becomes:

$$U(s) = K_p \cdot \left((b \cdot U_c(s) - Y(s)) + \frac{1}{s \cdot T_i} \cdot (U_c(s) - Y(s)) - \frac{s \cdot T_d}{1 + s \cdot T_d / N} \cdot Y(s) \right).$$

$U(s)$ expression is discretized such that the proportional, integral and derivative terms are separately obtained, as follows: $u(k) = P(k) + I(k) + D(k)$, where

$P(k) = K_p \cdot b \cdot U_c(k) - K_p \cdot Y(k)$ and $I(k) = I(k-1) + K_p \cdot \frac{T_s}{T_i} \cdot (U_c(k-1) - Y(k-1))$.

To determine the derivative term $D(k)$, we use the differential equation representing the transfer function of $G_d(s)$: $G_d(s) = \frac{U_d(s)}{Y(s)} = -K_p \frac{s \cdot T_d}{1 + s \cdot T_d / N}$. By performing cross products, we get:

$$U_d(s) \cdot \left(1 + \frac{s \cdot T_d}{N} \right) = -K_p \cdot Y(s) \cdot s \cdot T_d.$$

Applying the inverse Laplace Transform to this latter equation, we obtain:

$$u_d(t) = -\frac{T_d}{N} \cdot \frac{du_d(t)}{dt} - K_p \cdot T_d \cdot \frac{dy_d(t)}{dt}.$$

Consequently, the discretized form of $u_d(t)$ is: $D(k) = -\frac{T_d}{N} \cdot \frac{D(k) - D(k-1)}{T_s} - K_p T_d \frac{Y(k) - Y(k-1)}{T_s}$.

After simplification, we obtain: $D(k) = \frac{T_d}{T_d + N \cdot T_s} D(k-1) - \frac{K \cdot N \cdot T_d}{T_d + N \cdot T_s} (Y(k) - Y(k-1))$.

Finally we can write: $u(k) = P(k) + I(k) + D(k)$ with

$P(k) = A \cdot u_c(k) + B \cdot y(k)$;

$I(k) = I(k-1) + C \cdot e(k-1)$;

$D(k) = H \cdot D(k-1) + L \cdot f(k)$ and

$A = K_p \cdot b$; $B = -K_p$; $C = -K_p \cdot \frac{T_s}{T_i}$; $H = \frac{T_d}{T_d + N \cdot T_s}$; $L = -\frac{K_p \cdot N \cdot T_d}{T_d + N \cdot T_s}$.

Appendix D

LQG Controller with Kalman Filter

Matlab Matriciel Model

```
clc;
clear all;
%% Simulation parameters
T0=1/50000;% Sampling period at start-up
Ts=1/20000;% Sampling period of simulation
tf=0.02;% Final time of simulation
long=(tf/Ts)+2;% Number of samples during simulation
t=[0:Ts:tf];% Time vector 1
tt=[0:Ts:tf tf]; % Time vector 2
sens=50.6; %% Sensitivity of the force sensor (µN/volts)

%% Coupled system model

%% Actuated system model
Aa=[1.6621,-0.9536;1,0];
Ba=[0.25;0];
Ca=[0.1104,0.1086];
Da=[0];
Ga=ss(Aa,Ba,Ca,Da,Ts);

%% Sensing system Model
Ac=[1.142174,-0.947527;1,0];
Bc=[0.5;0];
Cc=[0.13208,0.129656];
Dc=[0];
Gc=ss(Ac,Bc,Cc,Dc,Ts);

%% Global model of the FT-G100 micro-gripper
k0=1000; % Stiffness of the micro-object
k2=6.45;
r=0.1290;
Kk=(k2*k0)/(k2+k0);
A=[Aa zeros(2,2);Bc*Kk*Ca Ac];
B=[Ba;zeros(2,1)];
C=[zeros(1,2) r*Cc];
D=[0];
Gg=ss(A,B,C,D,Ts);

% Canonical form
    Gg=canon(Gg);
```

```
    [A,B,C,D]=ssdata(Gg);

%% LQ Control

%%Parameters of the optimal control
R=1;
Q=[1 0 0 0;0 10 0 0;0 0 1 0;0 0 0 1000];

%% Solving Riccati equation
[Pc,mat1,mat2]=dare(A,B,Q,R);

%%Gain of the optimal control
Kc=inv(R+(B'*Pc*B))*B'*Pc*A;% Gain of the optimal state-feedback

%% Prefiltre
L=inv(sens*C*inv(eye(4)-(A-B*Kc))*B);

%% Setpoint
fcc=10*ones(1,long+1);

%% Parameter initialisation
Vin(:,1)=0;

%% Kalman filter

%% Parameter initialisation
M=0.01*[1;1;1;1];
W= 9.2375e-005;% Variance of the state noise
V=1.7419e-005;% Variance of the measure noise
x(:,1)=[0;0;0;0];
xe(:,1)=[0;0;0;0];
pe=zeros(4,4);
Vin(:,1)=0;

%% Noise loading
load bre.txt;
load brm.txt;
load tb.txt; %Time vector

% State noise
    w=bre;
    w=[w w];
    w=w(1:long);

% Measure noise
    v=brm;
    v=[v v];
    v=v(1:long);

%% LQG controller loop
q=0;
 for i=0:Ts:tf;
    q=q+1;
```

```
        % Kalman discret model
         x(:,q+1)=(A*x(:,q))+(B*Vin(:,q))+(M*w(:,q));
         xa(:,q+1)=((C*x(:,q+1))+(v(:,q+1)));

         % Prediction
         pf=(A*pe*A')+(M*W*M');
         xp(:,q+1)=(A*xe(:,q))+(B*Vin(:,q));

        % Kalman gain
         ke(:,q+1)=(pf*C'*inv((C*pf*C')+V));

        % Estimation -Update-
         xe(:,q+1)=xp(:,q+1)+(ke(:,q+1)*(xa(:,q+1)-(C*xp(:,q+1))));
         pe=(eye(4)-(ke(:,q+1)*C))*pf;

        % Response of the controlled system
         xae(:,q)=(C*xe(:,q));

         % Control voltage  -Estimator form-
         Vin(:,q+1)=((L*fcc(:,q))-Kc*xe(:,q+1));
   end
 kef=ke(:,402);

%% Curve display
    figure(1);
    plot(tt,sens*xa);
    hold on;
    plot(t,sens*xae,'r');
    title('Noisy and filtered force (Fc) of the actuated arm');
    xlabel ('Time (s)');
    ylabel('Force(µN)');
    grid on;

    figure(2);
    plot(tt,Vin);
    title('Voltage control (Vin) at the input of the electrostatic actuator');
    xlabel ('Time (s)');
    ylabel('Voltage (Volts)');
    grid on;
```

Matlab Fixed-Point SCM/MCM Scalar Model

```
clc;
clear all;

%% Simulation parameters
T0=1/50000;% Sampling period at startup
Ts=1/20000;% Sampling period of simulation
tf=0.02;% Final time of simulation
long=(tf/Ts)+2;% Number of samples during simulation
t=[0:Ts:tf];% Time vector 1
tt=[0:Ts:tf tf]; % Time vector 2
```

```
sens=50.6; %% Stiffness of the force sensor (µN/volts)

%% Coupled system model

%% Actuated system model
Aa=[1.6621,-0.9536;1,0];
Ba=[0.25;0];
Ca=[0.1104,0.1086];
Da=[0];
Ga=ss(Aa,Ba,Ca,Da,Ts);

%% Sensing system  model
Ac=[1.142174,-0.947527;1,0];
Bc=[0.5;0];
Cc=[0.13208,0.129656];
Dc=[0];
Gc=ss(Ac,Bc,Cc,Dc,Ts);

% Fixed-point conversion parameters
Wl = 21; % Word length, integer+fractional parts
Fl = 16;   % Fractional part
S=2^Fl ;
ke1=[5.017862284983336e-04; -1.962693211189190e-05; 4.758862132834878e-04;-
3.039545010122170e-05];
ke = fi(ke1,1,Wl,Fl);
MW=9.2375e-09;

%% Global model of the FT-G100 micro-gripper
k0=1000;% Stiffness of the micro-object
k2=6.45;
r=0.1290;
Kk=(k2*k0)/(k2+k0);
A=[Aa zeros(2,2);Bc*Kk*Ca Ac];
B=[Ba;zeros(2,1)];
C=[zeros(1,2) r*Cc];
D=[0];
Gg=ss(A,B,C,D,Ts);

 %Canonical form
  Gg=canon(Gg);
   [A,B,C,D]=ssdata(Gg);

%% Declaration of the scalar elements of the matrices

%Matrice A
a11 = fi(A(1,1),1,Wl,Fl);
a12 = fi(A(1,2),1,Wl,Fl);
a13 = fi(A(1,3),1,Wl,Fl);
a14 = fi(A(1,4),1,Wl,Fl);

a21 = fi(A(2,1),1,Wl,Fl);
a22 = fi(A(2,2),1,Wl,Fl);
```

```
a23 = fi(A(2,3),1,WI,FI);
a24 = fi(A(2,4),1,WI,FI);

a31 = fi(A(3,1),1,WI,FI);
a32 = fi(A(3,2),1,WI,FI);
a33 = fi(A(3,3),1,WI,FI);
a34 = fi(A(3,4),1,WI,FI);

a41 = fi(A(4,1),1,WI,FI);
a42 = fi(A(4,2),1,WI,FI);
a43 = fi(A(4,3),1,WI,FI);
a44 = fi(A(4,4),1,WI,FI);

%Matrice B
b11 = fi(B(1,1),1,WI,FI);
b21 = fi(B(2,1),1,WI,FI);
b31 = fi(B(3,1),1,WI,FI);
b41 = fi(B(4,1),1,WI,FI);

%Matrice C
c11 = fi(C(1,1),1,WI,FI);
c12 = fi(C(1,2),1,WI,FI);
c13 = fi(C(1,3),1,WI,FI);
c14 = fi(C(1,4),1,WI,FI);

%Matrice XE
xe11(:,1) = fi(0,1,WI,FI);
xe21(:,1) = fi(0,1,WI,FI);
xe31(:,1) = fi(0,1,WI,FI);
xe41(:,1) = fi(0,1,WI,FI);

%% LQ Control

%% Parameters of the optimal control
%R=1;
%Q=[1 0 0 0;0 10 0 0;0 0 1 0;0 0 0 1000];

%%Solving Recatti equation
%[Pc,mat1,mat2]=dare(A,B,Q,R);

%%Gain of the optimal control
%Kc=inv(R+(B'*Pc*B))*B'*Pc*A;%Gain of the optimal state feedback
Kc1=[-0.000497622140588062,-0.0506923522114969,-0.970182175915640,1.60838698935252];
Kc = fi(Kc1,1,WI,FI);

%% Prefiltre
L=0.101783536095972;
%L = fi(L1,1,WI,FI);


%%Setpoint
fcc = 10;
Lfcc=L*fcc;
Lfcc= fi(Lfcc,1,WI,FI);
```

```
%% Parameter initialisation
Vin(:,1)=fi(0,1,Wl,Fl);

%% Kalman filter

% Parameter initialisation
M=0.01*[1;1;1;1];
W= 9.2375e-005;% Variance of the state noise
V=1.7419e-005;% Variance of the measure noise
x(:,1)=[0;0;0;0];

%% Noise loading
load bre.txt;
load brm.txt;
load tb.txt; %vecteur de temps

% State noise
    w=bre;
    w=[w w];
    w=w(1:long);

% Measure noise
    v=brm;
    v=[v v];
    v=v(1:long);

%% LQG control loop
q=0;
 for i=0:Ts:tf;
    q=q+1;

    % Kalman discret model
     x(:,q+1)=(A*x(:,q))+(B*double(Vin(:,q)))+(M*w(:,q));
     xa(:,q+1)=((C*x(:,q+1))+(v(:,q+1)));
     xa(:,q+1)= fi(xa(:,q+1),1,Wl,Fl);

     % Prédiction

     %)------------------------

     V =Vin(:,q);
     V3=2*V+V;
     V5=4*V+V;
     V7=8*V-V;

     ea1=xe11(q);
     ea3=2*ea1+ea1;
     ea7=8*ea1-ea1;

     eb1=xe21(q);
     eb3=2*eb1+eb1;
     eb7=8*eb1-eb1;

     ec1=xe31(q);
```

```
ec3=2*ec1+ec1;
ec5=4*ec1+ec1;

ed1=xe41(q);
ed3=2*ed1+ed1;
ed5=4*ed1+ed1;

%------------------------------

xp11(q+1)= fi((-V5*2^2+V*2^9)/S + (ea3*2^4+ea3-ea7*2^12+ea1*2^9+ea1*2^16)/S + (-eb3*2^4-
eb3-eb7*2^11+eb1*2^9+eb1*2^16)/S,1,Wl,Fl);
xp21(q+1)= fi((V7*2^2-V-V*2^14+V*2^9)/S + (eb3*2^4+eb3-eb7*2^12+eb1*2^9+eb1*2^16)/S - (-
ea3*2^4-ea3-ea7*2^11+ea1*2^9+ea1*2^16)/S,1,Wl,Fl);
xp31(q+1)= fi((V7*2^3-V+V*2^13+V*2^10)/S + (-ec1*2^6-ec3*2^8-ec5*2^11+ec1*2^16)/S +
(ed5*2^4-ed5*2-ed1*2^15+ed3*2^8+ed1*2^16)/S,1,Wl,Fl);
xp41(q+1)= fi((V5*2^4-V*2^15+V3*2^8+V*2^16)/S + (-ed1*2^6-ed3*2^8-ed5*2^11+ed1*2^16)/S -
(ec5*2^4-ec5*2-ec1*2^15+ec3*2^8+ec1*2^16)/S,1,Wl,Fl);

%------------------------------

pa1=fi(xp11(q+1),1,Wl,Fl);
pa3=fi(2*pa1+pa1,1,Wl,Fl);
pa5=fi(4*pa1+pa1,1,Wl,Fl);
pa7=fi(8*pa1-pa1,1,Wl,Fl);

pb1=xp21(q+1);
pb3=2*pb1+pb1;

pc1=xp31(q+1);
pc3=2*pc1+pc1;
pc5=4*pc1+pc1;

pd1=xp41(q+1);
pd3=2*pd1+pd1;

% Estimation -update-

xec = fi((pa7*2^3+pa3*2^10)/S - xa(q+1) + (pb3*2^4+pb3)/S + (-pc3*2^5-pc3+pc5*2^9)/S +
(pd1*2^6+pd1*2^2-pd3*2^8)/S,1,Wl,Fl);

xe11(q+1)= fi(xp11(q+1) - (xec+xec*2^5)/S,1,Wl,Fl);
xe21(q+1)= fi(xp21(q+1) + xec/S,1,Wl,Fl);
xe31(q+1)= fi(xp31(q+1) - (-xec+xec*2^5)/S,1,Wl,Fl);
xe41(q+1)= fi(xp41(q+1) + 2*xec/S,1,Wl,Fl);

%------------------------------

xae(:,q)= fi(((7*2^3+3*2^10)*xe11(q)/S + (3*2^4+3)*xe21(q)/S + (-3*2^5-3+5*2^9)*xe31(q)/S +
(2^6+2^2-3*2^8)*xe41(q)/S),1,Wl,Fl);
xae1(:,q)=double(xae(:,q));

%------------------------------

ee1=xe11(q+1);

ef1=xe21(q+1);
```

```
    ef3=fi(2*ef1+ef1,1,Wl,Fl);

    eg1=xe31(q+1);
    eg3=fi(2*eg1+eg1,1,Wl,Fl);

    eh1=xe41(q+1);
    eh3=fi(2*eh1+eh1,1,Wl,Fl);

    % Control voltage  -Estimator form-
     Vin(:,q+1)= fi((66705/S-((-ee1-ee1*2^5)/S+(ef3*2-ef1*2^8-ef3*2^10)/S+(-
eg3*2^5+eg1*2+eg1*2^11-eg1*2^16)/S+(-eh1-eh1*2^6-eh1*2^10-eh3*2^13+eh1*2^17)/S)),1,Wl,Fl);
    % Vin(:,q+1)= fi((Lfcc -( Kc(1,1)*xe11(q+1)+Kc(1,2)*xe21(q+1) +Kc(1,3)*xe31(q+1)  +
Kc(1,4)*xe41(q+1))),1,Wl,Fl);
     Vin1(:,q+1) =double(Vin(:,q+1));
end

%% Affichage des courbes

%% Curve display
  figure(1);
  plot(tt,sens*xa);
  hold on;
  plot(t,sens*xae,'r');
  title('Noisy and filtered force (Fc) of the actuated arm');
  xlabel ('Time (s)');
  ylabel('Force(µN)');
  grid on;

  figure(2);
  plot(tt,Vin);
  title('Voltage control (Vin) at the input of the electrostatic actuator');
  xlabel ('Time (s)');
  ylabel('Voltage (Volts)');
  grid on;
```

Binary Arithmetic for Finite-Word-Length Linear Controllers: MEMS Applications

Abstract: This thesis addresses the problem of optimal hardware-realization of finite-word-length (FWL) linear controllers dedicated to MEMS applications. The biggest challenge is to ensure satisfactory control performances with a minimal hardware. To come up, two distinct but complementary optimizations can be undertaken: in control theory and in binary arithmetic. Only the latter is involved in this work.

Because MEMS applications are targeted, the binary arithmetic must be fast enough to cope with the rapid dynamic of MEMS; power-efficient for an embedded control; highly scalable for an easy adjustment of the control performances; and easily predictable to provide a precise idea on the required logic resources before the implementation.

The exploration of a number of binary arithmetics showed that radix-2^r is the best candidate that fits the aforementioned requirements. It has been fully exploited to designing efficient multiplier cores, which are the real engine of the linear systems.

The radix-2^r arithmetic was applied to the hardware integration of two FWL structures: a linear time variant PID controller and a linear time invariant LQG controller with a Kalman filter. Both controllers showed a clear superiority over their existing counterparts, or in comparison to their initial forms.

Key-words: Finite-Word-Length Controllers, High-Speed and Low-Power Design, Radix-2^r arithmetic

Résumé: Cette thèse traite le problème d'intégration hardware optimale de contrôleurs linéaires à taille de mot finie, dédiés aux applications MEMS. Le plus grand défi est d'assurer des performances de contrôle satisfaisantes avec un minimum de ressources logiques. Afin d'y parvenir, deux optimisations distinctes mais complémentaires peuvent être entreprises: en théorie de contrôle et en arithmétique binaire. Seule cette dernière est considérée dans ce travail.

Comme cette arithmétique cible des applications MEMS, elle doit faire preuve de vitesse afin de prendre en charge la dynamique rapide des MEMS, à faible consommation de puissance pour un contrôle intégré, hautement reconfigurabe pour un ajustement facile des performances de contrôle, et facilement prédictible pour fournir une idée précise sur les ressources logiques nécessaires avant l'implémentation même.

L'exploration d'un certain nombre d'arithmétiques binaires a montré que l'arithmétique radix-2^r est celle qui répond au mieux aux exigences précitées. Elle a été pleinement exploitée afin de concevoir des circuits de multiplication efficaces, qui sont au fait, le véritable moteur des systèmes linéaires.

L'arithmétique radix-2^r a été appliquée à l'intégration hardware de deux structures linéaires à taille de mot finie: un contrôleur PID variant dans le temps et à un contrôleur LQG invariant dans le temps, avec un filtre de Kalman. Le contrôleur PID a montré une nette supériorité sur ses homologues existants. Quant au contrôleur LQG, une réduction très importante des ressources logiques a été obtenue par rapport à sa forme initiale non optimisée.

Mots-clès: Contrôleurs à Taille de Mot Finie, Circuits à Haute vitesse et Faible Consommation de Puissance, Arithmétique Radix-2^r

SPIM

■ École doctorale SPIM 16 route de Gray F - 25030 Besançon cedex

■ tél. +33 (0)3 81 66 66 02 ■ ed-spim@univ-fcomte.fr ■ www.ed-spim.univ-fcomte.fr

UFC UNIVERSITÉ DE FRANCHE-COMTÉ